aws

Panduan Pengguna untuk Aurora

Amazon Aurora

Amazon Aurora: Panduan Pengguna untuk Aurora

Table of Contents

Apa itu Aurora? 1
Model tanggung jawab bersama Amazon RDS 2
Cara kerja Amazon Aurora dengan Amazon RDS 2
Klaster DB Aurora 4
Versi Aurora 6
Basis data relasional yang tersedia di Aurora 6
Perbedaan nomor versi antara basis data komunitas dan Aurora 7
Versi mayor Amazon Aurora 8
Versi minor Amazon Aurora 20
Versi patch Amazon Aurora 21
Mempelajari apa yang baru di setiap versi Amazon Aurora 22
Menentukan versi basis data Amazon Aurora untuk klaster basis data Anda 22
Versi Amazon Aurora default 22
Peningkatan versi minor otomatis 22
Berapa lama versi mayor Amazon Aurora tetap tersedia 23
Seberapa sering Amazon Aurora versi minor dirilis 23
Berapa lama versi minor Amazon Aurora tetap tersedia 23
Dukungan jangka panjang untuk versi minor Amazon Aurora tertentu 24
Dukungan yang Diperluas Amazon RDS untuk versi Aurora tertentu 25
Mengontrol secara manual apakah dan kapan klaster basis data Anda akan ditingkatkan ke versi baru 25
Peningkatan Amazon Aurora wajib 26
Menguji klaster DB Anda dengan versi Aurora baru sebelum melakukan peningkatan 26
Wilayah dan Zona Ketersediaan 27
AWS Daerah 28
Zona Ketersediaan 36
Zona waktu lokal untuk klaster DB 37
Fitur Aurora yang didukung berdasarkan Wilayah dan mesin 43
Konvensi tabel 44
Deployment Blue/Green 44
Konfigurasi klaster Aurora 45
Aliran aktivitas basis data di Aurora 45
Mengekspor data klaster ke Amazon S3 54
Mengekspor data snapshot ke Amazon S3 55

Basis data global Aurora 56
Autentikasi basis data IAM di Aurora 65
Autentikasi Kerberos dengan Aurora 66
Machine Learning Aurora 72
Wawasan Performa dengan Aurora 80
Integrasi nol-ETL 90
Proksi Amazon RDS 91
Integrasi Secrets Manager 100
Aurora Serverless v2 100
Aurora Serverless v1 105
API Data RDS 109
Zero-downtime patching (ZDP) 116
Fitur asli mesin 116
Manajemen koneksi Aurora 117
Jenis titik akhir Aurora 118
Melihat titik akhir 121
Menggunakan endpoint klaster 121
Menggunakan titik akhir pembaca 122
Menggunakan titik akhir kustom 122
Membuat titik akhir kustom 126
Melihat titik akhir kustom 128
Mengedit titik akhir kustom 131
Menghapus titik akhir kustom 133
nd-to-end AWS CLIContoh E untuk titik akhir kustom 134
Menggunakan titik akhir instans 140
Titik akhir dan ketersediaan tinggi 141
Kelas instans DB 142
Jenis kelas instans DB 142
Mesin DB yang didukung 145
Menentukan dukungan kelas instans DB di Wilayah AWS 153
Spesifikasi perangkat keras 157
Penyimpanan dan keandalan Aurora 162
Gambaran umum penyimpanan Aurora 163
Konten volume klaster 163
Konfigurasi penyimpanan klaster Aurora 163
Bagaimana penyimpanan berubah ukuran 164

Penagihan data ... 166
Keandalan ... 166
Keamanan Aurora ... 169
Menggunakan SSL dengan klaster DB Aurora ... 170
Ketersediaan yang tinggi untuk Amazon Aurora ... 171
Ketersediaan data Aurora yang tinggi ... 171
Ketersediaan yang tinggi untuk instans DB Aurora ... 171
Ketersediaan yang tinggi di seluruh Wilayah AWS dengan basis data global Aurora ... 172
Toleransi kesalahan ... 173
Ketersediaan yang tinggi dengan Proksi Amazon RDS ... 175
Replikasi dengan Aurora ... 175
Replika Aurora ... 175
Aurora MySQL ... 177
Aurora PostgreSQL ... 178
Penagihan instans DB untuk Aurora ... 178
Instans DB Sesuai Permintaan ... 181
Instans DB terpesan ... 182
Menyiapkan lingkungan Anda ... 197
Mendaftar Akun AWS ... 197
Membuat pengguna administratif ... 198
Memberikan akses terprogram ... 199
Menentukan persyaratan ... 200
Memberikan akses ke klaster DB ... 202
Memulai ... 205
Membuat dan terhubung ke klaster DB Aurora MySQL ... 205
Prasyarat ... 207
Langkah 1: Buat instans EC2 ... 207
Langkah 2: Membuat klaster DB Aurora MySQL ... 213
(Opsional) Buat VPC, instans EC2, dan cluster Aurora MySQL menggunakan AWS CloudFormation ... 218
Langkah 3: Membuat klaster DB Aurora MySQL ... 220
Langkah 4: Hapus instans EC2 dan klaster DB ... 223
(Opsional) Hapus instans EC2 dan cluster DB yang dibuat dengan CloudFormation ... 224
(Opsional) Menghubungkan klaster DB Anda ke fungsi Lambda ... 225
Membuat dan terhubung ke klaster DB Aurora PostgreSQL ... 225
Prasyarat ... 227

Langkah 1: Buat instans EC2 ... 227
Langkaah 2: Membuat klaster DB Aurora PostgreSQL ... 233
(Opsional) Buat VPC, instans EC2, dan cluster Aurora PostgreSQL menggunakan AWS CloudFormation ... 238
Langkah 3: Membuat klaster DB Aurora PostgreSQL ... 240
Langkah 4: Hapus instans EC2 dan klaster DB ... 243
(Opsional) Hapus instans EC2 dan cluster DB yang dibuat dengan CloudFormation ... 244
(Opsional) Menghubungkan klaster DB Anda ke fungsi Lambda ... 244
Tutorial: Membuat server web dan klaster DB Amazon Aurora ... 246
Meluncurkan instans EC2 ... 247
Membuat klaster DB ... 253
Menginstal server web ... 264
Tutorial dan kode sampel ... 277
Tutorial dalam panduan ini ... 277
Tutorial dalam AWS panduan lain ... 278
AWS lokakarya dan portal konten lab untuk Aurora PostgreSQL ... 279
AWS lokakarya dan portal konten lab untuk ... 280
Tutorial dan kode sampel di GitHub ... 282
Bekerja dengan AWS SDK ... 282
Mengonfigurasi klaster DB Aurora Anda ... 284
Membuat klaster DB ... 285
Prasyarat ... 286
Membuat klaster DB ... 292
Pengaturan yang tersedia ... 302
Pengaturan yang tidak berlaku untuk klaster DB Aurora ... 324
Pengaturan yang tidak berlaku untuk instans DB Aurora ... 325
Membuat sumber daya dengan AWS CloudFormation ... 328
Aurora dan templat AWS CloudFormation ... 328
Pelajari selengkapnya tentang AWS CloudFormation ... 328
Menghubungkan ke klaster DB ... 329
Menghubungkan ke Aurora MySQL ... 330
Menghubungkan ke Aurora PostgreSQL ... 336
Pemecahan masalah koneksi ... 339
Bekerja dengan grup parameter ... 341
Ikhtisar grup parameter ... 341
Bekerja dengan grup parameter klaster DB ... 345

Bekerja dengan grup parameter DB 363
Membandingkan grup parameter DB 379
Menentukan parameter DB 380
Memigrasikan data ke klaster DB 385
Aurora MySQL 385
Aurora PostgreSQL 385
Membuat ElastiCache cache dari Amazon RDS 386
Ikhtisar pembuatan ElastiCache cache dengan pengaturan instans DB cluster Aurora DB ... 386
Membuat ElastiCache cache dengan pengaturan dari instance Aurora DB cluster DB 387
Mengelola klaster DB Aurora 390
Menghentikan dan memulai klaster 391
Ikhtisar menghentikan dan memulai klaster 391
Pembatasan 392
Menghentikan klaster DB 392
Saat klaster DB dihentikan 394
Memulai klaster DB 394
Menghubungkan sumber daya komputasi AWS 396
Menghubungkan instans EC2 396
Menghubungkan fungsi Lambda 407
Memodifikasi klaster DB Aurora 425
Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API 425
Memodifikasi instans DB dalam klaster DB 427
Mengubah kata sandi pengguna utama 430
Pengaturan yang tersedia 432
Pengaturan yang tidak berlaku untuk klaster DB Aurora 469
Pengaturan yang tidak berlaku untuk instans DB Aurora 470
Menambahkan Replika Aurora 472
Mengelola performa dan penskalaan 479
Penskalaan penyimpanan 479
Penskalaan instans 486
Penskalaan baca 486
Mengelola koneksi 486
Mengelola rencana eksekusi kueri 487
Mengkloning volume untuk klaster DB Aurora 488
Gambaran umum kloning Aurora 488
Batasan kloning Aurora 489

Cara kerja kloning Aurora 490
Membuat klon Aurora 493
Kloning lintas akun 503
Mengintegrasikan dengan layanan AWS 520
Aurora MySQL 520
Aurora PostgreSQL 520
Menggunakan Auto Scaling dengan Replika Aurora 521
Memelihara klaster DB Aurora 545
Melihat pemeliharaan Tertunda 546
Menerapkan pembaruan 548
Periode pemeliharaan 551
Menyesuaikan periode pemeliharaan untuk klaster DB 553
Peningkatan versi minor otomatis untuk klaster DB Aurora 555
Memilih frekuensi pembaruan pemeliharaan Aurora MySQL 559
Bekerja dengan pembaruan sistem operasi 560
Mem-boot ulang instans atau klaster DB Aurora 565
Mem-boot ulang instans DB dalam klaster Aurora 566
Mem-boot ulang klaster Aurora dengan ketersediaan baca 567
Mem-boot ulang klaster Aurora tanpa ketersediaan baca 569
Memeriksa waktu aktif untuk klaster dan instans Aurora 570
Contoh operasi boot ulang Aurora 573
Menghapus klaster dan instans Aurora 590
Menghapus klaster DB Aurora 590
Perlindungan penghapusan untuk klaster Aurora 598
Menghapus klaster Aurora yang berhenti 599
Menghapus klaster Aurora MySQL yang merupakan replika baca 599
Snapshot akhir saat menghapus klaster 599
Menghapus instans dari klaster DB Aurora 599
Memberi tag pada sumber daya RDS 602
Gambaran Umum 603
Menggunakan tag untuk kontrol akses dengan IAM 604
Menggunakan tag untuk menghasilkan laporan penagihan mendetail 605
Menambahkan, menampilkan daftar, dan menghapus tag 605
Menggunakan Editor AWS Tag 609
Menyalin tag ke snapshot klaster DB 609
Tutorial: Menggunakan tag untuk menentukan klaster DB Aurora yang akan dihentikan. 610

Bekerja dengan ARN 614
Membuat konsep ARN 614
Mendapatkan ARN yang sudah ada 621
Pembaruan Aurora 624
Mengidentifikasi versi Amazon Aurora Anda 624
Menggunakan RDS Extended Support 626
Biaya RDS Extended Support 627
Versi dengan RDS Extended Support 627
Aurora DB atau cluster global 627
Pertimbangan untuk RDS Extended Support 628
Aurora DB atau cluster global dengan RDS Extended Support 628
Melihat pendaftaran RDS Extended Support 630
Aurora DB atau cluster global 632
Pertimbangan untuk RDS Extended Support 632
Kembalikan Aurora DB atau cluster global dengan RDS Extended Support 633
Menggunakan Deployment Blue/Green untuk pembaruan basis data 635
Gambaran Umum Deployment Blue/Green Amazon RDS 636
Manfaat 637
Alur kerja 637
Mengizinkan akses 642
Pertimbangan 644
Praktik terbaik 646
Wilayah dan ketersediaan versi 647
Batasan 647
Membuat deployment blue/green 652
Mempersiapkan deployment blue/green 652
Menentukan perubahan 654
Membuat deployment blue/green 654
Melihat deployment blue/green 658
Mengganti deployment blue/green 662
Waktu habis switchover 663
Pagar pembatas switchover 663
Tindakan switchover 664
Praktik terbaik switchover 665
Memverifikasi CloudWatch metrik sebelum peralihan 666
Memantau kelambatan replika sebelum peralihan 667

Melakukan switchover pada deployment blue/green ... 667
Setelah switchover ... 670
Menghapus deployment blue/green ... 671
Mencadangkan dan memulihkan klaster DB Aurora ... 675
Gambaran umum pencadangan dan pemulihan ... 676
Cadangan ... 676
Jendela cadangan ... 677
Mempertahankan cadangan otomatis ... 680
Memulihkan data ... 684
Kloning basis data ... 685
Backtrack ... 685
Penyimpanan cadangan ... 686
Penyimpanan cadangan otomatis ... 686
Penyimpanan snapshot ... 686
Metrik CloudWatch untuk penyimpanan cadangan ... 687
Menghitung penggunaan penyimpanan cadangan ... 688
FAQ ... 689
Membuat snapshot klaster DB ... 692
Menentukan apakah snapshot tersedia ... 694
Memulihkan dari snapshot klaster DB ... 695
Grup parameter ... 695
Grup keamanan ... 696
Pertimbangan Aurora ... 696
Memulihkan dari snapshot ... 696
Menyalin snapshot klaster DB ... 700
Batasan ... 700
Retensi snapshot ... 701
Menyalin snapshot bersama ... 701
Menangani enkripsi ... 702
Penyalinan snapshot inkremental ... 702
Penyalinan lintas Wilayah ... 702
Grup parameter ... 703
Menyalin snapshot klaster DB ... 703
Berbagi snapshot klaster DB ... 715
Berbagi snapshot ... 716
Berbagi snapshot publik ... 719

Berbagi snapshot terenkripsi 721
Menghentikan berbagi snapshot 725
Mengekspor data klaster DB ke Amazon S3 727
Batasan 728
Ikhtisar pengeksporan data klaster DB 729
Menyiapkan akses ke bucket S3 730
Mengekspor data klaster DB ke S3 733
Memantau ekspor klaster DB 737
Membatalkan ekspor klaster DB 739
Pesan kegagalan 741
Memecahkan masalah kesalahan izin PostgreSQL 742
Konvensi penamaan file 743
Konversi data 743
Mengekspor data snapshot klaster DB ke Amazon S3 744
Batasan 745
Ringkasan pengeksporan data snapshot 746
Menyiapkan akses ke bucket S3 747
Mengekspor snapshot ke bucket S3 752
Performa ekspor di Aurora MySQL 756
Memantau ekspor snapshot 757
Membatalkan ekspor snapshot 759
Pesan kegagalan 760
Memecahkan masalah kesalahan izin PostgreSQL 762
Konvensi penamaan file 762
Konversi data 764
oint-in-time Pemulihan P 775
oint-in-time Pemulihan P dari cadangan otomatis yang dipertahankan 778
P oint-in-time pemulihan menggunakan AWS Backup 781
Menghapus snapshot klaster DB 788
Menghapus snapshot klaster DB 788
Tutorial: Memulihkan klaster DB dari snapshot 790
Memulihkan klaster DB menggunakan konsol 790
Memulihkan klaster DB menggunakan AWS CLI 795
Memantau metrik di klaster DB Aurora 802
Ikhtisar pemantauan 803
Rencana pemantauan 803

Garis dasar kinerja 803
Pedoman kinerja 804
Alat-alat pemantauan 805
Melihat status cluster 809
Melihat klaster DB 810
Melihat status klaster DB 816
Melihat status instans DB di klaster Aurora 820
Melihat dan menanggapi rekomendasi Amazon RDS 826
Melihat rekomendasi Amazon Aurora 828
Menanggapi rekomendasi Amazon Aurora 853
Melihat metrik di konsol Amazon RDS 863
Menampilkan metrik gabungan di konsol Amazon RDS 867
Memilih tampilan pemantauan baru di tab Pemantauan 867
Memilih tampilan pemantauan baru dengan Wawasan Performa di panel navigasi 868
Memilih tampilan lama dengan Wawasan Performa di panel navigasi 870
Membuat dasbor kustom dengan Wawasan Performa di panel navigasi 871
Memilih dasbor yang telah dikonfigurasi sebelumnya dengan Wawasan Performa di panel navigasi 874
Memantau Aurora dengan CloudWatch 876
Ikhtisar Amazon Aurora dan Amazon CloudWatch 877
Melihat CloudWatch metrik 879
Mengekspor metrik Performance Insights ke CloudWatch 884
Membuat alarm CloudWatch 890
Memantau muatan DB dengan Wawasan Performa 892
Ikhtisar Wawasan Performa 892
Mengaktifkan dan menonaktifkan Wawasan Performa 903
Mengaktifkan Skema Performa untuk Aurora MySQL 907
Kebijakan Wawasan Performa 913
Menganalisis metrik dengan dasbor Wawasan Performa 920
Melihat rekomendasi proaktif Performance Insights 955
Mengambil metrik dengan API Wawasan Performa 958
Mencatat panggilan Wawasan Performa menggunakan AWS CloudTrail 983
Menganalisis kinerja dengan DevOps Guru untuk RDS 987
Manfaat DevOps Guru untuk RDS 987
Bagaimana DevOps Guru untuk RDS bekerja 988
Menyiapkan DevOps Guru untuk RDS 990

Memantau ancaman dengan GuardDuty Perlindungan RDS 998
Memantau OS dengan Pemantauan yang Disempurnakan 1000
Ikhtisar Pemantauan yang Disempurnakan 1000
Menyiapkan dan mengaktifkan Pemantauan yang Ditingkatkan 1002
Melihat metrik OS di konsol RDS 1007
Melihat metrik OS menggunakan Log CloudWatch 1009
Referensi metrik Aurora 1011
CloudWatch metrik untuk Aurora 1011
Dimensi-dimensi CloudWatch untuk Aurora 1043
Ketersediaan metrik Aurora di konsol Amazon RDS 1044
CloudWatch metrik untuk Performance Insights 1048
Metrik penghitung untuk Wawasan Performa 1051
Statistik SQL untuk Wawasan Performa 1074
Metrik OS dalam Pemantauan yang Disempurnakan 1081
Memantau peristiwa, log, dan aliran aktivitas basis data 1089
Melihat log, peristiwa, dan stream di konsol Amazon RDS 1090
Memantau peristiwa Aurora 1095
Ikhtisar peristiwa untuk Aurora 1095
Melihat peristiwa Amazon RDS 1097
Bekerja dengan pemberitahuan peristiwa Amazon RDS 1101
Membuat aturan yang dipicu pada peristiwa Amazon Aurora 1127
Kategori peristiwa dan pesan peristiwa Amazon RDS 1131
Memantau log Aurora 1155
Melihat dan mencantumkan file log basis data 1155
Mengunduh file log basis data 1157
Melihat file log basis data 1158
Menerbitkan ke Log CloudWatch 1160
Membaca isi file log dengan menggunakan REST 1163
File log basis data MySQL 1165
File log basis data PostgreSQL 1174
Memantau panggilan API Aurora di CloudTrail 1184
Integrasi CloudTrail dengan Amazon Aurora 1184
Entri file log Amazon Aurora 1185
Memantau Aurora dengan Aliran Aktivitas Basis Data 1190
Ikhtisar 1190
Prasyarat jaringan Aurora MySQL 1194

Memulai aliran aktivitas basis data 1196
Mendapatkan status aliran aktivitas 1199
Menghentikan aliran aktivitas basis data 1200
Memantau aliran aktivitas 1201
Mengelola akses ke aliran aktivitas 1239
Menggunakan Aurora MySQL 1242
Gambaran umum Aurora MySQL 1243
Peningkatan performa Amazon Aurora MySQL 1243
Aurora MySQL dan data spasial 1244
Aurora MySQL versi 3 yang kompatibel dengan MySQL 8.0 1245
Aurora MySQL versi 2 yang kompatibel dengan MySQL 5.7 1293
Keamanan dengan Aurora MySQL 1296
Hak akses pengguna master Aurora MySQL 1297
Menggunakan TLS dengan klaster DB Aurora MySQL 1298
Memperbarui aplikasi untuk menggunakan sertifikat TLS baru 1307
Menentukan apakah ada aplikasi yang tersambung ke klaster DB Aurora MySQL menggunakan TLS 1308
Menentukan apakah klien memerlukan verifikasi sertifikat untuk terhubung 1308
Memperbarui penyimpanan kepercayaan aplikasi Anda 1309
Contoh kode Java untuk membangun koneksi TLS 1310
Menggunakan autentikasi Kerberos untuk Aurora MySQL 1312
Ikhtisar autentikasi Kerberos untuk Aurora MySQL 1313
Pembatasan 1314
Menyiapkan autentikasi Kerberos untuk Aurora MySQL 1315
Membuat koneksi dengan Aurora MySQL lewat autentikasi Kerberos 1326
Mengelola klaster DB di dalam domain 1329
Memigrasikan data ke Aurora MySQL 1331
Bermigrasi dari basis data MySQL eksternal ke Aurora MySQL 1336
Bermigrasi dari instans DB MySQL ke Aurora MySQL 1363
Mengelola Aurora MySQL 1388
Mengelola performa dan penskalaan untuk Amazon Aurora MySQL 1388
Melakukan backtracking klaster DB 1396
Menguji Amazon Aurora MySQL menggunakan kueri injeksi kesalahan 1417
Mengubah tabel di Amazon Aurora menggunakan DDL Cepat 1421
Menampilkan status volume untuk klaster DB Aurora 1428
Menyesuaikan Aurora MySQL 1430

Konsep penting untuk penyesuaian Aurora MySQL 1430
Menyesuaikan Aurora MySQL dengan peristiwa tunggu 1433
Menyesuaikan Aurora MySQL dengan status thread 1486
Menyesuaikan Aurora MySQL dengan wawasan proaktif Amazon DevOps Guru 1494
Kueri paralel untuk Aurora MySQL 1500
Ikhtisar kueri paralel 1501
Merencanakan klaster kueri paralel 1506
Membuat klaster kueri paralel 1507
Mengaktifkan dan menonaktifkan kueri paralel 1511
Meningkatkan klaster kueri paralel 1514
Penyempurnaan kinerja 1516
Membuat objek skema 1517
Memverifikasi penggunaan kueri paralel 1517
Memantau 1521
Kueri paralel dan konsep SQL 1528
Audit Lanjutan dengan Aurora MySQL 1550
Mengaktifkan Audit Lanjutan 1550
Melihat log audit 1554
Detail log audit 1554
Replikasi dengan Aurora MySQL 1556
Replika Aurora 1556
Opsi replikasi 1558
Performa replikasi 1559
Zero-downtime restart (ZDR) 1559
Mengonfigurasi filter replikasi 1562
Memantau replikasi 1569
Menggunakan penerusan tulis 1570
Replikasi lintas Wilayah 1589
Menggunakan replikasi log biner (binlog) 1605
Penggunaan replikasi berbasis GTID 1654
Mengintegrasikan Aurora MySQL dengan layanan AWS 1660
Mengotorisasi Aurora MySQL untuk mengakses layanan AWS 1661
Memuat data dari file teks di Amazon S3 1679
Menyimpan data ke dalam file teks di Amazon S3 1693
Menginvokasi fungsi Lambda dari Aurora MySQL 1704
Menerbitkan log Aurora MySQL ke Log CloudWatch 1715

Mode lab Aurora MySQL 1721
Fitur mode lab Aurora 1721
Praktik terbaik dengan Aurora MySQL 1723
Menentukan ke instans DB mana Anda terhubung 1724
Praktik terbaik untuk performa dan penskalaan Aurora MySQL 1724
Praktik terbaik untuk ketersediaan tinggi Aurora MySQL 1734
Rekomendasi untuk Aurora MySQL 1736
Memecahkan masalah kinerja Aurora MySQL 1744
AWS opsi pemantauan 1744
Alasan paling umum untuk masalah kinerja DB 1745
Beban kerja 1745
Pencatatan log 1750
Kinerja kueri 1752
Referensi Aurora MySQL 1758
Parameter konfigurasi 1758
Peristiwa tunggu 1831
Status thread 1837
Tingkat isolasi 1841
Petunjuk 1848
Prosedur tersimpan 1852
Tabel information_schema 1901
Pembaruan Aurora MySQL 1908
Nomor Versi dan Versi Khusus 1909
Bersiap untuk akhir masa pakai Aurora MySQL versi 2 1913
Bersiap untuk akhir masa pakai Aurora MySQL versi 1 1918
Meng-upgrade klaster DB Amazon Aurora MySQL 1921
Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 3 1959
Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 2 1959
Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 1 1959
Pembaruan mesin basis data untuk klaster Aurora MySQL Serverless 1959
Bug MySQL diperbaiki oleh pembaruan mesin basis data Aurora MySQL 1959
Kerentanan keamanan yang diperbaiki di Amazon Aurora MySQL 1959
Menggunakan Aurora PostgreSQL 1960
Lingkungan Pratinjau Basis Data 1961
Jenis kelas instans DB yang didukung 1962
Fitur yang tidak didukung di Lingkungan Pratinjau 1962

Membuat klaster DB baru di Preview Environment 1963
PostgreSQL versi 16 di lingkungan Pratinjau Basis Data 1965
Keamanan dengan Aurora PostgreSQL 1966
Memahami peran dan izin PostgreSQL 1967
Mengamankan data Aurora PostgreSQL dengan SSL/TLS 1983
Memperbarui aplikasi untuk sertifikat SSL/TLS baru 1993
Menentukan apakah aplikasi terhubung ke klaster DB Aurora PostgreSQL menggunakan SSL 1994
Menentukan apakah klien memerlukan verifikasi sertifikat agar dapat terhubung 1995
Memperbarui penyimpanan kepercayaan aplikasi Anda 1995
Menggunakan koneksi SSL/TLS untuk berbagai jenis aplikasi 1996
Menggunakan autentikasi Kerberos 1997
Kawasan dan ketersediaan versi 1998
Ikhtisar autentikasi Kerberos 1998
Menyiapkan 2000
Mengelola klaster DB di Domain 2013
Menghubungkan dengan autentikasi Kerberos 2015
Menggunakan grup keamanan AD untuk kontrol akses Aurora PostgreSQL 2018
Memigrasikan data ke Aurora PostgreSQL 2030
Memigrasikan instans DB RDS for PostgreSQL menggunakan snapshot 2031
Memigrasikan instans DB RDS for PostgreSQL menggunakan replika baca Aurora 2038
Meningkatkan performa kueri dengan Aurora Optimized Reads 2051
Gambaran umum Aurora Optimized Reads di PostgreSQL 2052
Penggunaan 2054
Kasus penggunaan 2054
Memantau 2055
Praktik terbaik 2057
Menggunakan Babelfish for Aurora PostgreSQL 2058
Batasan Babelfish 2060
Memahami konfigurasi dan arsitektur Babelfish 2061
Membuat klaster DB Babelfish for Aurora PostgreSQL 2100
Memigrasi basis data SQL Server ke Babelfish 2110
Autentikasi basis data dengan Babelfish for Aurora PostgreSQL 2120
Menghubungkan ke klaster DB Babelfish 2126
Menggunakan Babelfish 2138
Memecahkan Masalah Babelfish 2197

Menonaktifkan Babelfish 2199
Versi Babelfish 2200
Referensi Babelfish 2218
Mengelola Aurora PostgreSQL 2271
Penskalaan instans DB Aurora PostgreSQL 2272
Koneksi maksimum 2272
Batas penyimpanan sementara 2274
Halaman besar untuk Aurora PostgreSQL 2277
Menguji Amazon Aurora PostgreSQL menggunakan kueri injeksi kesalahan 2278
Menampilkan status volume untuk klaster DB Aurora 2283
Menentukan disk RAM untuk stats_temp_directory 2284
Mengelola file sementara dengan PostgreSQL 2285
Menyetel dengan peristiwa tunggu di Aurora PostgreSQL 2291
Konsep penting untuk penyetelan Aurora PostgreSQL 2292
Peristiwa tunggu Aurora PostgreSQL 2297
Client:ClientRead 2300
Client:ClientWrite 2303
CPU 2305
IO:BufFileRead dan IO:BufFileWrite 2311
IO:DataFileRead 2320
IO:XactSync 2335
IPC:DamRecordTxAck 2337
Lock:advisory 2338
Lock:extend 2341
Lock:Relation 2344
Lock:transactionid 2349
Lock:tuple 2352
LWLock:buffer_content (BufferContent) 2357
LWLock:buffer_mapping 2359
LWLock:BufferIO (IPC:BufferIO) 2361
LWLock:lock_manager 2364
LWLock: MultiXact 2368
Timeout:PgSleep 2372
Menyesuaikan Aurora PostgreSQL dengan wawasan proaktif Amazon DevOps Guru 2373
Basis data telah lama berjalan idle dalam koneksi transaksi 2373
Praktik terbaik dengan Aurora PostgreSQL 2377

Menghindari performa lambat, pengaktifan ulang otomatis, dan failover untuk instans DB Aurora PostgreSQL ... 2377
Mendiagnosis bloat tabel dan indeks ... 2378
Manajemen memori yang ditingkatkan dalam Aurora PostgreSQL ... 2382
Failover cepat ... 2383
Pemulihan cepat setelah failover ... 2395
Mengelola churn koneksi ... 2402
Menyetel parameter memori untuk Aurora PostgreSQL ... 2410
Menganalisis penggunaan sumber daya dengan CloudWatch metrik ... 2419
Menggunakan replikasi logis untuk upgrade versi mayor ... 2423
Pemecahan masalah penyimpanan ... 2432
Replikasi dengan Aurora PostgreSQL ... 2433
Aurora Replica ... 2434
Meningkatkan ketersediaan Aurora Replica ... 2435
Memantau replikasi ... 2436
Menggunakan replikasi logis ... 2437
Menggunakan Aurora PostgreSQL sebagai Basis Pengetahuan untuk Amazon Bedrock ... 2448
Prasyarat ... 2448
Mempersiapkan Aurora PostgreSQL menjadi Basis Pengetahuan ... 2449
Membuat basis pengetahuan di konsol Bedrock ... 2450
Mengintegrasikan Aurora PostgreSQL dengan layanan AWS ... 2451
Mengimpor data dari Amazon S3 ke Aurora PostgreSQL ... 2452
Mengekspor data PostgreSQL ke Amazon S3 ... 2472
Menginvokasi fungsi Lambda dari Aurora PostgreSQL ... 2489
Menerbitkan log PostgreSQL Aurora ke Log CloudWatch ... 2505
Memantau rencana eksekusi kueri untuk Aurora PostgreSQL ... 2516
Mengakses rencana eksekusi kueri menggunakan fungsi Aurora ... 2516
Referensi parameter untuk rencana eksekusi kueri Aurora PostgreSQL ... 2516
Mengelola rencana eksekusi kueri untuk Aurora PostgreSQL ... 2521
Gambaran umum manajemen rencana kueri Aurora PostgreSQL ... 2521
Praktik terbaik untuk manajemen rencana kueri Aurora PostgreSQL ... 2530
Memahami manajemen rencana kueri ... 2533
Mengambil rencana eksekusi Aurora PostgreSQL ... 2535
Menggunakan rencana terkelola Aurora PostgreSQL ... 2538
Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans ... 2543
Mengelola rencana eksekusi Aurora PostgreSQL ... 2544

Referensi ... 2551
Fitur lanjutan dalam Manajemen Rencana Kueri ... 2574
Menangani ekstensi dan pembungkus data asing ... 2588
Menggunakan dukungan ekstensi yang didelegasikan Amazon Aurora untuk PostgreSQL 2589
Mengelola objek besar secara lebih efisien dengan modul lo ... 2603
Mengelola data spasial dengan PostGIS ... 2606
Mengelola partisi dengan ekstensi pg_partman ... 2615
Menjadwalkan pemeliharaan dengan ekstensi pg_cron ... 2621
Menggunakan PGAudit untuk mencatat aktivitas database ... 2630
Menggunakan pglogical untuk menyinkronkan data ... 2644
Pembungkus data asing yang didukung ... 2658
Bekerja dengan Ekstensi Bahasa Tepercaya untuk PostgreSQL ... 2674
Terminologi ... 2675
Persyaratan untuk menggunakan Ekstensi Bahasa Tepercaya ... 2676
Menyiapkan Ekstensi Bahasa Tepercaya ... 2679
Ikhtisar Ekstensi Bahasa Tepercaya ... 2683
Membuat ekstensi TLE ... 2685
Menghapus ekstensi TLE dari basis data ... 2690
Meng-uninstal Ekstensi Bahasa Tepercaya ... 2691
Menggunakan hook PostgreSQL dengan ekstensi TLE ... 2692
Referensi fungsi untuk Ekstensi Bahasa Tepercaya ... 2698
Referensi hook untuk Ekstensi Bahasa Tepercaya ... 2712
Referensi Aurora PostgreSQL ... 2715
Kolasi Aurora PostgreSQL untuk EBCDIC dan migrasi mainframe lainnya ... 2715
Kolasi yang didukung di Aurora PostgreSQL ... 2717
Referensi fungsi Aurora PostgreSQL ... 2717
Parameter Aurora PostgreSQL ... 2773
Peristiwa tunggu Aurora PostgreSQL ... 2833
Pembaruan Aurora PostgreSQL ... 2862
Mengidentifikasi versi Amazon Aurora PostgreSQL ... 2862
Rilis Aurora PostgreSQL ... 2864
Versi ekstensi untuk Aurora PostgreSQL ... 2865
Meningkatkan klaster DB Amazon Aurora PostgreSQL ... 2865
Menggunakan rilis dukungan jangka panjang (LTS) ... 2891
Menggunakan basis data global Aurora ... 2894
Gambaran umum basis data global Aurora ... 2894

Keuntungan basis data global Amazon Aurora 2896
Kawasan dan ketersediaan versi 2896
Keterbatasan basis data global Aurora 2896
Memulai basis data global Aurora 2899
Persyaratan konfigurasi basis data global Amazon Aurora 2900
Membuat basis data global Aurora 2901
Menambahkan Wilayah AWS ke basis data global Aurora 2917
Membuat klaster DB Aurora tanpa kepala di Wilayah sekunder 2922
Menggunakan snapshot untuk basis data global Aurora Anda 2925
Mengelola basis data global Aurora 2927
Memodifikasi basis data global Aurora 2927
Memodifikasi parameter global basis data 2929
Menghapus klaster dari basis data global Aurora 2930
Menghapus basis data global Aurora 2933
Terhubung ke basis data global Aurora 2935
Menggunakan penerusan menulis dalam basis data global Aurora 2936
Menggunakan penerusan tulis di dalam Aurora MySQL 2937
Menggunakan penerusan tulis di Aurora PostgreSQL 2958
Menggunakan switchover atau failover dalam basis data global Aurora 2973
Memulihkan basis data global Aurora dari pemadaman yang tidak direncanakan 2974
Melakukan switchover untuk basis data global Aurora 2984
Mengelola RPO untuk basis data global berbasis Aurora PostgreSQL 2990
Memantau basis data global Aurora 2995
Memantau basis data global Aurora dengan Wawasan Performa 2997
Memantau basis data global Aurora dengan Aliran Aktivitas Basis Data 2998
Memantau basis data global berbasis Aurora MySQL 2998
Memantau basis data global berbasis Aurora PostgreSQL 3002
Menggunakan basis data global Aurora dengan layanan AWS lainnya 3005
Meningkatkan basis data global Amazon Aurora 3006
Peningkatan versi utama 3007
Peningkatan versi kecil. 3008
Menggunakan Proksi RDS 3010
Kawasan dan ketersediaan versi 3011
Kuota dan Batasan 3011
Batasan MySQL 3013
Batasan PostgreSQL 3013

Merencanakan lokasi penggunaan Proksi RDS 3014
Konsep dan terminologi Proksi RDS 3015
Ikhtisar konsep Proksi RDS 3016
Pengumpulan koneksi 3017
Keamanan 3018
Failover 3020
Transaksi 3021
Memulai dengan Proksi RDS 3022
Menyiapkan prasyarat jaringan 3022
Menyiapkan kredensial basis data di Secrets Manager 3026
Menyiapkan kebijakan IAM 3028
Membuat Proksi RDS 3031
Melihat Proksi RDS 3038
Terhubung melalui Proksi RDS 3040
Mengelola Proksi RDS 3043
Mengubah Proksi RDS 3044
Menambahkan pengguna basis data 3051
Pengubahan kata sandi basis data 3051
Koneksi klien dan basis data 3052
Mengonfigurasi pengaturan koneksi 3052
Menghindari penyematan 3056
Menghapus Proksi RDS 3060
Bekerja dengan titik akhir Proksi RDS 3061
Ikhtisar titik akhir proksi 3062
Menggunakan titik akhir pembaca dengan klaster Aurora 3063
Mengakses basis data Aurora di seluruh VPC 3068
Membuat titik akhir proksi 3069
Melihat titik akhir proksi 3072
Mengubah titik akhir proksi 3073
Menghapus titik akhir proksi 3074
Batasan untuk titik akhir proksi 3076
Memantau Proxy RDS dengan CloudWatch 3076
Bekerja dengan peristiwa Proksi RDS 3084
Peristiwa Proksi RDS 3085
Contoh Proksi RDS 3088
Memecahkan Masalah Proksi RDS 3090

Memverifikasi konektivitas untuk proksi ... 3091
Masalah dan solusi umum ... 3093
Penggunaan Proksi RDS dengan AWS CloudFormation ... 3101
Menggunakan Proksi RDS dengan basis data global Aurora ... 3101
Batasan untuk Proksi RDS dengan basis data global ... 3102
Cara kerja titik akhir Proksi RDS dengan basis data global ... 3102
Menggunakan integrasi nol-ETL ... 3104
Manfaat ... 3105
Konsep utama ... 3106
Batasan ... 3106
Batasan umum ... 3107
Batasan Aurora MySQL ... 3107
Batasan Aurora PostgreSQL pratinjau ... 3108
Batasan Amazon Redshift ... 3109
Kuota ... 3109
Wilayah yang Didukung ... 3110
Mulai menggunakan integrasi nol-ETL ... 3110
Langkah 1: Buat grup parameter klaster DB kustom ... 3111
Langkah 2: Buat cluster DB sumber ... 3112
Langkah 3: Buat gudang data Amazon Redshift ... 3113
Siapkan integrasi menggunakan AWS SDK (hanya Aurora MySQL) ... 3115
Langkah selanjutnya ... 3120
Membuat integrasi nol-ETL ... 3120
Prasyarat ... 3120
Izin yang diperlukan ... 3121
Membuat integrasi nol-ETL ... 3124
Langkah selanjutnya ... 3128
Pemfilteran data untuk integrasi nol-ETL ... 3128
Format filter data ... 3129
Filter logika ... 3131
Filter prioritas ... 3132
Contoh-contoh ... 3132
Menambahkan filter data ... 3133
Menghapus filter data ... 3135
Menambahkan dan mengueri data ... 3136
Buat basis data tujuan di Amazon Redshift ... 3136

Tambahkan data ke sumber DB cluster ... 3136
Kueri data Anda di Amazon Redshift ... 3138
Perbedaan jenis data ... 3139
Melihat dan memantau integrasi nol-ETL ... 3148
Melihat integrasi ... 3148
Pemantauan menggunakan tabel sistem ... 3150
Pemantauan EventBridge dengan ... 3151
Memodifikasi integrasi nol-ETL ... 3151
Menghapus integrasi nol-ETL ... 3152
Memecahkan masalah integrasi nol-ETL ... 3154
Saya tidak dapat membuat integrasi nol-ETL ... 3154
Integrasi saya dalam status `Syncing` permanen ... 3155
Satu atau beberapa tabel Amazon Redshift saya memerlukan sinkronisasi ulang ... 3155
Menggunakan Aurora Serverless v2 ... 3159
Kasus penggunaan Aurora Serverless v2 ... 3159
Mengonversi beban kerja terprovisi ... 3162
Keuntungan Aurora Serverless v2 ... 3162
Cara kerja Aurora Serverless v2 ... 3164
Gambaran umum ... 3164
Konfigurasi klaster ... 3166
Kapasitas ... 3167
Penskalaan ... 3168
Ketersediaan tinggi ... 3171
Penyimpanan ... 3172
Parameter konfigurasi ... 3172
Persyaratan dan batasan untuk Aurora Serverless v2 ... 3173
Kawasan dan ketersediaan versi ... 3173
Klaster yang menggunakan Aurora Serverless v2 harus memiliki rentang kapasitas yang ditentukan ... 3174
Beberapa fitur terprovisi tidak didukung di Aurora Serverless v2 ... 3174
Beberapa aspek Aurora Serverless v2 berbeda dari Aurora Serverless v1 ... 3175
Membuat klaster DB Aurora Serverless v2 ... 3175
Pengaturan ... 3176
Membuat klaster DB Aurora Serverless v2 ... 3177
Membuat penulis Aurora Serverless v2 ... 3181
Mengelola Aurora Serverless v2 ... 3182

Mengatur rentang kapasitas Aurora Serverless v2 untuk sebuah klaster 3182
Memeriksa rentang kapasitas Aurora Serverless v2 3187
Menambahkan pembaca Aurora Serverless v2 3189
Mengonversi dari terprovisi menjadi Aurora Serverless v2 3191
Mengonversi dari Aurora Serverless v2 menjadi terprovisi 3192
Memilih tingkat promosi untuk pembaca Aurora Serverless v2 3193
Menggunakan TLS/SSL dengan Aurora Serverless v2 3194
Melihat penulis dan pembaca Aurora Serverless v2 3196
Logging untuk Aurora Serverless v2 3197
Performa dan penskalaan untuk Aurora Serverless v2 3202
Memilih rentang kapasitas 3202
Menggunakan grup parameter untuk Aurora Serverless v2 3216
Menghindari out-of-memory kesalahan 3221
CloudWatch Metrik penting 3223
Memantau performa Aurora Serverless v2 dengan Wawasan Performa 3228
Memecahkan masalah kapasitas Aurora Serverless v2 3228
Migrasi ke Aurora Serverless v2 3229
Menggunakan Aurora Serverless v2 dengan klaster yang ada 3231
Beralih dari klaster terprovisi 3234
Perbandingan Aurora Serverless v2 dan Aurora Serverless v1 3240
Upgrade dari Aurora Serverless v1 ke Aurora Serverless v2 3250
Bermigrasi dari basis data on-premise ke Aurora Serverless v2 3253
Menggunakan Aurora Serverless v1 3254
Kawasan dan ketersediaan versi 3255
Keuntungan Aurora Serverless v1 3255
Kasus penggunaan untuk Aurora Serverless v1 3256
Batasan Aurora Serverless v1 3256
Persyaratan konfigurasi untuk Aurora Serverless v1 3259
Menggunakan TLS/SSL dengan Aurora Serverless v1 3259
Cipher suite yang didukung untuk koneksi ke klaster DB Aurora Serverless v1 3263
Cara kerja Aurora Serverless v1 3263
Aurora Serverless v1 arsitektur 3263
Penskalaan otomatis 3265
Tindakan batas waktu habis 3266
Jeda dan lanjutkan 3268
Menentukan max_connections 3268

Grup parameter 3271
Logging 3274
Pemeliharaan 3278
Failover 3279
Snapshot 3279
Membuat klaster DB Aurora Serverless v1 3280
Memulihkan klaster DB Aurora Serverless v1 3288
Memodifikasi klaster DB Aurora Serverless v1 3294
Memodifikasi konfigurasi penskalaan 3295
Meningkatkan versi mayor 3297
Mengonversi dari Aurora Serverless v1 menjadi terprovisi 3299
Menskalakan kapasitas klaster DB Aurora Serverless v1 secara manual 3302
Melihat klaster DB Aurora Serverless v1 3304
Memantau klaster DB Aurora Serverless v1 dengan CloudWatch 3307
Menghapus klaster DB Aurora Serverless v1 3308
Aurora Serverless v1 dan versi mesin basis data Aurora 3311
Aurora MySQL Serverless 3312
Aurora PostgreSQL Serverless 3312
Menggunakan RDS Data API 3313
Ketersediaan Wilayah dan versi 3314
Batasan 3314
Perbandingan dengan Serverless v2 dan provisioned, dan Aurora Serverless v1 3315
Memberikan otorisasi akses 3319
Otorisasi berbasis tag 3320
Menyimpan kredensial dalam rahasia 3322
Mengaktifkan API Data RDS 3323
Mengaktifkan RDS Data API saat Anda membuat database 3323
Mengaktifkan API Data RDS pada database yang ada 3325
Membuat titik akhir Amazon VPC 3328
Memanggil RDS Data API 3331
Memanggil RDS Data API dengan AWS CLI 3334
Memanggil RDS Data API dari aplikasi Python 3345
Memanggil RDS Data API dari aplikasi Java 3349
Menggunakan pustaka klien Java 3354
Mengunduh pustaka klien Java untuk API Data 3354
Contoh pustaka klien Java 3354

Memproses hasil kueri dalam format JSON 3356
Mengambil hasil kueri dalam format JSON 3357
Pemetaan Tipe Data 3357
Memecahkan masalah 3358
Contoh-contoh 3359
Memecahkan masalah API Data 3364
Transaksi <transaction_ID> tidak ditemukan 3364
Paket untuk kueri terlalu besar 3364
Respons basis data melebihi batas ukuran 3365
HttpEndpoint tidak diaktifkan untuk klaster <cluster_ID> 3365
Mencatat panggilan API Data RDS dengan AWS CloudTrail 3365
Bekerja dengan informasi API Data di CloudTrail 3366
Menyertakan dan mengecualikan peristiwa API Data dari jejak CloudTrail 3367
Memahami entri file log API Data 3369
Menggunakan editor kueri 3372
Ketersediaan editor kueri 3372
Mengotorisasi akses 3372
Menjalankan kueri 3374
Referensi API DBQMS 3378
CreateFavoriteQuery 3379
CreateQueryHistory 3379
CreateTab 3379
DeleteFavoriteQueries 3379
DeleteQueryHistory 3379
DeleteTab 3379
DescribeFavoriteQueries 3379
DescribeQueryHistory 3379
DescribeTabs 3380
GetQueryString 3380
UpdateFavoriteQuery 3380
UpdateQueryHistory 3380
UpdateTab 3380
Menggunakan machine learning Aurora 3381
Menggunakan machine learning Aurora dengan Aurora MySQL 3382
Persyaratan untuk menggunakan machine learning Aurora 3383
Wilayah dan ketersediaan versi 3384

Fitur yang didukung dan batasan 3385
Menyiapkan klaster Aurora Anda untuk machine learning Aurora 3385
Menggunakan Amazon Bedrock dengan cluster DB MySQL Aurora Anda 3399
Menggunakan Amazon Comprehend dengan klaster DB Aurora MySQL 3402
Menggunakan SageMaker dengan cluster DB MySQL Aurora Anda 3404
Pertimbangan performa 3408
Memantau 3410
Menggunakan machine learning Aurora dengan Aurora PostgreSQL 3411
Persyaratan untuk menggunakan machine learning Aurora 3412
Fitur yang didukung dan batasan 3413
Menyiapkan klaster DB Aurora untuk menggunakan machine learning Aurora 3414
Menggunakan Amazon Bedrock dengan cluster Aurora PostgreSQL DB Anda 3426
Menggunakan Amazon Comprehend dengan klaster DB Aurora PostgreSQL 3429
Menggunakan SageMaker dengan cluster DB PostgreSQL Aurora Anda 3431
Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model (Lanjutan) 3435
Pertimbangan performa 3436
Memantau 3442
Contoh kode 3444
Tindakan 3453
Membuat klaster DB 3454
Buat grup parameter klaster DB 3472
Membuat snapshot klaster DB 3482
Buat instans DB di klaster DB 3500
Menghapus klaster DB 3518
Menghapus grup parameter klaster DB 3532
Hapus instans basis data 3547
Menjelaskan grup parameter klaster DB 3561
Menjelaskan snapshot klaster DB 3568
Menjelaskan klaster DB 3575
Jelaskan instans basis data 3593
Jelaskan versi mesin basis data 3609
Jelaskan opsi untuk instans basis data 3619
Menjelaskan parameter dari grup parameter klaster DB 3629
Memperbarui parameter dalam grup parameter klaster DB 3641
Skenario 3651
Memulai dengan klaster DB 3651

Contoh lintas layanan 3819
Membuat API REST pustaka peminjaman 3820
Buat pelacak butir kerja Aurora Nirserver 3821
Praktik terbaik dengan Aurora 3826
Pedoman operasional dasar untuk Amazon Aurora 3826
Rekomendasi RAM instans DB 3827
Memantau Amazon Aurora 3828
Menggunakan grup parameter DB dan grup parameter klaster DB 3828
Video praktik terbaik Amazon Aurora 3829
Melakukan pembuktian konsep Aurora 3830
Gambaran umum bukti konsep Aurora 3830
1. Identifikasi tujuan Anda 3831
2. Pahami karakteristik beban kerja Anda 3832
3. Berlatih dengan konsol atau CLI 3833
Berlatih dengan konsol 3833
Berlatih dengan AWS CLI 3834
4. Buat klaster Aurora Anda 3835
5. Siapkan skema Anda 3836
6. Impor data Anda 3837
7. Lakukan porting kode SQL Anda 3838
8. Tentukan pengaturan konfigurasi 3839
9. Hubungkan ke Aurora 3839
10. Jalankan beban kerja Anda 3841
11. Ukur performa 3842
12. Coba ketersediaan tinggi Aurora 3845
13. Apa yang harus dilakukan selanjutnya 3847
Keamanan 3849
Autentikasi basis data 3851
Autentikasi kata sandi 3852
Autentikasi basis data IAM 3853
Autentikasi Kerberos 3853
Manajemen kata sandi dengan Aurora dan Secrets Manager 3855
Kawasan dan ketersediaan versi 3855
Batasan: 3855
Ikhtisar 3856
Manfaat 3856

Izin yang diperlukan untuk integrasi Secrets Manager 3857
Menerapkan manajemen Aurora 3858
Mengelola kata sandi pengguna utama untuk klaster DB 3859
Merotasi rahasia kata sandi pengguna utama untuk klaster DB 3863
Melihat detail tentang rahasia untuk klaster DB 3865
Perlindungan data 3868
Enkripsi data 3869
Privasi lalu lintas antarjaringan 3898
Pengelolaan identitas dan akses 3900
Audiens 3900
Mengautentikasi dengan identitas 3901
Mengelola akses menggunakan kebijakan 3905
Cara kerja Amazon Aurora dengan IAM 3907
Contoh kebijakan berbasis identitas 3915
AWS kebijakan terkelola 3934
Pembaruan kebijakan 3940
Pencegahan confused deputy lintas layanan 3949
Autentikasi basis data IAM 3951
Pemecahan Masalah 3995
Pencatatan dan pemantauan 3997
Validasi kepatuhan 4001
Ketahanan 4002
Pencadangan dan pemulihan 4002
Replikasi 4003
Failover 4003
Keamanan infrastruktur 4004
Grup keamanan 4004
Aksesibilitas publik 4004
Titik akhir VPC (AWS PrivateLink) 4006
Pertimbangan 4006
Ketersediaan 4006
Membuat titik akhir VPC antarmuka 4008
Membuat kebijakan titik akhir VPC 4008
Praktik terbaik keamanan 4009
Mengontrol akses dengan grup keamanan 4010
Ikhtisar grup keamanan VPC 4011

Skenario grup keamanan 4012
Membuat grup keamanan VPC 4013
Mengaitkan dengan klaster DB 4014
Hak akses akun pengguna master 4014
Peran tertaut layanan 4016
Izin peran tertaut layanan untuk Amazon Aurora 4016
Menggunakan Amazon Aurora dengan Amazon VPC 4020
Bekerja dengan instans DB dalam VPC 4020
Skenario untuk mengakses klaster DB di VPC 4036
Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4) 4043
Tutorial: Membuat VPC untuk digunakan dengan klaster DB (mode tumpukan ganda) 4051
Kuota dan batasan 4062
Kuota dalam Amazon Aurora 4062
Batasan penamaan dalam Amazon Aurora 4068
Batas ukuran Amazon Aurora 4069
Pemecahan Masalah 4071
Tidak dapat terhubung ke instans DB 4071
Menguji koneksi instans DB 4073
Memecahkan masalah autentikasi koneksi 4074
Masalah keamanan 4074
Pesan kesalahan “gagal mengambil atribut akun, fungsi konsol tertentu mungkin terganggu.” 4075
Mengatur ulang kata sandi pemilik instans DB 4075
Penghentian atau boot ulang instans DB 4075
Perubahan parameter tidak diberlakukan 4076
Masalah memori Aurora yang dapat dikosongkan 4076
Masalah Aurora MySQL out-of-memory 4078
Masalah replikasi Aurora MySQL 4079
Mendiagnosis dan mengatasi jeda di antara replika baca 4079
Mendiagnosis dan menyelesaikan kegagalan replikasi baca MySQL 4081
Kesalahan replikasi terhenti 4083
Referensi Amazon RDS API 4084
Menggunakan API Kueri 4084
Parameter Kueri 4084
Autentikasi permintaan Kueri 4085
Memecahkan masalah aplikasi 4085

Mengambil kesalahan 4085
Tips pemecahan masalah 4086
Riwayat dokumen 4087
AWS Glosarium 4175
..... mmmmclxxvi

Apa itu Amazon Aurora?

Amazon Aurora (Aurora) adalah mesin basis data relasional yang dikelola sepenuhnya dan kompatibel dengan MySQL dan PostgreSQL. Anda sudah tahu bagaimana MySQL dan PostgreSQL menggabungkan kecepatan dan keandalan basis data komersial kelas atas dengan kesederhanaan dan efektivitas biaya basis data sumber terbuka. Kode, alat, dan aplikasi yang Anda gunakan saat ini dengan basis data MySQL dan PostgreSQL Anda yang sudah ada dapat digunakan dengan Aurora. Dengan sejumlah beban kerja, Aurora dapat memberikan hingga lima kali throughput MySQL dan hingga tiga kali throughput PostgreSQL tanpa memerlukan perubahan pada sebagian besar aplikasi Anda saat ini.

Aurora mencakup subsistem penyimpanan performa tinggi. Mesin basis data-nya yang kompatibel dengan MySQL dan PostgreSQL disesuaikan untuk memanfaatkan penyimpanan terdistribusi cepat tersebut. Penyimpanan dasarnya meningkat secara otomatis sesuai kebutuhan. Volume klaster Aurora dapat bertambah ke ukuran maksimum 128 tebibyte (TiB). Aurora juga mengotomatiskan dan membakukan pengklasteran dan replikasi basis data, yang biasanya termasuk dalam aspek konfigurasi dan administrasi basis data yang paling menantang.

Aurora adalah bagian dari layanan basis data terkelola Amazon Relational Database Service (Amazon RDS). Amazon RDS memudahkan untuk mengatur, mengoperasikan, dan menskalakan database relasional di cloud. Jika Anda belum memahami tentang Amazon RDS, lihat Panduan pengguna Amazon Relational Database Service. Untuk mempelajari lebih lanjut tentang berbagai opsi basis data yang tersedia di Amazon Web Services, lihat Memilih basis data yang tepat untuk organisasi Anda pada AWS.

Topik

- Model tanggung jawab bersama Amazon RDS
- Cara kerja Amazon Aurora dengan Amazon RDS
- Klaster DB Amazon Aurora
- Versi Amazon Aurora
- Wilayah dan Zona Ketersediaan
- Fitur yang didukung di Amazon Aurora oleh Wilayah AWS dan mesin DB Aurora
- Manajemen koneksi Amazon Aurora
- Kelas instans DB Aurora
- Penyimpanan dan keandalan Amazon Aurora

- Keamanan Amazon Aurora
- Ketersediaan yang tinggi untuk Amazon Aurora
- Replikasi dengan Amazon Aurora
- Penagihan instans DB untuk Aurora

Model tanggung jawab bersama Amazon RDS

Amazon RDS bertanggung jawab untuk meng-host komponen perangkat lunak dan infrastruktur instans DB dan klaster DB. Anda bertanggung jawab untuk penyetelan kueri, yang merupakan proses menyesuaikan kueri SQL untuk meningkatkan performa. Performa kueri sangat bergantung pada desain basis data, ukuran data, distribusi data, beban kerja aplikasi, dan pola kueri, yang dapat sangat bervariasi. Pemantauan dan penyetelan adalah proses sangat khusus yang Anda miliki untuk basis data RDS Anda. Anda dapat menggunakan Wawasan Performa Amazon RDS dan alat lainnya untuk mengidentifikasi kueri bermasalah.

Cara kerja Amazon Aurora dengan Amazon RDS

Poin-poin berikut mengilustrasikan bagaimana Amazon Aurora berkaitan dengan mesin standar MySQL dan PostgreSQL yang tersedia di Amazon RDS:

- Anda memilih Aurora MySQL atau Aurora PostgreSQL sebagai opsi mesin DB saat mengatur server basis data baru melalui Amazon RDS.
- Aurora memanfaatkan fitur Amazon Relational Database Service (Amazon RDS) yang sudah Anda ketahui untuk manajemen dan administrasi. Aurora menggunakan AWS Management Console antarmuka Amazon RDS, AWS CLI perintah, dan operasi API untuk menangani tugas database rutin seperti penyediaan, penambalan, pencadangan, pemulihan, deteksi kegagalan, dan perbaikan.
- Operasi manajemen Aurora biasanya melibatkan seluruh klaster server basis data yang disinkronkan melalui replikasi, bukan instans basis data secara individual. Pengklasteran, replikasi, dan alokasi penyimpanan otomatis membuatnya sederhana dan hemat biaya untuk mengatur, mengoperasikan, dan menskalakan deployment MySQL dan PostgreSQL terbesar Anda.
- Anda dapat membawa data dari Amazon RDS for MySQL dan Amazon RDS for PostgreSQL ke Aurora dengan membuat dan memulihkan snapshot, atau dengan mengatur replikasi satu arah. Anda dapat menggunakan alat migrasi tombol-tekan untuk mengonversi aplikasi RDS for MySQL dan RDS for PostgreSQL Anda saat ini ke Aurora.

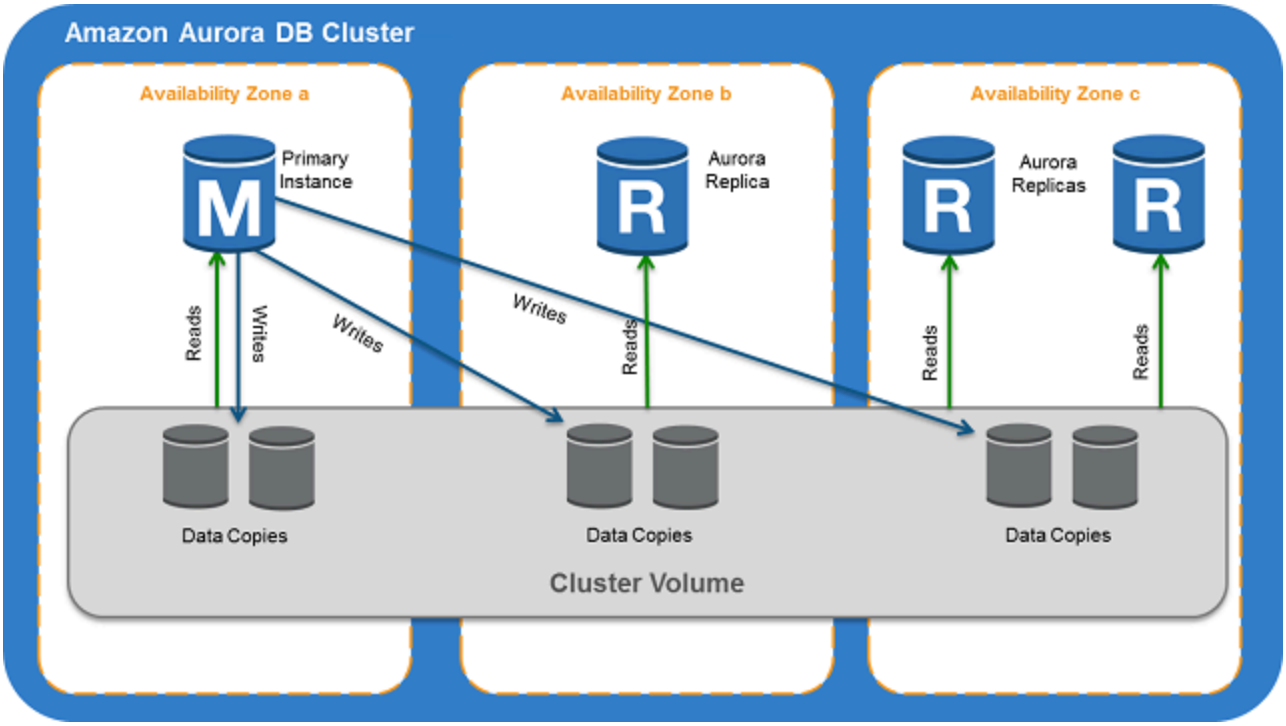
Sebelum menggunakan Amazon Aurora, selesaikan langkah-langkah dalam Menyiapkan lingkungan Anda untuk Amazon Aurora, lalu tinjau konsep dan fitur Aurora di Klaster DB Amazon Aurora.

Klaster DB Amazon Aurora

Klaster DB Amazon Aurora terdiri dari satu atau beberapa instans DB dan sebuah volume klaster yang mengelola data untuk instans DB tersebut. Volume klaster Aurora adalah volume penyimpanan basis data virtual yang mencakup beberapa Zona Ketersediaan, dengan masing-masing Zona Ketersediaan memiliki salinan data klaster DB. Ada dua jenis instans DB yang membentuk klaster DB Aurora:

- Instans DB primer – Mendukung operasi baca dan tulis, serta melakukan semua modifikasi data pada volume klaster. Setiap klaster DB Aurora memiliki satu instans DB primer.
- Replika Aurora – Menghubungkan ke volume penyimpanan yang sama seperti instans DB primer dan mendukung operasi hanya baca. Setiap klaster DB Aurora dapat memiliki hingga 15 Replika Aurora di samping instans DB primer. Ketersediaan tinggi dipertahankan dengan menempatkan Replika Aurora di Zona Ketersediaan terpisah. Aurora secara otomatis melakukan failover ke sebuah Replika Aurora jika instans DB primer tidak tersedia. Anda dapat menentukan prioritas failover untuk Replika Aurora. Replika Aurora juga dapat mengalihkan beban kerja baca dari instans DB primer.

Diagram berikut mengilustrasikan relasi antara volume klaster, instans DB primer, dan Replika Aurora dalam klaster DB Aurora.



> **Note**
>
> Informasi di atas berlaku untuk klaster terprovisi, klaster kueri paralel, klaster basis data global, klaster Aurora Serverless, dan semua klaster yang kompatibel dengan MySQL 8.0, 5.7, dan PostgreSQL.

Klaster Aurora mengilustrasikan pemisahan kapasitas komputasi dan penyimpanan. Misalnya, sebuah konfigurasi Aurora yang hanya memiliki satu instans DB tetap disebut sebagai klaster karena volume penyimpanan dasarnya memerlukan beberapa simpul penyimpanan yang didistribusikan ke beberapa Zona Ketersediaan (AZ).

Operasi input/output (I/O) di klaster DB Aurora dihitung dengan cara yang sama, terlepas dari apakah klaster tersebut berada di instans DB penulis atau pembaca. Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.

Versi Amazon Aurora

Amazon Aurora menggunakan kembali kode dan mempertahankan kompatibilitas dengan mesin DB MySQL dan PostgreSQL yang mendasarinya. Namun, Aurora memiliki atribut sendiri seperti nomor versi, siklus rilis, jadwal penghentian versi, dan sebagainya. Bagian berikut menjelaskan poin-poin umum dan perbedaannya. Informasi ini dapat membantu Anda memutuskan hal-hal seperti versi mana yang akan dipilih dan cara memverifikasi fitur dan perbaikan mana yang tersedia di setiap versi. Hal ini juga dapat membantu Anda memutuskan seberapa sering peningkatan dilakukan dan bagaimana rencana proses peningkatan Anda.

Topik

- Basis data relasional yang tersedia di Aurora
- Perbedaan nomor versi antara basis data komunitas dan Aurora
- Versi mayor Amazon Aurora
- Versi minor Amazon Aurora
- Versi patch Amazon Aurora
- Mempelajari apa yang baru di setiap versi Amazon Aurora
- Menentukan versi basis data Amazon Aurora untuk klaster basis data Anda
- Versi Amazon Aurora default
- Peningkatan versi minor otomatis
- Berapa lama versi mayor Amazon Aurora tetap tersedia
- Seberapa sering Amazon Aurora versi minor dirilis
- Berapa lama versi minor Amazon Aurora tetap tersedia
- Dukungan jangka panjang untuk versi minor Amazon Aurora tertentu
- Dukungan yang Diperluas Amazon RDS untuk versi Aurora tertentu
- Mengontrol secara manual apakah dan kapan klaster basis data Anda akan ditingkatkan ke versi baru
- Peningkatan Amazon Aurora wajib
- Menguji klaster DB Anda dengan versi Aurora baru sebelum melakukan peningkatan

Basis data relasional yang tersedia di Aurora

Basis data relasional berikut tersedia di Aurora:

- Amazon Aurora Edisi Kompatibel MySQL. Untuk informasi penggunaan, lihat Menggunakan Amazon Aurora MySQL. Untuk daftar lengkap versi yang tersedia, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL.
- Amazon Aurora Edisi Kompatibel PostgreSQL. Untuk informasi penggunaan, lihat Menggunakan Amazon Aurora PostgreSQL. Untuk daftar lengkap versi yang tersedia, lihat Pembaruan Amazon Aurora PostgreSQL.

Perbedaan nomor versi antara basis data komunitas dan Aurora

Setiap versi Amazon Aurora kompatibel dengan versi basis data komunitas tertentu dari MySQL atau PostgreSQL. Anda dapat menemukan versi komunitas basis data Anda menggunakan fungsi `version` dan versi Aurora menggunakan fungsi `aurora_version`.

Contoh untuk Aurora MySQL dan Aurora PostgreSQL ditampilkan sebagai berikut.

```
mysql> select version();
+------------------+
|   version()      |
+------------------+
|  5.7.12          |
+------------------+

mysql> select aurora_version(), @@aurora_version;
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 2.08.1           | 2.08.1           |
+------------------+------------------+
```

```
postgres=> select version();
-------------------------------------------------------------------------------
PostgreSQL 11.7 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.9.3, 64-bit
(1 row)

postgres=> select aurora_version();
aurora_version
----------------
3.2.2
```

Lihat informasi yang lebih lengkap di Memeriksa versi Aurora MySQL menggunakan SQL dan Mengidentifikasi versi Amazon Aurora PostgreSQL.

Versi mayor Amazon Aurora

Versi Aurora menggunakan skema *major*.*minor*.*patch*. Versi mayor Aurora mengacu pada versi mayor komunitas MySQL atau PostgreSQL yang kompatibel dengan Aurora. Versi mayor Aurora MySQL dan Aurora PostgreSQL tersedia berdasarkan dukungan standar setidaknya sampai akhir masa penggunaan komunitas untuk versi komunitas yang sesuai. Anda dapat terus menjalankan versi mayor melewati tanggal akhir dukungan standar Aurora dengan biaya tertentu. Untuk informasi selengkapnya, lihat Menggunakan Dukungan Diperpanjang Amazon RDS dan Harga Amazon Aurora.

Jika Amazon memperluas dukungan untuk versi Aurora lebih lama dari yang dinyatakan sebelumnya, kami akan memperbarui tabel ini untuk mencerminkan tanggal selanjutnya.

Versi mayor komunitas	Versi mayor Aurora	Tanggal akhir masa penggunaan komunitas	Tanggal akhir dukungan standar Aurora	Tanggal mulai harga tahun ke-1 Dukungan yang Diperluas RDS	Tanggal mulai harga tahun ke-3 Dukungan yang Diperluas RDS	Tanggal akhir Dukungan yang Diperluas RDS	Versi minor yang memenuhi syarat untuk Dukungan yang Diperluas RDS
MySQL 5.6 (sudah dihentikan)	Aurora MySQL versi 1 (usang)	5 Februari 2021	28 Februari 2023	N/A	N/A	N/A	N/A
MySQL 5.7	Aurora MySQL versi 2	Oktober 2023	31 Oktober 2024	1 Desember 2024	N/A	28 Februari 2027	Aurora MySQL 2.11 dan 2.12

Versi mayor komunitas	Versi mayor Aurora	Tanggal akhir masa penggunaan komunitas	Tanggal akhir dukungan standar Aurora	Tanggal mulai harga tahun ke-1 Dukungan yang Diperluas RDS	Tanggal mulai harga tahun ke-3 Dukungan yang Diperluas RDS	Tanggal akhir Dukungan yang Diperluas RDS	Versi minor yang memenuhi syarat untuk Dukungan yang Diperluas RDS
MySQL 8.0	Aurora MySQL versi 3	April 2026	30 April 2027	1 Mei 2027	N/A	31 Juli 2029	Akan ditentukan
PostgreSQL 9.6 (sudah dihentikan)	Aurora PostgreSQL 1 (sudah dihentikan)	11 November 2021	31 Januari 2022	N/A	N/A	N/A	N/A

Versi mayor komunitas	Versi mayor Aurora	Tanggal akhir masa penggunaan komunitas	Tanggal akhir dukungan standar Aurora	Tanggal mulai harga tahun ke-1 Dukungan yang Diperluas RDS	Tanggal mulai harga tahun ke-3 Dukungan yang Diperluas RDS	Tanggal akhir Dukungan yang Diperluas RDS	Versi minor yang memenuhi syarat untuk Dukungan yang Diperluas RDS
PostgreSQL 10 (sudah dihentikan)	Aurora PostgreSQL 2 (sudah dihentikan). Berlaku untuk PostgreSQL 10.17 dan versi yang lebih lama saja. Untuk versi 10.18 dan lebih tinggi, versi Aurora sama dengan versi	10 November 2022	31 Januari 2023	N/A	N/A	N/A	N/A

Versi mayor komunitas	Versi mayor Aurora	Tanggal akhir masa penggunaan komunitas	Tanggal akhir dukungan standar Aurora	Tanggal mulai harga tahun ke-1 Dukungan yang Diperluas RDS	Tanggal mulai harga tahun ke-3 Dukungan yang Diperluas RDS	Tanggal akhir Dukungan yang Diperluas RDS	Versi minor yang memenuhi syarat untuk Dukungan yang Diperluas RDS
	mayor.*min* untuk PostgreSQL versi komunitas, dengan digit ketiga di lokasi *patch*.						

Versi mayor komunitas	Versi mayor Aurora	Tanggal akhir masa penggunaan komunitas	Tanggal akhir dukungan standar Aurora	Tanggal mulai harga tahun ke-1 Dukungan yang Diperluas RDS	Tanggal mulai harga tahun ke-3 Dukungan yang Diperluas RDS	Tanggal akhir Dukungan yang Diperluas RDS	Versi minor yang memenuhi syarat untuk Dukungan yang Diperluas RDS
PostgreSQL 11	Aurora PostgreSQL 3. Berlaku untuk PostgreSQL 11.12 dan versi yang lebih lama saja. Untuk versi 11.13 dan lebih tinggi, versi Aurora sama dengan versi *mayor.min* untuk PostgreSQ	November 2023	29 Februari 2024	1 April 2024	1 April 2026	31 Maret 2027	Aurora PostgreSQL 11.9 dan 11.21

Versi mayor komunitas	Versi mayor Aurora	Tanggal akhir masa penggunaan komunitas	Tanggal akhir dukungan standar Aurora	Tanggal mulai harga tahun ke-1 Dukungan yang Diperluas RDS	Tanggal mulai harga tahun ke-3 Dukungan yang Diperluas RDS	Tanggal akhir Dukungan yang Diperluas RDS	Versi minor yang memenuhi syarat untuk Dukungan yang Diperluas RDS
	L versi komunitas, dengan digit ketiga di lokasi *patch*.						

Versi mayor komunitas	Versi mayor Aurora	Tanggal akhir masa penggunaan komunitas	Tanggal akhir dukungan standar Aurora	Tanggal mulai harga tahun ke-1 Dukungan yang Diperluas RDS	Tanggal mulai harga tahun ke-3 Dukungan yang Diperluas RDS	Tanggal akhir Dukungan yang Diperluas RDS	Versi minor yang memenuhi syarat untuk Dukungan yang Diperluas RDS
PostgreSQL 12	Aurora PostgreSQL 4. Berlaku untuk PostgreSQL 12.7 dan versi yang lebih lama saja. Untuk versi 12.8 dan lebih tinggi, versi Aurora sama dengan versi *mayor*.*min* untuk PostgreSQL versi	November 2024	28 Februari 2025	1 Maret 2025	1 Maret 2027	29 Februari 2028	Akan ditentukan

Versi mayor komunitas	Versi mayor Aurora	Tanggal akhir masa penggunaan komunitas	Tanggal akhir dukungan standar Aurora	Tanggal mulai harga tahun ke-1 Dukungan yang Diperluas RDS	Tanggal mulai harga tahun ke-3 Dukungan yang Diperluas RDS	Tanggal akhir Dukungan yang Diperluas RDS	Versi minor yang memenuhi syarat untuk Dukungan yang Diperluas RDS
	komunitas, dengan digit ketiga di lokasi *patch*.						

Versi mayor komunitas	Versi mayor Aurora	Tanggal akhir masa penggunaan komunitas	Tanggal akhir dukungan standar Aurora	Tanggal mulai harga tahun ke-1 Dukungan yang Diperluas RDS	Tanggal mulai harga tahun ke-3 Dukungan yang Diperluas RDS	Tanggal akhir Dukungan yang Diperluas RDS	Versi minor yang memenuhi syarat untuk Dukungan yang Diperluas RDS
PostgreSQL 13	Aurora PostgreSQL 13. Untuk versi 13.3 dan lebih tinggi, versi Aurora sama dengan versi *mayor*.*min* untuk PostgreSQL versi komunitas, dengan digit ketiga di lokasi *patch*.	November 2025	28 Februari 2026	1 Maret 2026	1 Maret 2028	28 Februari 2029	Akan ditentukan

Versi mayor komunitas	Versi mayor Aurora	Tanggal akhir masa penggunaan komunitas	Tanggal akhir dukungan standar Aurora	Tanggal mulai harga tahun ke-1 Dukungan yang Diperluas RDS	Tanggal mulai harga tahun ke-3 Dukungan yang Diperluas RDS	Tanggal akhir Dukungan yang Diperluas RDS	Versi minor yang memenuhi syarat untuk Dukungan yang Diperluas RDS
PostgreSQL 14	Aurora PostgreSQL 14.3 dan versi lebih tinggi. Versi Aurora sama dengan versi *mayor*.*min* PostgreSQL komunitas, dengan digit ketiga di lokasi *patch* saat patch ke Aurora dirilis.	November 2026	28 Februari 2027	1 Maret 2027	1 Maret 2029	28 Februari 2030	Akan ditentukan

Versi mayor komunitas	Versi mayor Aurora	Tanggal akhir masa penggunaan komunitas	Tanggal akhir dukungan standar Aurora	Tanggal mulai harga tahun ke-1 Dukungan yang Diperluas RDS	Tanggal mulai harga tahun ke-3 Dukungan yang Diperluas RDS	Tanggal akhir Dukungan yang Diperluas RDS	Versi minor yang memenuhi syarat untuk Dukungan yang Diperluas RDS
PostgreSQL 15	Aurora PostgreSQL 15.2 dan versi lebih tinggi. Versi Aurora sama dengan versi *mayor*.*min* PostgreSQL komunitas, dengan digit ketiga di lokasi *patch* saat patch ke Aurora dirilis.	November 2027	29 Februari 2028	1 Maret 2028	1 Maret 2030	28 Februari 2031	Akan ditentukan

Versi mayor komunitas	Versi mayor Aurora	Tanggal akhir masa penggunaan komunitas	Tanggal akhir dukungan standar Aurora	Tanggal mulai harga tahun ke-1 Dukungan yang Diperluas RDS	Tanggal mulai harga tahun ke-3 Dukungan yang Diperluas RDS	Tanggal akhir Dukungan yang Diperluas RDS	Versi minor yang memenuhi syarat untuk Dukungan yang Diperluas RDS
PostgreSQL 16	Aurora PostgreSQL 16.1 dan lebih tinggi. Versi Aurora sama dengan versi *mayor*.*min* PostgreSQL komunitas, dengan digit ketiga di lokasi *patch* saat patch ke Aurora dirilis.	9 November 2028	28 Februari 2029	Akan ditentukan	Akan ditentukan	Akan ditentukan	Akan ditentukan

> **Note**
>
> Amazon RDS Extended Support untuk Aurora MySQL versi 2 dimulai pada 1 November 2024, tetapi Anda tidak akan dikenakan biaya hingga 1 Desember 2024. Antara 1 November dan 30 November 2024, semua cluster DB Aurora MySQL versi 2 tercakup dalam Amazon RDS Extended Support.
> Amazon RDS Extended Support untuk PostgreSQL 11 dimulai pada 1 Maret 2024, tetapi Anda tidak akan dikenakan biaya hingga 1 April 2024. Antara 1 Maret dan 31 Maret 2024, semua cluster Aurora PostgreSQL versi 11 DB tercakup dalam Amazon RDS Extended Support.

Versi minor Amazon Aurora

Versi Aurora menggunakan skema *major*.*minor*.*patch*. Versi minor Aurora menyediakan peningkatan inkremental khusus komunitas dan Aurora pada layanan ini, misalnya fitur dan perbaikan baru.

Amazon Aurora saat ini mendukung versi MySQL berikut.

> **Note**
>
> Amazon RDS Extended Support tidak tersedia untuk versi minor.

Versi Aurora MySQL	Tanggal rilis Aurora MySQL	Aurora MySQL akhir dari tanggal dukungan standar
3.06 (Kompatibel dengan MySQL Komunitas 8.0.34)	Maret 2024	Mei 2025
3.05 (Kompatibel dengan Community MySQL 8.0.32)	Oktober 2023	Januari 2025
3.04[1] (Kompatibel dengan Community MySQL 8.0.28)	Juli 2023	Oktober 2026

Versi Aurora MySQL	Tanggal rilis Aurora MySQL	Aurora MySQL akhir dari tanggal dukungan standar
3.03 (Kompatibel dengan Community MySQL 8.0.26)	Maret 2023	Mei 2024
3.02 (Kompatibel dengan Community MySQL 8.0.23)	April 2022	Januari 2024
3.01 (Kompatibel dengan Community MySQL 8.0.23)	November 2021	Januari 2024
2.12[2] (Kompatibel dengan Community MySQL 5.7.40)	Juli 2023	Oktober 2024
2.11[2] (Kompatibel dengan Community MySQL 5.7.12)	Oktober 2022	Oktober 2024
2.07 (Kompatibel dengan Community MySQL 5.7.12)	November 2019	April 2024

[1] Versi dukungan jangka panjang (LTS) Aurora MySQL. Untuk informasi selengkapnya, lihat Rilis dukungan jangka panjang (LTS) Aurora MySQL.

[2] Versi minor ini akan terus tersedia ketika versi utama ada di Amazon RDS Extended Support. Untuk informasi selengkapnya, lihat Versi mayor Amazon Aurora.

Versi patch Amazon Aurora

Versi Aurora menggunakan skema *major*.*minor*.*patch*. Versi patch Aurora mencakup perbaikan penting yang ditambahkan ke versi minor setelah rilis awal (misalnya, Aurora MySQL 2.10.0, 2.10.1, ..., 2.10.3). Sementara setiap versi minor baru menyediakan fitur Aurora baru, versi patch baru dalam versi minor tertentu terutama digunakan untuk menyelesaikan masalah penting.

Untuk informasi selengkapnya tentang patching, lihat Memelihara klaster DB Amazon Aurora.

Mempelajari apa yang baru di setiap versi Amazon Aurora

Setiap versi Aurora baru dilengkapi dengan catatan rilis yang mencantumkan fitur baru, perbaikan, penyempurnaan lainnya, dan sebagainya yang berlaku untuk setiap versi.

Untuk catatan rilis Aurora MySQL, lihat Catatan Rilis untuk Aurora MySQL. Untuk catatan rilis Aurora PostgreSQL, lihat Catatan Rilis untuk Aurora PostgreSQL.

Menentukan versi basis data Amazon Aurora untuk klaster basis data Anda

Anda dapat menentukan versi apa pun yang tersedia saat ini (mayor dan minor) saat membuat cluster DB baru menggunakan operasi Buat database di AWS Management Console AWS CLI, operasi, atau `CreateDBCluster` API. Tidak semua versi basis data Aurora tersedia di setiap Wilayah AWS .

Untuk mempelajari cara membuat klaster Aurora, lihat Membuat klaster DB Amazon Aurora. Untuk mempelajari cara mengubah versi klaster Aurora yang ada, lihat Memodifikasi klaster DB Amazon Aurora.

Versi Amazon Aurora default

Ketika versi minor Aurora baru berisi peningkatan yang signifikan dibandingkan dengan yang sebelumnya, versi tersebut ditandai sebagai versi default untuk klaster DB baru. Biasanya, kami merilis dua versi default untuk setiap versi mayor per tahun.

Kami menyarankan agar klaster DB Anda ditingkatkan ke versi minor default terbaru karena versi tersebut berisi perbaikan keamanan dan fungsionalitas terbaru.

Peningkatan versi minor otomatis

Anda dapat terus menggunakan versi minor Aurora terbaru dengan mengaktifkan Peningkatan versi minor otomatis untuk setiap instans DB di klaster Aurora. Aurora hanya melakukan peningkatan otomatis jika pengaturan ini diaktifkan untuk semua instans di klaster DB Anda. Peningkatan versi minor otomatis dilakukan ke versi minor default.

Kami biasanya menjadwalkan peningkatan otomatis dua kali setahun untuk klaster DB yang memiliki pengaturan Peningkatan versi minor otomatis diatur ke `Yes`. Peningkatan ini dimulai selama periode pemeliharaan yang Anda tentukan untuk klaster Anda. Untuk informasi selengkapnya, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora.

Peningkatan versi minor otomatis dikomunikasikan terlebih dahulu melalui peristiwa klaster DB Amazon RDS dengan kategori `maintenance` dan ID `RDS-EVENT-0156`. Untuk informasi selengkapnya, lihat Kategori peristiwa dan pesan peristiwa Amazon RDS.

Berapa lama versi mayor Amazon Aurora tetap tersedia

Amazon Aurora versi mayor tetap tersedia setidaknya sampai akhir masa penggunaan komunitas untuk versi komunitas yang sesuai. Anda dapat menggunakan tanggal akhir dukungan standar Aurora untuk merencanakan siklus pengujian dan peningkatan Anda. Tanggal ini merepresentasikan tanggal paling awal saat peningkatan ke versi yang lebih baru mungkin diperlukan. Untuk informasi selengkapnya tentang tanggal, lihat Versi mayor Amazon Aurora.

Sebelum kami meminta Anda meningkatkan ke versi mayor yang lebih baru dan untuk membantu Anda merencanakan, kami memberikan pengingat setidaknya 12 bulan sebelumnya. Kami melakukannya untuk mengomunikasikan proses peningkatan terperinci. Detailnya mencakup waktu milestone tertentu, dampak pada klaster DB Anda, dan tindakan yang kami sarankan untuk Anda lakukan. Kami selalu menyarankan Anda menguji aplikasi Anda secara menyeluruh dengan versi basis data baru sebelum melakukan peningkatan versi mayor.

Setelah periode 12 bulan ini, peningkatan otomatis ke versi mayor berikutnya dapat diterapkan ke klaster basis data apa pun yang masih menjalankan versi yang lebih lama. Jika demikian, tingkatkan dimulai selama periode pemeliharaan terjadwal.

Seberapa sering Amazon Aurora versi minor dirilis

Secara umum, Amazon Aurora versi minor dirilis setiap tiga bulan. Jadwal rilis mungkin bervariasi untuk menyertakan fitur atau perbaikan tambahan.

Berapa lama versi minor Amazon Aurora tetap tersedia

Kami bermaksud untuk membuat setiap Amazon Aurora versi minor dari versi mayor tertentu tersedia setidaknya selama 12 bulan. Pada akhir periode ini, Aurora mungkin menerapkan peningkatan versi minor otomatis ke versi minor default berikutnya. Peningkatan semacam itu dimulai selama periode pemeliharaan terjadwal untuk klaster apa pun yang masih menjalankan versi minor yang lebih lama.

Kami mungkin mengganti versi minor dari versi mayor tertentu lebih cepat dari periode 12 bulan biasa jika ada masalah krusial seperti masalah keamanan, atau jika versi mayor tersebut telah mencapai akhir masa penggunaan.

Sebelum memulai peningkatan otomatis versi minor yang mendekati akhir masa penggunaan, kami biasanya memberikan pengingat tiga bulan sebelumnya. Kami melakukannya untuk mengomunikasikan proses peningkatan terperinci. Detailnya mencakup waktu milestone tertentu, dampak pada klaster DB Anda, dan tindakan yang kami sarankan untuk Anda lakukan. Notifikasi tiga bulan sebelumnya akan digunakan ketika ada hal-hal penting, seperti masalah keamanan, yang memerlukan tindakan lebih cepat.

Jika Anda tidak mengaktifkan pengaturan Peningkatan versi minor otomatis, Anda akan mendapatkan pengingat, tetapi tidak ada notifikasi peristiwa RDS. Peningkatan terjadi dalam periode pemeliharaan setelah batas waktu peningkatan wajib telah berlalu.

Jika Anda mengaktifkan pengaturan Peningkatan versi minor otomatis, Anda akan mendapatkan pengingat dan peristiwa klaster DB Amazon RDS dengan kategori `maintenance` dan ID `RDS-EVENT-0156`. Peningkatan terjadi selama periode pemeliharaan berikutnya.

Untuk informasi selengkapnya tentang peningkatan versi minor otomatis, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora.

Dukungan jangka panjang untuk versi minor Amazon Aurora tertentu

Untuk setiap versi utama Aurora, versi minor tertentu ditetapkan sebagai versi long-term-support (LTS) dan tersedia setidaknya selama tiga tahun. Artinya, setidaknya satu versi minor per versi mayor tersedia lebih lama dari 12 bulan yang biasa. Kami biasanya memberikan pengingat enam bulan sebelum akhir periode ini. Kami melakukannya untuk mengomunikasikan proses peningkatan terperinci. Detailnya mencakup waktu milestone tertentu, dampak pada klaster DB Anda, dan tindakan yang kami sarankan untuk Anda lakukan. Notifikasi enam bulan sebelumnya akan digunakan ketika ada hal-hal penting, seperti masalah keamanan, yang memerlukan tindakan lebih cepat.

Versi minor LTS hanya mencakup perbaikan kritis (melalui versi patch). Versi LTS tidak menyertakan fitur baru yang dirilis setelah diperkenalkan. Setahun sekali, klaster DB yang berjalan pada versi minor LTS di-patch ke versi patch terbaru untuk rilis LTS. Kami melakukan patching ini untuk membantu memastikan bahwa Anda mendapatkan manfaat dari perbaikan keamanan dan stabilitas kumulatif. Kami mungkin memberikan patch pada versi minor LTS lebih sering jika ada perbaikan penting, seperti untuk keamanan, yang perlu diterapkan.

> **Note**
>
> Jika Anda ingin tetap menggunakan versi minor LTS selama durasi siklus penggunaannya, pastikan untuk menonaktifkan Peningkatan versi minor otomatis untuk instans DB Anda. Untuk menghindari peningkatan klaster DB Anda secara otomatis dari versi minor LTS, tetapkan Peningkatan versi minor otomatis ke `No` pada instans DB apa pun di klaster Aurora Anda.

Untuk nomor versi untuk semua Aurora versi LTS, lihat Rilis dukungan jangka panjang (LTS) Aurora MySQL dan Rilis dukungan jangka panjang (LTS) Aurora PostgreSQL.

Dukungan yang Diperluas Amazon RDS untuk versi Aurora tertentu

Dengan Dukungan yang Diperluas Amazon RDS, Anda dapat terus menjalankan basis data Anda pada versi mesin mayor melewati tanggal akhir dukungan standar Aurora dengan biaya tambahan. Selama RDS Extended Support, Amazon RDS akan memasok patch untuk CVE Kritis dan Tinggi seperti yang didefinisikan oleh peringkat tingkat keparahan CVSS National Vulnerability Database (NVD). Untuk informasi selengkapnya, lihat Menggunakan Dukungan Diperpanjang Amazon RDS.

RDS Extended Support hanya tersedia pada versi Aurora tertentu. Untuk informasi selengkapnya, lihat Versi mayor Amazon Aurora.

Mengontrol secara manual apakah dan kapan klaster basis data Anda akan ditingkatkan ke versi baru

Peningkatan versi minor otomatis dilakukan ke versi minor default. Kami biasanya menjadwalkan peningkatan otomatis dua kali setahun untuk klaster DB yang memiliki pengaturan Peningkatan versi minor otomatis diatur ke `Yes`. Peningkatan ini dimulai selama periode pemeliharaan yang ditentukan pelanggan. Jika Anda ingin menonaktifkan peningkatan versi minor otomatis, tetapkan Peningkatan versi minor otomatis ke `No` pada instans DB apa pun dalam klaster Aurora. Aurora melakukan peningkatan versi minor otomatis hanya jika pengaturannya telah diaktifkan pada semua instans DB di klaster Anda.

Karena peningkatan versi mayor menimbulkan beberapa risiko kompatibilitas, tingkatkan ini tidak terjadi secara otomatis. Anda harus memulai peningkatan ini, kecuali dalam kasus penghentian versi mayor, seperti yang dijelaskan sebelumnya. Kami selalu menyarankan Anda menguji aplikasi Anda secara menyeluruh dengan versi basis data baru sebelum melakukan peningkatan versi mayor.

Untuk informasi selengkapnya tentang meningkatkan klaster DB ke versi mayor Aurora baru, lihat Meng-upgrade klaster DB Amazon Aurora MySQL dan Meningkatkan klaster DB Amazon Aurora PostgreSQL.

Peningkatan Amazon Aurora wajib

Untuk perbaikan kritis tertentu, kami mungkin melakukan peningkatan terkelola ke tingkat patch yang lebih baru dalam versi minor yang sama. Peningkatan wajib ini terjadi bahkan jika Peningkatan versi minor otomatis dinonaktifkan. Sebelum melakukannya, kami akan mengomunikasikan proses peningkatan yang terperinci. Detailnya mencakup waktu milestone tertentu, dampak pada klaster DB Anda, dan tindakan yang kami sarankan untuk Anda lakukan. Peningkatan terkelola tersebut dilakukan secara otomatis. Setiap peningkatan tersebut dimulai dalam periode pemeliharaan klaster.

Menguji klaster DB Anda dengan versi Aurora baru sebelum melakukan peningkatan

Anda dapat menguji proses peningkatan dan fungsi versi baru dengan aplikasi dan beban kerja Anda. Pilih salah satu metode berikut:

- Kloning klaster Anda menggunakan fitur klon basis data cepat Amazon Aurora. Lakukan peningkatan dan pengujian pasca-peningkatan pada klaster baru.
- Pulihkan dari snapshot klaster untuk membuat klaster Aurora baru. Anda dapat membuat snapshot klaster sendiri dari klaster Aurora yang ada. Aurora juga secara otomatis membuat snapshot berkala bagi Anda untuk setiap klaster Anda. Anda kemudian dapat memulai peningkatan versi untuk klaster baru tersebut. Anda dapat bereksperimen pada salinan klaster yang ditingkatkan sebelum memutuskan apakah akan meningkatkan klaster asli Anda.

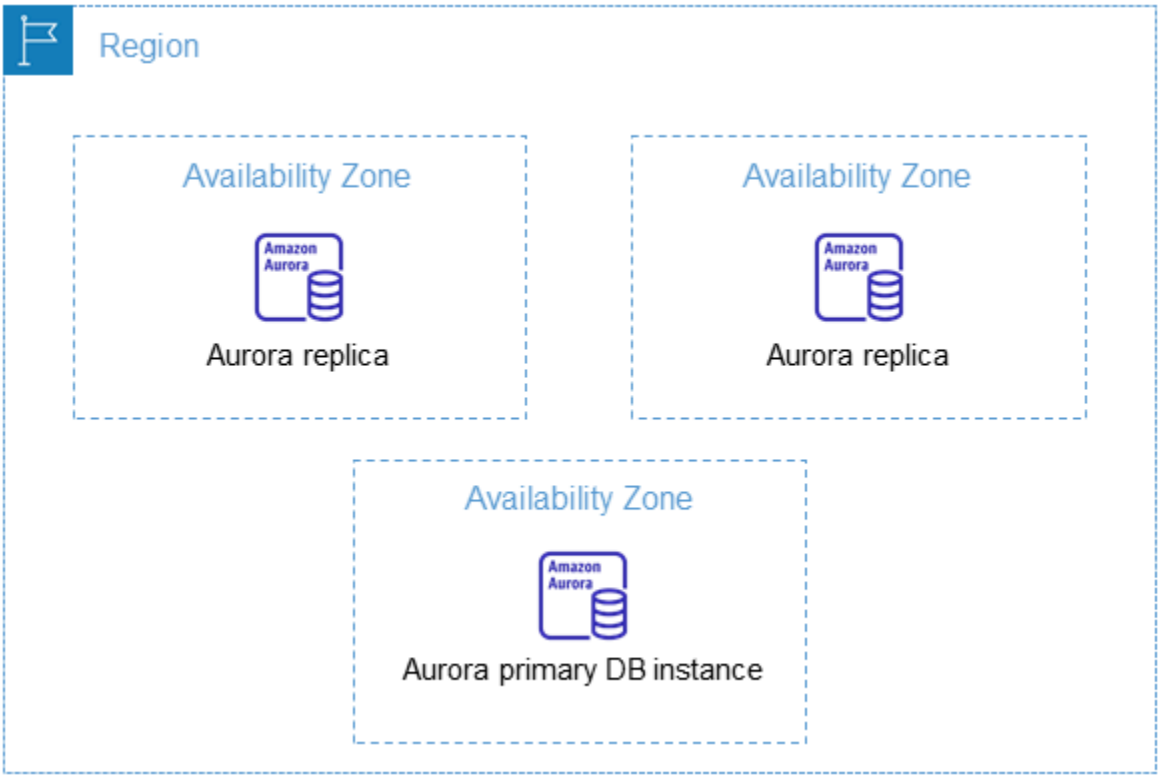
Untuk informasi selengkapnya tentang berbagai cara membuat klaster baru untuk pengujian ini, lihat Mengkloning volume untuk klaster DB Amazon Aurora dan Membuat snapshot klaster DB.

Wilayah dan Zona Ketersediaan

Sumber daya komputasi cloud Amazon di-hosting di beberapa lokasi di seluruh dunia. Lokasi ini terdiri dari AWS Wilayah dan Zona Ketersediaan. Setiap Wilayah AWS adalah wilayah geografis yang terpisah. Setiap AWS Wilayah memiliki beberapa lokasi terisolasi yang dikenal sebagai Availability Zone.

> **Note**
>
> Untuk informasi tentang menemukan Availability Zone untuk suatu AWS Wilayah, lihat Menjelaskan Availability Zone Anda di dokumentasi Amazon EC2.

Amazon beroperasi state-of-the-art, pusat data yang sangat tersedia. Meskipun jarang, kegagalan yang memengaruhi ketersediaan instans DB yang berada di lokasi yang sama dapat terjadi. Jika Anda meng-hosting semua instans DB di satu lokasi yang terpengaruh oleh kegagalan tersebut, tidak satu pun instans DB Anda akan tersedia.



Penting untuk diingat bahwa setiap AWS Wilayah sepenuhnya independen. Aktivitas Amazon RDS apa pun yang Anda lakukan (misalnya, membuat instance database atau mencantumkan instance

database yang tersedia) hanya berjalan di Wilayah default Anda saat ini. AWS AWS Wilayah default dapat diubah di konsol, atau dengan mengatur variabel `AWS_DEFAULT_REGION`lingkungan. Atau dapat diganti dengan menggunakan `--region` parameter dengan (). AWS Command Line Interface AWS CLI Untuk informasi selengkapnya, lihat Mengonfigurasi AWS Command Line Interface, khususnya bagian tentang variabel lingkungan dan opsi baris perintah.

Amazon RDS mendukung AWS Wilayah khusus yang disebut AWS GovCloud (US). Hal ini dirancang untuk memungkinkan lembaga pemerintah AS dan pelanggan memindahkan beban kerja yang lebih sensitif ke cloud. Wilayah AWS GovCloud (US) memenuhi persyaratan peraturan dan kepatuhan khusus pemerintah AS. Untuk informasi lebih lanjut, lihat Apa itu AWS GovCloud (US)?

Untuk membuat atau bekerja dengan instans Amazon RDS DB di AWS Wilayah tertentu, gunakan titik akhir layanan regional yang sesuai.

> **Note**
>
> Aurora tidak mendukung Zona Lokal.

AWS Daerah

Setiap AWS Wilayah dirancang untuk diisolasi dari AWS Wilayah lain. Rancangan ini mencapai toleransi kesalahan dan stabilitas sebesar mungkin.

Saat Anda melihat sumber daya, Anda hanya melihat sumber daya yang terkait dengan AWS Wilayah yang Anda tentukan. Ini karena AWS Wilayah terisolasi satu sama lain, dan kami tidak secara otomatis mereplikasi sumber daya di seluruh AWS Wilayah.

Ketersediaan wilayah

Ketika menggunakan klaster DB Aurora dengan operasi antarmuka baris perintah atau API, pastikan bahwa Anda menentukan titik akhir regional.

Topik

- Ketersediaan Wilayah Aurora MySQL
- Ketersediaan Wilayah Aurora PostgreSQL

Ketersediaan Wilayah Aurora MySQL

Tabel berikut menunjukkan AWS Wilayah di mana Aurora MySQL saat ini tersedia dan titik akhir untuk setiap Wilayah.

Nama Wilayah	Wilayah	Titik Akhir	Protokol	
AS Timur (Ohio)	us–east-2	rds.us-east-2.amazonaws.com	HTTPS	
AS Timur (Virginia Utara)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
AS Barat (California Utara)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	
AS Barat (Oregon)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS	
Afrika (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	
Asia Pasifik (Hong Kong)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
Asia Pasifik (Hyderabad)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS	
Asia Pasifik (Jakarta)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS	

Nama Wilayah	Wilayah	Titik Akhir	Protokol	
Asia Pasifik (Melbourne)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS	
Asia Pasifik (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS	
Asia Pasifik (Osaka)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS	
Asia Pasifik (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS	
Asia Pasifik (Singapura)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS	
Asia Pasifik (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS	
Asia Pasifik (Tokyo)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS	
Kanada (Pusat)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS	
Kanada Barat (Calgary)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS	

Nama Wilayah	Wilayah	Titik Akhir	Protokol	
Eropa (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
Eropa (Irlandia)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	
Eropa (London)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS	
Eropa (Milan)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS	
Eropa (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
Eropa (Spanyol)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS	
Eropa (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	
Eropa (Zürich)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS	
Israel (Tel Aviv)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS	
Timur Tengah (Bahrain)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS	
Timur Tengah (UAE)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS	

Nama Wilayah	Wilayah	Titik Akhir	Protokol	
Amerika Selatan (Sao Paulo)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS	
AWS GovCloud (AS-Timur)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS	
AWS GovCloud (AS-Barat)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

Ketersediaan Wilayah Aurora PostgreSQL

Tabel berikut menunjukkan AWS Wilayah di mana Aurora PostgreSQL saat ini tersedia dan titik akhir untuk setiap Wilayah.

Nama Wilayah	Wilayah	Titik Akhir	Protokol	
AS Timur (Ohio)	us–east-2	rds.us-east-2.amazonaws.com	HTTPS	
AS Timur (Virginia Utara)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS	
AS Barat (California Utara)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS	

Nama Wilayah	Wilayah	Titik Akhir	Protokol	
AS Barat (Oregon)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS	
Afrika (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	
Asia Pasifik (Hong Kong)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
Asia Pasifik (Hyderabad)	ap-south-2	rds.ap-south-2.amazonaws.com	HTTPS	
Asia Pasifik (Jakarta)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com	HTTPS	
Asia Pasifik (Melbourne)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com	HTTPS	
Asia Pasifik (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS	
Asia Pasifik (Osaka)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS	

Nama Wilayah	Wilayah	Titik Akhir	Protokol	
Asia Pasifik (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS	
Asia Pasifik (Singapura)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS	
Asia Pasifik (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS	
Asia Pasifik (Tokyo)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS	
Kanada (Pusat)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS	
Kanada Barat (Calgary)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS	
Eropa (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS	
Eropa (Irlandia)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS	
Eropa (London)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS	
Eropa (Milan)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS	

Nama Wilayah	Wilayah	Titik Akhir	Protokol	
Eropa (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
Eropa (Spanyol)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS	
Eropa (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	
Eropa (Zürich)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS	
Israel (Tel Aviv)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS	
Timur Tengah (Bahrain)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS	
Timur Tengah (UAE)	me-central-1	rds.me-central-1.amazonaws.com	HTTPS	
Amerika Selatan (Sao Paulo)	sa-east-1	rds.sa-east-1.amazonaws.com	HTTPS	
AWS GovCloud (AS-Timur)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com	HTTPS	

Nama Wilayah	Wilayah	Titik Akhir	Protokol	
AWS GovCloud (AS-Barat)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

Zona Ketersediaan

Zona Ketersediaan adalah lokasi yang terisolasi di Wilayah AWS tertentu. Setiap Wilayah memiliki beberapa Zona Ketersediaan (AZ) yang dirancang untuk menyediakan ketersediaan tinggi bagi Wilayah tersebut. AZ diidentifikasi oleh kode AWS Wilayah diikuti oleh pengidentifikasi huruf (misalnya,`us-east-1a`). Jika membuat VPC dan subnet, bukan menggunakan VPC default, Anda akan menentukan setiap subnet dalam AZ tertentu. Saat Anda membuat klaster DB Aurora, Aurora membuat instans utama di salah satu subnet dalam grup subnet DB VPC. Dengan demikian, instans tersebut akan dikaitkan dengan AZ tertentu yang dipilih oleh Aurora.

Setiap klaster DB Aurora meng-hosting salinan penyimpanan dalam tiga AZ terpisah. Setiap instans DB di klaster harus berada di salah satu dari tiga AZ ini. Ketika Anda membuat instans DB di klaster, Aurora memilih AZ yang sesuai secara otomatis jika Anda tidak menentukan AZ.

Gunakan perintah describe-availability-zonesAmazon EC2 sebagai berikut untuk menjelaskan Availability Zone dalam Wilayah tertentu yang diaktifkan untuk akun Anda.

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

Misalnya, untuk mendeskripsikan Zona Ketersediaan dalam Wilayah AS Timur (Virginia Utara) (us-east-1) yang diaktifkan untuk akun Anda, jalankan perintah berikut:

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

Untuk mempelajari cara menentukan AZ saat membuat klaster atau menambahkan instans ke klaster, lihat Konfigurasikan jaringan untuk klaster DB.

Zona waktu lokal untuk klaster DB Amazon Aurora

Secara default, zona waktu untuk klaster DB Amazon Aurora adalah Waktu Universal Terkoordinasi (UTC). Anda dapat mengatur zona waktu untuk instans di klaster DB ke zona waktu lokal untuk aplikasi Anda.

Untuk mengatur zona waktu lokal bagi klaster DB, atur parameter zona waktu ke salah satu nilai yang mendukung. Anda mengatur parameter ini di grup parameter klaster untuk klaster DB.

- Untuk Aurora MySQL, nama parameter ini adalah `time_zone`. Untuk informasi tentang praktik terbaik untuk mengatur parameter `time_zone`, lihat Mengoptimalkan operasi stempel waktu.
- Untuk Aurora PostgreSQL, nama parameter ini adalah `timezone`.

Saat Anda mengatur parameter zona waktu untuk klaster DB, semua instans di klaster DB berubah menggunakan zona waktu lokal baru. Dalam beberapa kasus, klaster Aurora DB lainnya mungkin menggunakan grup parameter klaster yang sama. Jika demikian, semua instans dalam klaster DB tersebut juga berubah menggunakan zona waktu lokal baru. Untuk informasi tentang parameter tingkat klaster, lihat Parameter instans DB dan klaster DB Amazon Aurora.

Setelah Anda mengatur zona waktu lokal, semua koneksi baru ke basis data mencerminkan perubahan tersebut. Dalam beberapa kasus, Anda mungkin memiliki koneksi terbuka ke basis data saat mengubah zona waktu lokal. Jika demikian, pembaruan zona waktu lokal tidak akan terlihat hingga Anda menutup koneksi dan membuka koneksi baru.

Jika Anda mereplikasi di seluruh AWS Wilayah, cluster DB sumber replikasi dan replika menggunakan grup parameter yang berbeda. Grup parameter unik untuk suatu AWS Wilayah. Untuk menggunakan zona waktu lokal yang sama bagi setiap instans, pastikan untuk mengatur parameter zona waktu dalam grup parameter untuk sumber replikasi dan replika.

Saat Anda memulihkan klaster DB dari snapshot klaster DB, zona waktu lokal diatur ke UTC. Anda dapat memperbarui zona waktu ke zona waktu lokal Anda setelah pemulihan selesai. Dalam beberapa kasus, Anda dapat memulihkan klaster DB ke titik waktu tertentu. Jika demikian, zona waktu lokal untuk klaster DB yang dipulihkan adalah pengaturan zona waktu dari grup parameter klaster DB yang dipulihkan.

Tabel berikut mencantumkan beberapa nilai yang dapat Anda gunakan untuk mengatur zona waktu lokal. Untuk mencantumkan semua zona waktu yang tersedia, Anda dapat menggunakan kueri SQL berikut:

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

> **Note**
>
> Untuk beberapa zona waktu, nilai waktu untuk rentang tanggal tertentu mungkin salah dilaporkan seperti yang tercantum dalam tabel. Untuk zona waktu Australia, singkatan zona waktu yang ditampilkan adalah nilai yang sudah tidak berlaku sebagaimana tercantum dalam tabel.

Zona waktu	Catatan
`Africa/Harare`	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 28 Feb 1903 21.49.40 GMT menjadi 28 Feb 1903 21.55.48 GMT.
`Africa/Monrovia`	
`Africa/Nairobi`	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 31 Des 1939 21.30.00 GMT menjadi 31 Des 1959 21.15.15 GMT.
`Africa/Windhoek`	
`America/Bogota`	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 23 Nov 1914 04.56.16 GMT menjadi 23 Nov 1914 04.56.20 GMT.
`America/Caracas`	
`America/Chihuahua`	
`America/Cuiaba`	
`America/Denver`	
`America/Fortaleza`	Dalam beberapa kasus, untuk klaster DB di Wilayah Amerika Selatan (Sao Paulo), waktu tidak ditampilkan dengan benar untuk zona waktu

Zona waktu	Catatan
	Brasil yang baru diubah. Jika demikian, atur ulang parameter zona waktu klaster DB ke `America/Fortaleza` .
`America/Guatemala`	
`America/Halifax`	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 27 Okt 1918 05.00.00 GMT menjadi 31 Okt 1918 05.00.00 GMT.
`America/Manaus`	Jika klaster DB Anda berada di zona waktu Amerika Selatan (Cuiaba) dan waktu yang diharapkan tidak ditampilkan dengan benar untuk zona waktu Brasil yang baru diubah, atur ulang parameter zona waktu klaster DB ke `America/Manaus` .
`America/Matamoros`	
`America/Monterrey`	
`America/Montevideo`	
`America/Phoenix`	
`America/Tijuana`	
`Asia/Ashgabat`	
`Asia/Baghdad`	
`Asia/Baku`	
`Asia/Bangkok`	
`Asia/Beirut`	
`Asia/Calcutta`	

Zona waktu	Catatan
Asia/Kabul	
Asia/Karachi	
Asia/Kathmandu	
Asia/Muscat	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 31 Des 1919 20.05.36 GMT menjadi 31 Des 1919 20.05.40 GMT.
Asia/Riyadh	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 13 Mar 1947 20.53.08 GMT menjadi 31 Des 1949 20.53.08 GMT.
Asia/Seoul	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 30 Nov 1904 15.30.00 GMT menjadi 07 Sep 1945 15.00.00 GMT.
Asia/Shanghai	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 31 Des 1927 15.54.08 GMT menjadi 02 Jun 1940 16.00.00 GMT.
Asia/Singapore	
Asia/Taipei	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 30 Sep 1937 16.00.00 GMT menjadi 29 Sep 1979 15.00.00 GMT.
Asia/Tehran	
Asia/Tokyo	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 30 Sep 1937 15.00.00 GMT menjadi 31 Des 1937 15.00.00 GMT.
Asia/Ulaanbaatar	
Atlantic/Azores	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 24 Mei 1911 01.54.32 GMT menjadi 01 Jan 1912 01.54.32 GMT.
Australia/Adelaide	Singkatan untuk zona waktu ini ditampilkan sebagai CST, bukan ACDT/ACST.

Zona waktu	Catatan
Australia/Brisbane	Singkatan untuk zona waktu ini ditampilkan sebagai EST, bukan AEDT/AEST.
Australia/Darwin	Singkatan untuk zona waktu ini ditampilkan sebagai CST, bukan ACDT/ACST.
Australia/Hobart	Singkatan untuk zona waktu ini ditampilkan sebagai EST, bukan AEDT/AEST.
Australia/Perth	Singkatan untuk zona waktu ini dikembalikan sebagai WST, bukan AWST AWDT/.
Australia/Sydney	Singkatan untuk zona waktu ini ditampilkan sebagai EST, bukan AEDT/AEST.
Brazil/East	
Canada/Saskatchewan	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 27 Okt 1918 08.00.00 GMT menjadi 31 Okt 1918 08.00.00 GMT.
Europe/Amsterdam	
Europe/Athens	
Europe/Dublin	
Europe/Helsinki	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 30 Apr 1921 22.20.08 GMT menjadi 30 Apr 1921 22.20.11 GMT.
Europe/Paris	
Europe/Prague	
Europe/Sarajevo	
Pacific/Auckland	

Zona waktu	Catatan
Pacific/Guam	
Pacific/Honolulu	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 21 Mei 1933 11.30.00 GMT menjadi 30 Sep 1945 11.30.00 GMT.
Pacific/Samoa	Pengaturan zona waktu ini dapat menghasilkan nilai yang salah dari 01 Jan 1911 11.22.48 GMT menjadi 01 Jan 1950 11.30.00 GMT.
US/Alaska	
US/Central	
US/Eastern	
US/East-Indiana	
US/Pacific	
UTC	

Fitur yang didukung di Amazon Aurora oleh Wilayah AWS dan mesin DB Aurora

Mesin basis data Aurora yang kompatibel dengan MySQL dan PostgreSQL mendukung beberapa fitur dan opsi Amazon Aurora dan Amazon RDS. Dukungan bervariasi di seluruh versi spesifik dari setiap mesin basis data, dan di seluruh Wilayah AWS. Untuk mengidentifikasi dukungan dan ketersediaan versi mesin DB RDS dalam versi Wilayah AWS tertentu, Anda dapat menggunakan bagian berikut ini.

Beberapa dari fitur ini adalah kemampuan khusus Aurora. Misalnya, Aurora Serverless, basis data global Aurora, dan dukungan untuk integrasi layanan AWS machine learning tidak didukung oleh Amazon RDS. Fitur lain, seperti Proksi Amazon RDS, didukung oleh Amazon Aurora dan Amazon RDS.

Topik

- Konvensi tabel
- Deployment Blue/Green
- Konfigurasi penyimpanan klaster Aurora
- Aliran aktivitas basis data di Aurora
- Mengekspor data klaster ke Amazon S3
- Mengekspor data snapshot ke Amazon S3
- Basis data global Aurora
- Autentikasi basis data IAM di Aurora
- Autentikasi Kerberos dengan Aurora
- Machine Learning Aurora
- Wawasan Performa dengan Aurora
- Integrasi nol-ETL dengan Amazon Redshift
- Proksi Amazon RDS
- Integrasi Secrets Manager
- Aurora Serverless v2
- Aurora Serverless v1
- API Data RDS
- Zero-downtime patching (ZDP)

- Fitur asli mesin

Konvensi tabel

Tabel-tabel di bagian fitur tersebut menggunakan pola berikut untuk menentukan nomor versi dan tingkat ketersediaan:

- Versi x.y – Hanya mendukung versi tertentu saja.
- Versi x.y dan yang lebih tinggi – Versi yang ditentukan dan semua versi minor yang lebih tinggi dari versi utamanya didukung. Misalnya, “versi 10.11 dan yang lebih tinggi” berarti versi 10.11, 10.11.1, dan 10.12 tersedia.
- - – Fitur saat ini tidak tersedia untuk fitur Aurora tertentu untuk mesin basis data Aurora, atau di Wilayah AWS spesifik tersebut.

Deployment Blue/Green

Deployment blue/green menyalin lingkungan basis data produksi di lingkungan pementasan yang terpisah dan tersinkron. Dengan menggunakan Deployment Blue/Green Amazon RDS, Anda dapat membuat perubahan pada basis data di lingkungan pementasan tanpa memengaruhi lingkungan produksi. Misalnya, Anda dapat meningkatkan versi mesin DB besar atau kecil, mengubah parameter basis data, atau membuat perubahan skema di lingkungan pementasannya. Setelah siap, Anda dapat mempromosikan lingkungan pementasan menjadi lingkungan basis data produksi baru. Untuk informasi selengkapnya, lihat Menggunakan Deployment Blue/Green Amazon RDS untuk pembaruan basis data.

Deployment Blue/Green dengan Aurora MySQL

Fitur Penerapan Biru/Hijau tersedia untuk semua versi Aurora MySQL secara keseluruhan. Wilayah AWS

Deployment Blue/Green dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk Deployment Blue/Green dengan Aurora PostgreSQL.

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Semua Wilayah AWS	Versi 16.1 dan lebih tinggi	Versi 15.4 dan yang lebih tinggi	Versi 14.9 dan yang lebih tinggi	Versi 13.12 dan yang lebih tinggi	Versi 12.16 dan yang lebih tinggi	Versi 11.21 dan yang lebih tinggi

Konfigurasi penyimpanan klaster Aurora

Amazon Aurora memiliki dua konfigurasi penyimpanan klaster DB, Aurora I/O-Optimized dan Aurora Standard. Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.

Aurora I/O-Optimized

Aurora I/O-Optimized tersedia di semua Wilayah AWS untuk versi Amazon Aurora berikut:

- Aurora MySQL versi 3.03.1 dan yang lebih tinggi
- Aurora PostgreSQL versi 15.2 dan yang lebih tinggi, 14.7 dan yang lebih tinggi, serta 13.10 dan yang lebih tinggi

Aurora Standard

Aurora Standard tersedia di semua Wilayah AWS untuk semua versi Aurora MySQL dan Aurora PostgreSQL.

Aliran aktivitas basis data di Aurora

Dengan menggunakan aliran aktivitas basis data di Aurora, Anda dapat memantau dan mengatur alarm untuk aktivitas audit di basis data Aurora. Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.

Aliran aktivitas basis data tidak didukung untuk fitur berikut:

- Aurora Serverless v1
- Aurora Serverless v2

- Babelfish for Aurora PostgreSQL

Topik

- Aliran aktivitas basis data dengan Aurora MySQL
- Aliran aktivitas basis data dengan Aurora PostgreSQL

Aliran aktivitas basis data dengan Aurora MySQL

Wilayah dan versi mesin berikut tersedia untuk aliran aktivitas basis data dengan Aurora MySQL.

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AS Timur (Ohio)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
AS Timur (Virginia Utara)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
AS Barat (California Utara)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
AS Barat (Oregon)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Afrika (Cape Town)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Asia Pasifik (Hong Kong)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Asia Pasifik (Hyderabad)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Asia Pasifik (Jakarta)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Asia Pasifik (Mumbai)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Asia Pasifik (Osaka)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Asia Pasifik (Seoul)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Asia Pasifik (Singapura)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Asia Pasifik (Sydney)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Asia Pasifik (Tokyo)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Kanada (Pusat)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Kanada Barat (Calgary)	–	–
China (Beijing)	–	–
Tiongkok (Ningxia)	–	–
Eropa (Frankfurt)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Eropa (Irlandia)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Eropa (London)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Eropa (Milan)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Eropa (Paris)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Eropa (Spanyol)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Eropa (Stockholm)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Eropa (Zürich)	–	–
Israel (Tel Aviv)	–	–
Timur Tengah (Bahrain)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Timur Tengah (UEA)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi
Amerika Selatan (Sao Paulo)	Semua versi yang tersedia	Aurora versi 2.11 dan yang lebih tinggi

Aliran aktivitas basis data dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk aliran aktivitas basis data dengan Aurora PostgreSQL.

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AS Timur (Ohio)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
AS Timur (Virginia Utara)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
						yang lebih tinggi
AS Barat (California Utara)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
AS Barat (Oregon)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Afrika (Cape Town)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Hong Kong)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Hyderabad)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asia Pasifik (Jakarta)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Melbourne)	–	–	–	–	–	–
Asia Pasifik (Mumbai)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Osaka)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Seoul)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Singapura)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asia Pasifik (Sydney)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Tokyo)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Kanada (Pusat)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Kanada Barat (Calgary)	–	–	–	–	–	–
China (Beijing)	–	–	–	–	–	–
Tiongkok (Ningxia)	–	–	–	–	–	–
Eropa (Frankfurt)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Eropa (Irlandia)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (London)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Milan)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Paris)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Spanyol)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Eropa (Stockholm)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Zürich)	–	–	–	–	–	–
Israel (Tel Aviv)	–	–	–	–	–	–
Timur Tengah (Bahrain)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Timur Tengah (UEA)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Amerika Selatan (Sao Paulo)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Semua versi yang tersedia	Semua versi yang tersedia	Semua versi yang tersedia	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Mengekspor data klaster ke Amazon S3

Anda dapat mengekspor data klaster DB Aurora ke bucket Amazon S3. Setelah data diekspor, Anda dapat menganalisis data yang diekspor secara langsung melalui alat seperti Amazon Athena atau Amazon Redshift Spectrum. Untuk informasi selengkapnya, lihat Mengekspor data klaster DB ke Amazon S3.

Mengekspor data klaster ke S3 tersedia di Wilayah AWS berikut:

- Asia Pasifik (Hong Kong)
- Asia Pasifik (Mumbai)
- Asia Pasifik (Osaka)
- Asia Pasifik (Seoul)
- Asia Pasifik (Singapura)
- Asia Pasifik (Sydney)
- Asia Pasifik (Tokyo)
- (Canada (Central)
- Kanada Barat (Calgary)
- Tiongkok (Ningxia)
- Eropa (Frankfurt)
- Eropa (Irlandia)
- Eropa (London)
- Eropa (Paris)
- Eropa (Stockholm)
- Amerika Selatan (Sao Paulo)
- AS Timur (Virginia Utara)
- AS Timur (Ohio)
- AS Barat (California Utara)
- AS Barat (Oregon)

Topik

- Mengekspor data klaster ke S3 dengan Aurora MySQL

- Mengekspor data klaster ke S3 dengan Aurora PostgreSQL

Mengekspor data klaster ke S3 dengan Aurora MySQL

Semua versi mesin Aurora MySQL mendukung mengekspor data klaster DB ke Amazon S3. Untuk informasi selengkapnya tentang versi, lihat Catatan Rilis untuk Aurora MySQL.

Mengekspor data klaster ke S3 dengan Aurora PostgreSQL

Semua versi mesin Aurora PostgreSQL mendukung mengekspor data klaster DB ke Amazon S3. Untuk informasi selengkapnya tentang versi, lihat Catatan Rilis untuk Aurora PostgreSQL.

Mengekspor data snapshot ke Amazon S3

Anda dapat mengekspor data snapshot klaster DB Aurora ke bucket Amazon S3. Anda dapat mengekspor snapshot manual dan snapshot sistem otomatis. Setelah data diekspor, Anda dapat menganalisis data yang diekspor secara langsung melalui alat seperti Amazon Athena atau Amazon Redshift Spectrum. Untuk informasi selengkapnya, lihat Mengekspor data snapshot klaster DB ke Amazon S3.

Mengekspor snapshot ke S3 di semua Wilayah AWS kecuali berikut:

- Asia Pasifik (Hyderabad)
- Asia Pasifik (Jakarta)
- Asia Pasifik (Melbourne)
- Kanada Barat (Calgary)
- Eropa (Spanyol)
- Eropa (Zürich)
- Israel (Tel Aviv)
- Timur Tengah (UEA)
- AWS GovCloud (AS-Timur)
- AWS GovCloud (AS-Barat)

Topik

- Mengekspor data snapshot ke S3 dengan Aurora MySQL
- Mengekspor data klaster ke S3 dengan Aurora PostgreSQL

Mengekspor data snapshot ke S3 dengan Aurora MySQL

Semua versi mesin Aurora MySQL yang tersedia mendukung ekspor data snapshot klaster DB ke Amazon S3. Untuk informasi selengkapnya tentang versi, lihat Catatan Rilis untuk Aurora MySQL.

Mengekspor data klaster ke S3 dengan Aurora PostgreSQL

Semua versi mesin Aurora PostgreSQL mendukung mengekspor data snapshot klaster DB ke Amazon S3. Untuk informasi selengkapnya tentang versi, lihat Catatan Rilis untuk Aurora PostgreSQL.

Basis data global Aurora

Basis data global Aurora adalah database tunggal yang mencakup beberapa Wilayah AWS, memungkinkan pembacaan global latensi rendah dan pemulihan bencana dari pemadaman di seluruh Wilayah. Ini memberikan toleransi kesalahan bawaan untuk penerapan Anda karena instans DB tidak bergantung pada satu Wilayah AWS, tetapi pada beberapa Wilayah dan Zona Ketersediaan yang berbeda. Untuk informasi selengkapnya, lihat Menggunakan basis data global Amazon Aurora.

Topik

- Basis data global Aurora dengan Aurora MySQL
- Basis data global Aurora dengan Aurora PostgreSQL

Basis data global Aurora dengan Aurora MySQL

Wilayah dan versi mesin berikut tersedia untuk basis data global dengan Aurora MySQL.

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AS Timur (Ohio)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
AS Timur (Virginia Utara)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
AS Barat (California Utara)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AS Barat (Oregon)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
Afrika (Cape Town)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.1 dan yang lebih tinggi
Asia Pasifik (Hong Kong)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.1 dan yang lebih tinggi
Asia Pasifik (Hyderabad)	Versi 3.02.0 dan yang lebih tinggi	Versi 2.11.2 dan yang lebih tinggi
Asia Pasifik (Jakarta)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.6 dan yang lebih tinggi
Asia Pasifik (Melbourne)	Versi 3.03.0 dan yang lebih tinggi	–
Asia Pasifik (Mumbai)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
Asia Pasifik (Osaka)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.3 dan yang lebih tinggi
Asia Pasifik (Seoul)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
Asia Pasifik (Singapura)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
Asia Pasifik (Sydney)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
Asia Pasifik (Tokyo)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
Kanada (Pusat)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Kanada Barat (Calgary)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
Tiongkok (Beijing)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.2 dan yang lebih tinggi
Tiongkok (Ningxia)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.2 dan yang lebih tinggi
Eropa (Frankfurt)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
Eropa (Irlandia)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
Eropa (London)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
Eropa (Milan)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.1 dan yang lebih tinggi
Eropa (Paris)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
Eropa (Spanyol)	Versi 3.02.0 dan yang lebih tinggi	–
Eropa (Stockholm)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
Eropa (Zürich)	Versi 3.02.0 dan yang lebih tinggi	–
Israel (Tel Aviv)	–	–
Timur Tengah (Bahrain)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.1 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Timur Tengah (UEA)	Versi 3.02.0 dan yang lebih tinggi	–
Amerika Selatan (Sao Paulo)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.1 dan yang lebih tinggi
AWS GovCloud (AS-Timur)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi
AWS GovCloud (AS-Barat)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.0 dan yang lebih tinggi

Basis data global Aurora dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk basis data global dengan Aurora PostgreSQL.

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AS Timur (Ohio)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
AS Timur (Virginia Utara)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
AS Barat (California Utara)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
						yang lebih tinggi
AS Barat (Oregon)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Afrika (Cape Town)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Hong Kong)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Hyderabad)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Jakarta)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asia Pasifik (Melbourne)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Mumbai)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Osaka)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Seoul)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Singapura)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asia Pasifik (Sydney)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Tokyo)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Kanada (Pusat)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Kanada Barat (Calgary)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Tiongkok (Beijing)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Tiongkok (Ningxia)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Frankfurt)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Irlandia)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (London)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Milan)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Eropa (Paris)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Spanyol)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Stockholm)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Zürich)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Israel (Tel Aviv)	–	–	–	–	–	–
Timur Tengah (Bahrain)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Timur Tengah (UEA)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Amerika Selatan (Sao Paulo)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
AWS GovCloud (AS-Timur)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
AWS GovCloud (AS-Barat)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Autentikasi basis data IAM di Aurora

Dengan autentikasi database IAM di Aurora, Anda dapat mengautentikasi ke cluster DB Anda menggunakan otentikasi database AWS Identity and Access Management (IAM) and Access Management (IAM). Dengan metode autentikasi ini, Anda tidak perlu menggunakan kata sandi saat terhubung ke klaster DB. Sebagai gantinya, Anda menggunakan token autentikasi. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.

Topik

- Autentikasi basis data IAM dengan Aurora MySQL
- Autentikasi basis data IAM dengan Aurora PostgreSQL

Autentikasi basis data IAM dengan Aurora MySQL

Autentikasi basis data IAM dengan Aurora MySQL tersedia di semua Wilayah untuk versi berikut:

- Aurora MySQL 3 – Semua versi yang tersedia
- Aurora MySQL 2 – Semua versi yang tersedia

Autentikasi basis data IAM dengan Aurora PostgreSQL

Autentikasi basis data IAM dengan Aurora PostgreSQL tersedia di semua Wilayah untuk versi berikut:

- Aurora PostgreSQL 16 - Semua versi yang tersedia
- Aurora PostgreSQL 15 – Semua versi yang tersedia
- Aurora PostgreSQL 14 – Semua versi yang tersedia
- Aurora PostgreSQL 13 – Semua versi yang tersedia
- Aurora PostgreSQL 12 – Semua versi yang tersedia
- Aurora PostgreSQL 11 – Semua versi yang tersedia

Autentikasi Kerberos dengan Aurora

Dengan menggunakan autentikasi Kerberos di Aurora, Anda dapat mendukung autentikasi eksternal pengguna basis data menggunakan Kerberos dan Microsoft Active Directory. Penggunaan Kerberos dan Active Directory memberikan manfaat masuk tunggal dan autentikasi terpusat pengguna basis data. Kerberos dan Active Directory tersedia dengan AWS Directory Service for Microsoft Active Directory, fitur dari AWS Directory Service. Untuk informasi selengkapnya, lihat Autentikasi Kerberos.

Topik

- Autentikasi Kerberos dengan Aurora MySQL
- Autentikasi Kerberos dengan Aurora PostgreSQL

Autentikasi Kerberos dengan Aurora MySQL

Wilayah dan versi mesin berikut tersedia untuk Autentikasi Kerberos dengan Aurora MySQL.

Wilayah	Aurora MySQL versi 3
AS Timur (Ohio)	Versi 3.03.0 dan yang lebih tinggi
AS Timur (Virginia Utara)	Versi 3.03.0 dan yang lebih tinggi
AS Barat (California Utara)	Versi 3.03.0 dan yang lebih tinggi
AS Barat (Oregon)	Versi 3.03.0 dan yang lebih tinggi
Afrika (Cape Town)	–
Asia Pasifik (Hong Kong)	–
Asia Pasifik (Jakarta)	–
Asia Pasifik (Mumbai)	Versi 3.03.0 dan yang lebih tinggi
Asia Pasifik (Osaka)	–
Asia Pasifik (Seoul)	Versi 3.03.0 dan yang lebih tinggi
Asia Pasifik (Singapura)	Versi 3.03.0 dan yang lebih tinggi
Asia Pasifik (Sydney)	Versi 3.03.0 dan yang lebih tinggi
Asia Pasifik (Tokyo)	Versi 3.03.0 dan yang lebih tinggi
Kanada (Pusat)	Versi 3.03.0 dan yang lebih tinggi
Kanada Barat (Calgary)	–
Tiongkok (Beijing)	Versi 3.03.0 dan yang lebih tinggi
Tiongkok (Ningxia)	Versi 3.03.0 dan yang lebih tinggi
Eropa (Frankfurt)	Versi 3.03.0 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3
Eropa (Irlandia)	Versi 3.03.0 dan yang lebih tinggi
Eropa (London)	Versi 3.03.0 dan yang lebih tinggi
Eropa (Milan)	–
Eropa (Paris)	Versi 3.03.0 dan yang lebih tinggi
Eropa (Spanyol)	–
Eropa (Stockholm)	Versi 3.03.0 dan yang lebih tinggi
Eropa (Zürich)	–
Israel (Tel Aviv)	–
Timur Tengah (Bahrain)	–
Timur Tengah (UEA)	–
Amerika Selatan (Sao Paulo)	Versi 3.03.0 dan yang lebih tinggi
AWS GovCloud (AS-Timur)	–
AWS GovCloud (AS-Barat)	–

Autentikasi Kerberos dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk Autentikasi Kerberos dengan Aurora PostgreSQL.

Wilayah	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AS Timur (Ohio)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi

Wilayah	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AS Timur (Virginia Utara)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
AS Barat (California Utara)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
AS Barat (Oregon)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Afrika (Cape Town)	–	–	–	–	–
Asia Pasifik (Hong Kong)	–	–	–	–	–
Asia Pasifik (Hyderabad)	–	–	–	–	–
Asia Pasifik (Jakarta)	–	–	–	–	–
Asia Pasifik (Melbourne)	–	–	–	–	–
Asia Pasifik (Mumbai)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Asia Pasifik (Osaka)	–	–	–	–	–
Asia Pasifik (Seoul)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi

Wilayah	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asia Pasifik (Singapura)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Asia Pasifik (Sydney)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Asia Pasifik (Tokyo)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Kanada (Pusat)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Kanada Barat (Calgary)	–	–	–	–	–
Tiongkok (Beijing)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Tiongkok (Ningxia)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Eropa (Frankfurt)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Eropa (Irlandia)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Eropa (London)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Europe (Milan)	–	–	–	–	–
Eropa (Paris)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi

Wilayah	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Eropa (Spanyol)	–	–	–	–	–
Eropa (Stockholm)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Eropa (Zürich)	–	–	–	–	–
Israel (Tel Aviv)	–	–	–	–	–
Timur Tengah (Bahrain)	–	–	–	–	–
Timur Tengah (UEA)	–	–	–	–	–
Amerika Selatan (Sao Paulo)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
AWS GovCloud (AS-Timur)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
AWS GovCloud (AS-Barat)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi

Machine Learning Aurora

Dengan menggunakan pembelajaran mesin Amazon Aurora, Anda dapat mengintegrasikan cluster Aurora DB Anda dengan Amazon Comprehend atau Amazon, tergantung pada kebutuhan Anda. SageMaker Amazon SageMaker Comprehend dan masing-masing mendukung kasus penggunaan pembelajaran mesin yang berbeda. Amazon Comprehend adalah layanan pemrosesan bahasa alami (NLP) yang digunakan untuk mengekstrak wawasan dari dokumen. Dengan menggunakan pembelajaran mesin Aurora dengan Amazon Comprehend, Anda dapat menentukan sentimen teks dalam tabel database Anda. SageMaker adalah layanan pembelajaran mesin berfitur lengkap. Ilmuwan data menggunakan Amazon SageMaker untuk membangun, melatih, dan menguji model pembelajaran mesin untuk berbagai tugas inferensi, seperti deteksi penipuan. Dengan menggunakan pembelajaran mesin Aurora dengan SageMaker, pengembang database dapat memanggil SageMaker fungsionalitas dalam kode SQL.

Tidak semua Wilayah AWS mendukung Amazon SageMaker Comprehend dan, dan hanya mendukung pembelajaran mesin Aurora Wilayah AWS tertentu dan dengan demikian menyediakan akses ke layanan ini dari cluster Aurora DB. Proses integrasi untuk machine learning Aurora juga berbeda menurut mesin basis data. Untuk informasi selengkapnya, lihat Menggunakan machine learning Amazon Aurora.

Topik

- Machine learning Aurora dengan Aurora MySQL
- Machine learning Aurora dengan Aurora PostgreSQL

Machine learning Aurora dengan Aurora MySQL

Pembelajaran mesin Aurora didukung untuk Aurora MySQL dalam tabel yang tercantum dalam tabel. Wilayah AWS Selain memiliki versi Aurora MySQL Anda yang tersedia, juga Wilayah AWS harus mendukung layanan yang ingin Anda gunakan. Untuk daftar Wilayah AWS tempat Amazon SageMaker tersedia, lihat SageMaker titik akhir dan kuota Amazon di. Referensi Umum Amazon Web Services Untuk daftar Wilayah AWS tempat Amazon Comprehend tersedia, lihat Amazon Comprehend endpoint dan kuota di. Referensi Umum Amazon Web Services

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AS Timur (Ohio)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AS Timur (Virginia Utara)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
AS Barat (California Utara)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
AS Barat (Oregon)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Afrika (Cape Town)	–	–
Asia Pasifik (Hong Kong)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Asia Pasifik (Hyderabad)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Asia Pasifik (Jakarta)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Asia Pasifik (Melbourne)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Asia Pasifik (Mumbai)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Asia Pasifik (Osaka)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07.3 dan yang lebih tinggi
Asia Pasifik (Seoul)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Asia Pasifik (Singapura)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Asia Pasifik (Sydney)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Asia Pasifik (Tokyo)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Kanada (Pusat)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Kanada Barat (Calgary)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Tiongkok (Beijing)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Tiongkok (Ningxia)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Eropa (Frankfurt)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Eropa (Irlandia)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Eropa (London)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Eropa (Milan)	–	–
Eropa (Paris)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Eropa (Spanyol)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Eropa (Stockholm)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Eropa (Zürich)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Israel (Tel Aviv)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Timur Tengah (Bahrain)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Timur Tengah (UEA)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
Amerika Selatan (Sao Paulo)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
AWS GovCloud (AS-Timur)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi
AWS GovCloud (AS-Barat)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan yang lebih tinggi

Machine learning Aurora dengan Aurora PostgreSQL

Pembelajaran mesin Aurora didukung untuk Aurora PostgreSQL dalam daftar dalam tabel. Wilayah AWS Selain memiliki versi Aurora PostgreSQL Anda tersedia, Wilayah AWS harus juga mendukung layanan yang ingin Anda gunakan. Untuk daftar Wilayah AWS tempat Amazon SageMaker tersedia, lihat SageMaker titik akhir dan kuota Amazon di. Referensi Umum Amazon Web Services Untuk daftar Wilayah AWS tempat Amazon Comprehend tersedia, lihat Amazon Comprehend endpoint dan kuota di. Referensi Umum Amazon Web Services

Wilayah dan versi mesin berikut tersedia untuk machine learning Aurora dengan Aurora PostgreSQL.

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AS Timur (Ohio)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
						yang lebih tinggi
AS Timur (Virginia Utara)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
AS Barat (California Utara)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
AS Barat (Oregon)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Afrika (Cape Town)	–	–	–	–	–	–
Asia Pasifik (Hong Kong)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Asia Pasifik (Hyderabad)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asia Pasifik (Jakarta)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Asia Pasifik (Melbourne)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Asia Pasifik (Mumbai)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Asia Pasifik (Osaka)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Asia Pasifik (Seoul)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Asia Pasifik (Singapura)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Asia Pasifik (Sydney)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asia Pasifik (Tokyo)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Kanada (Pusat)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Kanada Barat (Calgary)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Tiongkok (Beijing)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Tiongkok (Ningxia)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Eropa (Frankfurt)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Eropa (Irlandia)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Eropa (London)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Eropa (Milan)	–	–	–	–	–	–
Eropa (Paris)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Eropa (Spanyol)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Eropa (Stockholm)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Eropa (Zürich)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Israel (Tel Aviv)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Timur Tengah (Bahrain)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Timur Tengah (UEA)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
Amerika Selatan (Sao Paulo)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
AWS GovCloud (AS-Timur)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi
AWS GovCloud (AS-Barat)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	versi 14.3	Versi 13.3 dan yang lebih tinggi	Versi 12.4 dan yang lebih tinggi	Versi 11.9, 11.12, dan yang lebih tinggi

Wawasan Performa dengan Aurora

Wawasan Performa memperlengkap fitur pemantauan Amazon RDS yang sudah ada untuk mengilustrasikan dan membantu Anda menganalisis performa basis data. Dengan dasbor Wawasan Performa, Anda dapat memvisualisasikan muatan basis data pada muatan instans DB Amazon RDS dan memfilter muatan menurut peristiwa tunggu, pernyataan SQL, host, atau pengguna. Untuk informasi selengkapnya, lihat Ikhtisar Wawasan Performa tentang Amazon Aurora.

Untuk informasi dukungan Wilayah, mesin DB, dan kelas instans untuk fitur Wawasan Performa, lihat Dukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk fitur Wawasan Performa.

Topik

- Wawasan Performa dengan Aurora MySQL
- Wawasan Performa dengan Aurora PostgreSQL
- Wawasan Performa dengan Aurora Nirserver

Wawasan Performa dengan Aurora MySQL

> **Note**
>
> Dukungan versi engine berbeda untuk Wawasan Performa dengan Aurora MySQL jika Anda mengaktifkan kueri paralel. Untuk informasi selengkapnya tentang kueri paralel, lihat Bekerja dengan kueri paralel untuk Amazon Aurora MySQL.

Topik

- Wawasan Performa dengan Aurora MySQL dan kueri paralel dimatikan
- Wawasan Performa dengan Aurora MySQL dan kueri paralel dinyalakan

Wawasan Performa dengan Aurora MySQL dan kueri paralel dimatikan

Wilayah dan versi mesin berikut tersedia untuk Wawasan Performa dengan Aurora MySQL dan kueri paralel dimatikan.

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AS Timur (Ohio)	Semua versi	Semua versi
AS Timur (Virginia Utara)	Semua versi	Semua versi
AS Barat (California Utara)	Semua versi	Semua versi
AS Barat (Oregon)	Semua versi	Semua versi
Afrika (Cape Town)	Semua versi	Semua versi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Asia Pasifik (Hong Kong)	Semua versi	Semua versi
Asia Pasifik (Hyderabad)	Semua versi	Semua versi
Asia Pasifik (Jakarta)	Semua versi	Semua versi
Asia Pasifik (Melbourne)	Semua versi	Semua versi
Asia Pasifik (Mumbai)	Semua versi	Semua versi
Asia Pasifik (Osaka)	Semua versi	Semua versi
Asia Pasifik (Seoul)	Semua versi	Semua versi
Asia Pasifik (Singapura)	Semua versi	Semua versi
Asia Pasifik (Sydney)	Semua versi	Semua versi
Asia Pasifik (Tokyo)	Semua versi	Semua versi
Kanada (Pusat)	Semua versi	Semua versi
Kanada Barat (Calgary)	Semua versi	Semua versi
Tiongkok (Beijing)	Semua versi	Semua versi
Tiongkok (Ningxia)	Semua versi	Semua versi
Eropa (Frankfurt)	Semua versi	Semua versi
Eropa (Irlandia)	Semua versi	Semua versi
Eropa (London)	Semua versi	Semua versi
Eropa (Milan)	Semua versi	Semua versi
Eropa (Paris)	Semua versi	Semua versi
Eropa (Spanyol)	Semua versi	Semua versi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Eropa (Stockholm)	Semua versi	Semua versi
Eropa (Zürich)	Semua versi	Semua versi
Israel (Tel Aviv)	Semua versi	Semua versi
Timur Tengah (Bahrain)	Semua versi	Semua versi
Timur Tengah (UEA)	Semua versi	Semua versi
Amerika Selatan (Sao Paulo)	Semua versi	Semua versi
AWS GovCloud (AS-Timur)	Semua versi	Semua versi
AWS GovCloud (AS-Barat)	Semua versi	Semua versi

Wawasan Performa dengan Aurora MySQL dan kueri paralel dinyalakan

Wilayah dan versi mesin berikut tersedia untuk Wawasan Performa dengan Aurora MySQL dan kueri paralel dinyalakan.

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AS Timur (Ohio)	–	Versi 2.09.0 dan yang lebih tinggi
AS Timur (Virginia Utara)	–	Versi 2.09.0 dan yang lebih tinggi
AS Barat (California Utara)	–	Versi 2.09.0 dan yang lebih tinggi
AS Barat (Oregon)	–	Versi 2.09.0 dan yang lebih tinggi
Afrika (Cape Town)	–	Versi 2.09.0 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Asia Pasifik (Hong Kong)	–	Versi 2.09.0 dan yang lebih tinggi
Asia Pasifik (Hyderabad)	–	Semua versi
Asia Pasifik (Jakarta)	–	Versi 2.09.0 dan yang lebih tinggi
Asia Pasifik (Melbourne)	–	Versi 2.09.0 dan yang lebih tinggi
Asia Pasifik (Mumbai)	–	Versi 2.09.0 dan yang lebih tinggi
Asia Pasifik (Osaka)	–	Versi 2.09.0 dan yang lebih tinggi
Asia Pasifik (Seoul)	–	Versi 2.09.0 dan yang lebih tinggi
Asia Pasifik (Singapura)	–	Versi 2.09.0 dan yang lebih tinggi
Asia Pasifik (Sydney)	–	Versi 2.09.0 dan yang lebih tinggi
Asia Pasifik (Tokyo)	–	Versi 2.09.0 dan yang lebih tinggi
Kanada (Pusat)	–	Versi 2.09.0 dan yang lebih tinggi
Kanada Barat (Calgary)	–	Versi 2.09.0 dan yang lebih tinggi
Tiongkok (Beijing)	–	Versi 2.09.0 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Tiongkok (Ningxia)	–	Versi 2.09.0 dan yang lebih tinggi
Eropa (Frankfurt)	–	Versi 2.09.0 dan yang lebih tinggi
Eropa (Irlandia)	–	Versi 2.09.0 dan yang lebih tinggi
Eropa (London)	–	Versi 2.09.0 dan yang lebih tinggi
Eropa (Milan)	–	Versi 2.09.0 dan yang lebih tinggi
Eropa (Paris)	–	Versi 2.09.0 dan yang lebih tinggi
Eropa (Spanyol)	–	Versi 2.09.0 dan yang lebih tinggi
Eropa (Stockholm)	–	Versi 2.09.0 dan yang lebih tinggi
Eropa (Zürich)	–	Versi 2.09.0 dan yang lebih tinggi
Israel (Tel Aviv)	–	Versi 2.09.0 dan yang lebih tinggi
Timur Tengah (Bahrain)	–	Versi 2.09.0 dan yang lebih tinggi
Timur Tengah (UAE)	–	Versi 2.09.0 dan yang lebih tinggi
Amerika Selatan (Sao Paulo)	–	Versi 2.09.0 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AWS GovCloud (AS-Timur)	–	Versi 2.09.0 dan yang lebih tinggi
AWS GovCloud (AS-Barat)	–	Versi 2.09.0 dan yang lebih tinggi

Wawasan Performa dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk Wawasan Performa dengan Aurora PostgreSQL.

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
AS Timur (Ohio)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
AS Timur (Virginia Utara)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
AS Barat (California Utara)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
AS Barat (Oregon)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Afrika (Cape Town)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Asia Pasifik (Hong Kong)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
Asia Pasifik (Hyderabad)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Asia Pasifik (Jakarta)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Asia Pasifik (Melbourne)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Asia Pasifik (Mumbai)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Asia Pasifik (Osaka)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Asia Pasifik (Seoul)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Asia Pasifik (Singapura)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Asia Pasifik (Sydney)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
Asia Pasifik (Tokyo)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Kanada (Pusat)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Kanada Barat (Calgary)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Tiongkok (Beijing)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Tiongkok (Ningxia)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Eropa (Frankfurt)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Eropa (Irlandia)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Eropa (London)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Eropa (Milan)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Eropa (Paris)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Eropa (Spanyol)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11	Aurora PostgreSQL 10
Eropa (Stockholm)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Eropa (Zürich)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Israel (Tel Aviv)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Timur Tengah (Bahrain)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Timur Tengah (UEA)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
Amerika Selatan (Sao Paulo)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
AWS GovCloud (AS-Timur)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi
AWS GovCloud (AS-Barat)	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi	Semua versi

Wawasan Performa dengan Aurora Nirserver

Aurora Serverless v2 mendukung Wawasan Performa untuk semua versi yang kompatibel dengan MySQL dan PostgreSQL. Sebaiknya atur kapasitas minimum setidaknya ke 2 unit kapasitas Aurora (ACU).

Aurora Serverless v1 tidak mendukung Wawasan Performa.

Integrasi nol-ETL dengan Amazon Redshift

Integrasi nol-ETL Amazon Aurora dengan Amazon Redshift adalah solusi yang dikelola sepenuhnya untuk menyediakan data transaksional di Amazon Redshift setelah ditulis ke klaster Aurora. Untuk informasi selengkapnya, lihat Menggunakan integrasi nol-ETL .

Wilayah dan versi berikut tersedia untuk integrasi nol-ETL dengan Amazon Redshift.

Topik

- Integrasi nol-ETL Aurora MySQL
- Integrasi nol-ETL Aurora PostgreSQL

Integrasi nol-ETL Aurora MySQL

Wilayah	Aurora MySQL versi 3
AS Timur (Virginia Utara)	Versi 3.05.2 dan lebih tinggi
AS Timur (Ohio)	Versi 3.05.2 dan lebih tinggi
AS Barat (Oregon)	Versi 3.05.2 dan lebih tinggi
Asia Pasifik (Tokyo)	Versi 3.05.2 dan lebih tinggi
Asia Pasifik (Singapura)	Versi 3.05.2 dan lebih tinggi
Eropa (Irlandia)	Versi 3.05.2 dan lebih tinggi
Asia Pasifik (Sydney)	Versi 3.05.2 dan lebih tinggi
Eropa (Frankfurt)	Versi 3.05.2 dan lebih tinggi

Wilayah	Aurora MySQL versi 3
Eropa (Stockholm)	Versi 3.05.2 dan lebih tinggi

Integrasi nol-ETL Aurora PostgreSQL

Untuk rilis pratinjau integrasi nol-ETL Aurora PostgreSQL dengan Amazon Redshift, Anda harus membuat integrasi di Lingkungan Pratinjau Basis Data Amazon RDS, di AS Timur (Ohio) (us-east-2) Wilayah AWS. Lingkungan pratinjau memungkinkan Anda menguji beta, kandidat rilis, dan versi produksi awal perangkat lunak mesin basis data PostgreSQL.

Klaster DB sumber Anda harus menjalankan Aurora PostgreSQL (kompatibel dengan PostgreSQL 15.4 dan Dukungan Nol-ETL).

Proksi Amazon RDS

Proksi Amazon RDS adalah proksi basis data yang sepenuhnya terkelola dengan ketersediaan tinggi yang membuat aplikasi lebih dapat ditingkatkan dengan menggabungkan dan berbagi koneksi basis data yang telah dibuat. Untuk informasi selengkapnya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS untuk Aurora.

Topik

- Proksi Amazon RDS dengan Aurora MySQL
- Proksi Amazon RDS dengan Aurora PostgreSQL

Proksi Amazon RDS dengan Aurora MySQL

Wilayah dan versi mesin berikut tersedia untuk Proksi RDS dengan Aurora MySQL.

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AS Timur (Ohio)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
AS Timur (Virginia Utara)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AS Barat (California Utara)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
AS Barat (Oregon)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Afrika (Cape Town)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Asia Pasifik (Hong Kong)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Asia Pasifik (Hyderabad)	–	–
Asia Pasifik (Jakarta)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Asia Pasifik (Melbourne)	–	–
Asia Pasifik (Mumbai)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Asia Pasifik (Osaka)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Asia Pasifik (Seoul)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Asia Pasifik (Singapura)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Asia Pasifik (Sydney)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Asia Pasifik (Tokyo)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Kanada (Pusat)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Kanada Barat (Calgary)	–	–
Tiongkok (Beijing)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Tiongkok (Ningxia)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Eropa (Frankfurt)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Eropa (Irlandia)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Eropa (London)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Eropa (Milan)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Eropa (Paris)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Eropa (Spanyol)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Eropa (Stockholm)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
Eropa (Zürich)	–	–
Israel (Tel Aviv)	–	–
Timur Tengah (Bahrain)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Timur Tengah (UEA)	–	–
Amerika Selatan (Sao Paulo)	Versi 3.01.0 dan yang lebih tinggi	Versi 2.07 dan versi 2.11 dan yang lebih tinggi
AWS GovCloud (AS-Timur)	–	–
AWS GovCloud (AS-Barat)	–	–

Proksi Amazon RDS dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk Proksi RDS dengan Aurora PostgreSQL.

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AS Timur (Ohio)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
AS Timur (Virginia Utara)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
AS Barat (California Utara)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AS Barat (Oregon)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Afrika (Cape Town)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Hong Kong)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Hyderabad)	–	–	–	–	–	–
Asia Pasifik (Jakarta)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Melbourne)	–	–	–	–	–	–

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asia Pasifik (Mumbai)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Osaka)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Seoul)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Singapura)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Asia Pasifik (Sydney)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Asia Pasifik (Tokyo)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Kanada (Pusat)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Kanada Barat (Calgary)	–	–	–	–	–	–
Tiongkok (Beijing)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Tiongkok (Ningxia)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Frankfurt)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Eropa (Irlandia)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (London)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Milan)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Paris)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Spanyol)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
Eropa (Stockholm)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Eropa (Zürich)	–	–	–	–	–	–
Israel (Tel Aviv)	–	–	–	–	–	–
Timur Tengah (Bahrain)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
Timur Tengah (UEA)	–	–	–	–	–	–
Amerika Selatan (Sao Paulo)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.4 dan yang lebih tinggi	Versi 12.8 dan yang lebih tinggi	Versi 11.9 dan versi 11.13 dan yang lebih tinggi
AWS GovCloud (AS-Timur)	–	–	–	–	–	–

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13	Aurora PostgreSQL 12	Aurora PostgreSQL 11
AWS GovCloud (AS-Barat)	–	–	–	–	–	–

Integrasi Secrets Manager

Dengan AWS Secrets Manager, Anda dapat mengganti kredensi hard-code dalam kode Anda, termasuk kata sandi database, dengan panggilan API ke Secrets Manager untuk mengambil rahasia secara terprogram. Untuk informasi selengkapnya tentang Secrets Manager, lihat Panduan Pengguna AWS Secrets Manager.

Anda dapat menentukan Amazon Aurora mengelola kata sandi pengguna master di Secrets Manager untuk klaster DB Aurora. Aurora menghasilkan kata sandi, menyimpannya di Secrets Manager, dan merotasinya secara teratur. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager.

Integrasi Secrets Manager tersedia untuk semua mesin DB Aurora dan semua versi.

Integrasi Secrets Manager tersedia di semua Wilayah AWS kecuali yang berikut:

- Kanada Barat (Calgary)
- AWS GovCloud (AS-Timur)
- AWS GovCloud (AS-Barat)

Aurora Serverless v2

Aurora Serverless v2 adalah fitur penskalaan otomatis sesuai permintaan yang dirancang sebagai pendekatan hemat biaya untuk menjalankan beban kerja yang terputus-putus atau tidak dapat diprediksi di Amazon Aurora. Fitur ini secara otomatis meningkatkan/menurunkan kapasitas sesuai kebutuhan aplikasi Anda. Penskalaan lebih cepat dan lebih granular daripada dengan Aurora Serverless v1. Dengan Aurora Serverless v2, setiap klaster dapat berisi instans DB penulis dan beberapa instans DB pembaca. Anda dapat menggabungkan Aurora Serverless v2 dan instans

DB tradisional yang disediakan dalam klaster yang sama. Untuk informasi selengkapnya, lihat Menggunakan Aurora Serverless v2.

Topik

- Aurora Serverless v2 dengan Aurora MySQL
- Aurora Serverless v2 dengan Aurora PostgreSQL

Aurora Serverless v2 dengan Aurora MySQL

Wilayah dan versi mesin berikut tersedia untuk Aurora Serverless v2 dengan Aurora MySQL.

Wilayah	Aurora MySQL versi 3
AS Timur (Ohio)	Versi 3.02.0 dan yang lebih tinggi
AS Timur (Virginia Utara)	Versi 3.02.0 dan yang lebih tinggi
AS Barat (California Utara)	Versi 3.02.0 dan yang lebih tinggi
AS Barat (Oregon)	Versi 3.02.0 dan yang lebih tinggi
Afrika (Cape Town)	Versi 3.02.0 dan yang lebih tinggi
Asia Pasifik (Hong Kong)	Versi 3.02.0 dan yang lebih tinggi
Asia Pasifik (Hyderabad)	Versi 3.02.3 dan yang lebih tinggi
Asia Pasifik (Jakarta)	Versi 3.02.0 dan yang lebih tinggi
Asia Pasifik (Melbourne)	Versi 3.02.3 dan yang lebih tinggi
Asia Pasifik (Mumbai)	Versi 3.02.0 dan yang lebih tinggi
Asia Pasifik (Osaka)	Versi 3.02.0 dan yang lebih tinggi
Asia Pasifik (Seoul)	Versi 3.02.0 dan yang lebih tinggi
Asia Pasifik (Singapura)	Versi 3.02.0 dan yang lebih tinggi
Asia Pasifik (Sydney)	Versi 3.02.0 dan yang lebih tinggi

Wilayah	Aurora MySQL versi 3
Asia Pasifik (Tokyo)	Versi 3.02.0 dan yang lebih tinggi
Kanada (Pusat)	Versi 3.02.0 dan yang lebih tinggi
Kanada Barat (Calgary)	Versi 3.04.0, 3.04.1, 3.05.0, 3.05.1 dan lebih tinggi
Tiongkok (Beijing)	Versi 3.02.2 dan yang lebih tinggi
Tiongkok (Ningxia)	Versi 3.02.2 dan yang lebih tinggi
Eropa (Frankfurt)	Versi 3.02.0 dan yang lebih tinggi
Eropa (Irlandia)	Versi 3.02.0 dan yang lebih tinggi
Eropa (London)	Versi 3.02.0 dan yang lebih tinggi
Eropa (Milan)	Versi 3.02.0 dan yang lebih tinggi
Eropa (Paris)	Versi 3.02.0 dan yang lebih tinggi
Eropa (Spanyol)	Versi 3.02.3 dan yang lebih tinggi
Eropa (Stockholm)	Versi 3.02.0 dan yang lebih tinggi
Eropa (Zürich)	Versi 3.02.3 dan yang lebih tinggi
Israel (Tel Aviv)	Versi 3.02.3 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi
Timur Tengah (Bahrain)	Versi 3.02.0 dan yang lebih tinggi
Timur Tengah (UEA)	Versi 3.02.3 dan yang lebih tinggi
Amerika Selatan (Sao Paulo)	Versi 3.02.0 dan yang lebih tinggi
AWS GovCloud (AS-Timur)	Versi 3.02.2 dan yang lebih tinggi
AWS GovCloud (AS-Barat)	Versi 3.02.2 dan yang lebih tinggi

Aurora Serverless v2 dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk Aurora Serverless v2 dengan Aurora PostgreSQL.

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
AS Timur (Ohio)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
AS Timur (Virginia Utara)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
AS Barat (California Utara)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
AS Barat (Oregon)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Afrika (Cape Town)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Asia Pasifik (Hong Kong)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Asia Pasifik (Hyderabad)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.6 dan yang lebih tinggi	Versi 13.9 dan yang lebih tinggi
Asia Pasifik (Jakarta)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Asia Pasifik (Melbourne)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.6 dan yang lebih tinggi	Versi 13.9 dan yang lebih tinggi
Asia Pasifik (Mumbai)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Asia Pasifik (Osaka)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
Asia Pasifik (Seoul)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Asia Pasifik (Singapura)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Asia Pasifik (Sydney)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Asia Pasifik (Tokyo)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Kanada (Pusat)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Kanada Barat (Calgary)	Versi 16.1 dan lebih tinggi	Versi 15.3 dan lebih tinggi	Versi 14.6, 14.8 dan lebih tinggi	Versi 13.9, 13.11 dan lebih tinggi
Tiongkok (Beijing)	–	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Tiongkok (Ningxia)	–	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Eropa (Frankfurt)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Eropa (Irlandia)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Eropa (London)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Eropa (Milan)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi

Wilayah	Aurora PostgreSQL 16	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
Eropa (Paris)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Eropa (Spanyol)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.6 dan yang lebih tinggi	Versi 13.9 dan yang lebih tinggi
Eropa (Stockholm)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Eropa (Zürich)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.6 dan yang lebih tinggi	Versi 13.9 dan yang lebih tinggi
Israel (Tel Aviv)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.6 dan yang lebih tinggi	Versi 13.9 dan yang lebih tinggi
Timur Tengah (Bahrain)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
Timur Tengah (UEA)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.6 dan yang lebih tinggi	Versi 13.9 dan yang lebih tinggi
Amerika Selatan (Sao Paulo)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
AWS GovCloud (AS-Timur)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi
AWS GovCloud (AS-Barat)	Versi 16.1 dan lebih tinggi	Versi 15.2 dan yang lebih tinggi	Versi 14.3 dan yang lebih tinggi	Versi 13.6 dan yang lebih tinggi

Aurora Serverless v1

Aurora Serverless v1 adalah fitur penskalaan otomatis sesuai permintaan yang dirancang sebagai pendekatan hemat biaya untuk menjalankan beban kerja yang terputus-putus atau tidak dapat diprediksi di Amazon Aurora. Fitur ini secara otomatis memulai, mematikan, dan meningkatkan/

menurunkan kapasitas, sesuai kebutuhan aplikasi Anda, menggunakan satu instans DB di setiap klaster. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Serverless v1.

Topik

- Aurora Serverless v1 dengan Aurora MySQL
- Aurora Serverless v1 dengan Aurora PostgreSQL

Aurora Serverless v1 dengan Aurora MySQL

Wilayah dan versi mesin berikut tersedia untuk Aurora Serverless v1 dengan Aurora MySQL.

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AS Timur (Ohio)	–	Versi 2.11.4
AS Timur (Virginia Utara)	–	Versi 2.11.4
AS Barat (California Utara)	–	Versi 2.11.4
AS Barat (Oregon)	–	Versi 2.11.4
Afrika (Cape Town)	–	–
Asia Pasifik (Hong Kong)	–	–
Asia Pasifik (Hyderabad)	–	–
Asia Pasifik (Jakarta)	–	–
Asia Pasifik (Melbourne)	–	–
Asia Pasifik (Mumbai)	–	Versi 2.11.4
Asia Pasifik (Osaka)	–	–
Asia Pasifik (Seoul)	–	Versi 2.11.4
Asia Pasifik (Singapura)	–	Versi 2.11.4
Asia Pasifik (Sydney)	–	Versi 2.11.4

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Asia Pasifik (Tokyo)	–	Versi 2.11.4
Kanada (Pusat)	–	Versi 2.11.4
Kanada Barat (Calgary)	–	–
China (Beijing)	–	–
China (Ningxia)	–	Versi 2.11.4
Eropa (Frankfurt)	–	Versi 2.11.4
Eropa (Irlandia)	–	Versi 2.11.4
Eropa (London)	–	Versi 2.11.4
Eropa (Milan)	–	–
Eropa (Paris)	–	Versi 2.11.4
Eropa (Spanyol)	–	–
Eropa (Stockholm)	–	–
Eropa (Zürich)	–	–
Israel (Tel Aviv)	–	–
Timur Tengah (Bahrain)	–	–
Timur Tengah (UEA)	–	–
Amerika Selatan (Sao Paulo)	–	–
AWS GovCloud (AS-Timur)	–	–
AWS GovCloud (AS-Barat)	–	–

Aurora Serverless v1 dengan Aurora PostgreSQL

Wilayah dan versi mesin berikut tersedia untuk Aurora Serverless v1 dengan Aurora PostgreSQL.

Wilayah	Aurora PostgreSQL 13	Aurora PostgreSQL 11
AS Timur (Ohio)	Versi 13.12	Versi 11.21
AS Timur (Virginia Utara)	Versi 13.12	Versi 11.21
AS Barat (California Utara)	Versi 13.12	Versi 11.21
AS Barat (Oregon)	Versi 13.12	Versi 11.21
Afrika (Cape Town)	–	–
Asia Pasifik (Hong Kong)	–	–
Asia Pasifik (Hyderabad)	–	–
Asia Pasifik (Jakarta)	–	–
Asia Pasifik (Melbourne)	–	–
Asia Pasifik (Mumbai)	Versi 13.12	Versi 11.21
Asia Pasifik (Osaka)	–	–
Asia Pasifik (Seoul)	Versi 13.12	Versi 11.21
Asia Pasifik (Singapura)	Versi 13.12	Versi 11.21
Asia Pasifik (Sydney)	Versi 13.12	Versi 11.21
Asia Pasifik (Tokyo)	Versi 13.12	Versi 11.21
Kanada (Pusat)	Versi 13.12	Versi 11.21
Kanada Barat (Calgary)	–	–
China (Beijing)	–	–

Wilayah	Aurora PostgreSQL 13	Aurora PostgreSQL 11
Tiongkok (Ningxia)	–	–
Eropa (Frankfurt)	Versi 13.12	Versi 11.21
Eropa (Irlandia)	Versi 13.12	Versi 11.21
Eropa (London)	Versi 13.12	Versi 11.21
Eropa (Milan)	–	–
Eropa (Paris)	Versi 13.12	Versi 11.21
Eropa (Spanyol)	–	–
Eropa (Stockholm)	–	–
Eropa (Zürich)	–	–
Israel (Tel Aviv)	–	–
Timur Tengah (Bahrain)	–	–
Timur Tengah (UEA)	–	–
Amerika Selatan (Sao Paulo)	–	–
AWS GovCloud (AS-Timur)	–	–
AWS GovCloud (AS-Barat)	–	–

API Data RDS

RDS Data API (Data API) menyediakan antarmuka layanan web ke cluster Amazon Aurora DB. Alih-alih mengelola koneksi basis data dari aplikasi klien, Anda dapat menjalankan perintah SQL pada titik akhir HTTPS. Untuk informasi selengkapnya, lihat Menggunakan RDS Data API.

Untuk Aurora MySQL, Data API tidak didukung untuk atau untuk Aurora Serverless v2 kluster DB yang disediakan.

Topik

- API Data dengan Aurora MySQL Serverless v1
- Data API dengan Aurora PostgreSQL Serverless v2 dan disediakan
- API Data dengan Aurora PostgreSQL Serverless v1

API Data dengan Aurora MySQL Serverless v1

Wilayah dan versi engine berikut tersedia untuk Data API dengan Aurora MySQL Serverless v1.

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AS Timur (Ohio)	–	Versi 2.11.3
AS Timur (Virginia Utara)	–	Versi 2.11.3
AS Barat (California Utara)	–	Versi 2.11.3
AS Barat (Oregon)	–	Versi 2.11.3
Afrika (Cape Town)	–	–
Asia Pasifik (Hong Kong)	–	–
Asia Pasifik (Hyderabad)	–	–
Asia Pasifik (Jakarta)	–	–
Asia Pasifik (Melbourne)	–	–
Asia Pasifik (Mumbai)	–	Versi 2.11.3
Asia Pasifik (Osaka)	–	–
Asia Pasifik (Seoul)	–	Versi 2.11.3
Asia Pasifik (Singapura)	–	Versi 2.11.3
Asia Pasifik (Sydney)	–	Versi 2.11.3
Asia Pasifik (Tokyo)	–	Versi 2.11.3

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
Kanada (Pusat)	–	Versi 2.11.3
Kanada Barat (Calgary)	–	–
China (Beijing)	–	–
China (Ningxia)	–	Versi 2.11.3
Eropa (Frankfurt)	–	Versi 2.11.3
Eropa (Irlandia)	–	Versi 2.11.3
Eropa (London)	–	Versi 2.11.3
Eropa (Milan)	–	–
Eropa (Paris)	–	Versi 2.11.3
Eropa (Spanyol)	–	–
Eropa (Stockholm)	–	–
Eropa (Zürich)	–	–
Israel (Tel Aviv)	–	–
Timur Tengah (Bahrain)	–	–
Timur Tengah (UEA)	–	–
Amerika Selatan (Sao Paulo)	–	–
AWS GovCloud (AS-Timur)	–	–
AWS GovCloud (AS-Barat)	–	–

Data API dengan Aurora PostgreSQL Serverless v2 dan disediakan

Wilayah dan versi engine berikut tersedia untuk Data API dengan Aurora PostgreSQL Serverless v2 dan kluster DB yang disediakan.

Wilayah	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
AS Timur (Ohio)	–	–	–
AS Timur (Virginia Utara)	Versi 15.3 dan lebih tinggi	Versi 14.8 dan lebih tinggi	Versi 13.11 dan lebih tinggi
AS Barat (California Utara)	–	–	–
AS Barat (Oregon)	Versi 15.3 dan lebih tinggi	Versi 14.8 dan lebih tinggi	Versi 13.11 dan lebih tinggi
Afrika (Cape Town)	–	–	–
Asia Pasifik (Hong Kong)	–	–	–
Asia Pasifik (Hyderabad)	–	–	–
Asia Pasifik (Jakarta)	–	–	–
Asia Pasifik (Melbourne)	–	–	–
Asia Pasifik (Mumbai)	–	–	–
Asia Pasifik (Osaka)	–	–	–
Asia Pasifik (Seoul)	–	–	–
Asia Pasifik (Singapura)	–	–	–

Wilayah	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
Asia Pasifik (Sydney)	–	–	–
Asia Pasifik (Tokyo)	Versi 15.3 dan lebih tinggi	Versi 14.8 dan lebih tinggi	Versi 13.11 dan lebih tinggi
Kanada (Pusat)	–	–	–
Kanada Barat (Calgary)	–	–	–
China (Beijing)	–	–	–
Tiongkok (Ningxia)	–	–	–
Eropa (Frankfurt)	Versi 15.3 dan lebih tinggi	Versi 14.8 dan lebih tinggi	Versi 13.11 dan lebih tinggi
Eropa (Irlandia)	–	–	–
Eropa (London)	–	–	–
Eropa (Milan)	–	–	–
Eropa (Paris)	–	–	–
Eropa (Spanyol)	–	–	–
Eropa (Stockholm)	–	–	–
Eropa (Zürich)	–	–	–
Israel (Tel Aviv)	–	–	–
Timur Tengah (Bahrain)	–	–	–
Timur Tengah (UEA)	–	–	–

Wilayah	Aurora PostgreSQL 15	Aurora PostgreSQL 14	Aurora PostgreSQL 13
Amerika Selatan (Sao Paulo)	–	–	–
AWS GovCloud (AS-Timur)	–	–	–
AWS GovCloud (AS-Barat)	–	–	–

API Data dengan Aurora PostgreSQL Serverless v1

Wilayah dan versi engine berikut tersedia untuk Data API dengan Aurora PostgreSQL Serverless v1.

Wilayah	Aurora PostgreSQL 13	Aurora PostgreSQL 11
AS Timur (Ohio)	Versi 13.9	Versi 11.18
AS Timur (Virginia Utara)	Versi 13.9	Versi 11.18
AS Barat (California Utara)	Versi 13.9	Versi 11.18
AS Barat (Oregon)	Versi 13.9	Versi 11.18
Afrika (Cape Town)	–	–
Asia Pasifik (Hong Kong)	–	–
Asia Pasifik (Hyderabad)	–	–
Asia Pasifik (Jakarta)	–	–
Asia Pasifik (Melbourne)	–	–
Asia Pasifik (Mumbai)	Versi 13.9	Versi 11.18
Asia Pasifik (Osaka)	–	–

Wilayah	Aurora PostgreSQL 13	Aurora PostgreSQL 11
Asia Pasifik (Seoul)	Versi 13.9	Versi 11.18
Asia Pasifik (Singapura)	Versi 13.9	Versi 11.18
Asia Pasifik (Sydney)	Versi 13.9	Versi 11.18
Asia Pasifik (Tokyo)	Versi 13.9	Versi 11.18
Kanada (Pusat)	Versi 13.9	Versi 11.18
China (Beijing)	–	–
Tiongkok (Ningxia)	–	–
Eropa (Frankfurt)	Versi 13.9	Versi 11.18
Eropa (Irlandia)	Versi 13.9	Versi 11.18
Eropa (London)	Versi 13.9	Versi 11.18
Eropa (Milan)	–	–
Eropa (Paris)	Versi 13.9	Versi 11.18
Eropa (Spanyol)	–	–
Eropa (Stockholm)	–	–
Eropa (Zürich)	–	–
Israel (Tel Aviv)	–	–
Timur Tengah (Bahrain)	–	–
Timur Tengah (UEA)	–	–
Amerika Selatan (Sao Paulo)	–	–
AWS GovCloud (AS-Timur)	–	–

Wilayah	Aurora PostgreSQL 13	Aurora PostgreSQL 11
AWS GovCloud (AS-Barat)	–	–

Zero-downtime patching (ZDP)

Peningkatan klaster DB Aurora kemungkinan akan menimbulkan pemadaman layanan saat basis data dinonaktifkan dan sedang ditingkatkan. Secara default, jika Anda memulai peningkatan saat basis data sibuk, Anda kehilangan semua koneksi dan transaksi yang sedang diproses oleh klaster DB. Jika Anda menunggu hingga basis data idle untuk melakukan peningkatan, Anda mungkin harus menunggu lama.

Fitur zero-downtime patching (ZDP) mencoba, berdasarkan upaya terbaik, untuk menjaga koneksi klien selama peningkatan Aurora MySQL. Jika ZDP berhasil diselesaikan, sesi aplikasi dipertahankan dan mesin basis data dimulai ulang saat peningkatan sedang berlangsung. Pengaktifan ulang mesin basis data dapat menyebabkan penurunan throughput yang berlangsung selama beberapa detik hingga kira-kira satu menit.

Untuk informasi mendetail tentang kondisi dan versi mesin di mana ZDP tersedia untuk peningkatan Aurora MySQL, lihat Menggunakan zero-downtime patching.

Untuk informasi mendetail tentang kondisi dan versi mesin di mana ZDP tersedia untuk peningkatan Aurora PostgreSQL, lihat Peningkatan rilis minor dan patching nol-waktu henti.

Fitur asli mesin

Mesin basis data Aurora juga mendukung fitur dan fungsionalitas tambahan khusus untuk Aurora. Beberapa fitur asli mesin mungkin memiliki dukungan terbatas atau hak istimewa terbatas untuk mesin, versi, atau Wilayah DB Aurora tertentu.

Topik

- Fitur asli mesin untuk Aurora MySQL
- Fitur asli mesin untuk Aurora PostgreSQL

Fitur asli mesin untuk Aurora MySQL

Berikut adalah fitur asli mesin untuk Aurora MySQL

- Audit Lanjutan
- Backtrack
- Pertanyaan injeksi kesalahan
- Penerusan tulis dalam klaster
- Kueri paralel

Fitur asli mesin untuk Aurora PostgreSQL

Berikut adalah fitur asli mesin untuk Aurora PostgreSQL

- Babelfish
- Pertanyaan injeksi kesalahan
- Manajemen rencana kueri

Manajemen koneksi Amazon Aurora

Amazon Aurora biasanya menangani sebuah klaster yang terdiri dari beberapa instans DB, bukan instans tunggal. Setiap koneksi ditangani oleh instans DB tertentu. Saat Anda terhubung ke klaster Aurora, nama dan port host yang Anda tentukan akan mengarah ke sebuah handler perantara yang disebut titik akhir. Aurora menggunakan mekanisme titik akhir untuk menyederhanakan koneksi tersebut. Dengan demikian, Anda tidak perlu melakukan hard-coding terhadap semua nama host atau menulis logika Anda sendiri untuk penyeimbang beban dan perutean ulang koneksi jika beberapa instans DB tidak tersedia.

Untuk tugas Aurora tertentu, beberapa instans atau grup instans yang berbeda menjalankan peran yang berbeda. Misalnya, instans primer menangani semua laporan bahasa definisi data (DDL) dan bahasa manipulasi data (DML). Hingga 15 Replika Aurora dapat menangani lalu lintas kueri hanya baca.

Dengan menggunakan titik akhir, Anda dapat memetakan setiap koneksi ke instans atau grup instans yang sesuai berdasarkan kasus penggunaan Anda. Misalnya, untuk melakukan laporan DDL, Anda dapat melakukan koneksi ke instans mana pun yang merupakan instans primer. Untuk melakukan kueri, Anda dapat melakukan koneksi ke titik akhir pembaca, dengan Aurora yang secara otomatis melakukan penyeimbangan beban di antara semua Replika Aurora. Untuk klaster dengan instans DB yang memiliki kapasitas atau konfigurasi berbeda, Anda dapat melakukan koneksi ke titik akhir kustom yang terkait dengan subset instans DB yang berbeda. Untuk diagnosis atau penyetelan, Anda

dapat melakukan koneksi ke titik akhir instans tertentu untuk memeriksa detail tentang instans DB tertentu.

Topik

- Jenis titik akhir Aurora
- Melihat titik akhir untuk klaster Aurora
- Menggunakan endpoint klaster
- Menggunakan titik akhir pembaca
- Menggunakan titik akhir kustom
- Membuat titik akhir kustom
- Melihat titik akhir kustom
- Mengedit titik akhir kustom
- Menghapus titik akhir kustom
- nd-to-end AWS CLIContoh E untuk titik akhir kustom
- Menggunakan titik akhir instans
- Bagaimana titik akhir Aurora berfungsi dengan ketersediaan tinggi

Jenis titik akhir Aurora

Titik akhir direpresentasikan sebagai URL khusus Aurora yang berisi alamat host dan port. Jenis titik akhir berikut tersedia dari klaster DB Aurora.

Titik akhir klaster

> Titik akhir klaster (atau titik akhir penulis) untuk sebuah klaster DB Aurora terhubung ke instans DB primer saat ini untuk klaster DB tersebut. Titik akhir ini adalah satu-satunya yang dapat melakukan operasi tulis seperti laporan DDL. Oleh karena itu, titik akhir klaster adalah titik akhir tempat Anda melakukan koneksi saat Anda pertama kali mengatur klaster atau ketika klaster Anda hanya berisi satu instans DB.
>
> Setiap klaster DB Aurora memiliki satu titik akhir klaster dan satu instans DB primer.
>
> Anda menggunakan titik akhir klaster untuk semua operasi tulis pada klaster DB tersebut, termasuk penyisipan, pembaruan, penghapusan, dan perubahan DDL. Anda juga dapat menggunakan titik akhir klaster untuk operasi baca, seperti kueri.

Titik akhir klaster memberikan dukungan failover untuk koneksi baca/tulis ke klaster DB. Jika instans DB primer saat ini pada sebuah klaster DB gagal, Aurora secara otomatis melakukan failover ke instans DB primer yang baru. Selama failover, klaster DB tersebut terus melayani permintaan koneksi ke titik akhir klaster dari instans DB primer yang baru, dengan interupsi layanan yang minimal.

Contoh berikut mengilustrasikan titik akhir klaster untuk sebuah klaster DB Aurora MySQL.

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

Titik akhir pembaca

Titik akhir pembaca untuk sebuah klaster DB Aurora memberikan dukungan penyeimbangan beban untuk koneksi hanya baca ke klaster DB. Gunakan titik akhir pembaca untuk operasi baca, seperti kueri. Dengan memproses laporan tersebut di Replika Aurora hanya baca, titik akhir ini mengurangi overhead pada instans primer. Hal ini juga membantu klaster menskalakan kapasitas untuk menangani kueri `SELECT` simultan, berbanding lurus dengan jumlah Replika Aurora di klaster. Setiap klaster DB Aurora memiliki satu titik akhir pembaca.

Jika klaster berisi satu atau beberapa Replika Aurora, maka titik akhir pembaca melakukan penyeimbangan beban untuk setiap permintaan koneksi di antara beberapa Replika Aurora. Dalam hal ini, Anda hanya dapat menjalankan pernyataan hanya baca seperti `SELECT` dalam sesi tersebut. Jika klaster hanya berisi satu instans primer dan tidak ada Replika Aurora, titik akhir pembaca terhubung ke instans primer. Dengan begitu, Anda dapat melakukan operasi baca melalui titik akhir.

Contoh berikut mengilustrasikan titik akhir pembaca untuk sebuah klaster DB Aurora MySQL.

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

Titik akhir kustom

Titik akhir kustom untuk sebuah klaster Aurora merepresentasikan serangkaian instans DB yang Anda pilih. Saat Anda melakukan koneksi ke titik akhir, Aurora melakukan penyeimbangan beban dan memilih salah satu instans dalam grup untuk menangani koneksi tersebut. Anda menentukan instans mana yang dirujuk oleh titik akhir, dan Anda memutuskan apa tujuan dari titik akhir tersebut.

Sebuah klaster DB Aurora tidak akan memiliki titik akhir kustom sebelum Anda membuatnya. Anda dapat membuat hingga lima titik akhir kustom untuk setiap klaster Aurora terprovisi atau klaster Aurora Serverless v2. Anda tidak dapat menggunakan titik akhir kustom untuk klaster Aurora Serverless v1.

Titik akhir kustom menyediakan koneksi basis data yang diseimbangkan bebannya berdasarkan kriteria selain kemampuan hanya baca atau baca/tulis instans DB. Misalnya, Anda dapat menentukan titik akhir kustom untuk melakukan koneksi ke instans yang menggunakan kelas instans AWS tertentu atau grup parameter DB tertentu. Selanjutnya Anda dapat memberi tahu kelompok pengguna tertentu tentang titik akhir kustom ini. Misalnya, Anda dapat mengarahkan pengguna internal ke instans berkapasitas rendah untuk pembuatan laporan atau kueri ad hoc (sekali waktu), dan mengarahkan lalu lintas produksi ke instans berkapasitas tinggi.

Karena koneksinya dapat menuju ke setiap instans DB yang dikaitkan dengan titik akhir kustom, kami menyarankan Anda untuk memastikan bahwa semua instans DB yang berada dalam grup tersebut memiliki beberapa karakteristik serupa. Hal tersebut akan memastikan bahwa performa, kapasitas memori, dsb, konsisten untuk semua orang yang melakukan koneksi ke titik akhir tersebut.

Fitur ini ditujukan bagi pengguna tingkat lanjut dengan beban kerja khusus yang kebutuhannya tidak akan terpenuhi jika semua Replika Aurora di klaster tetap identik. Dengan titik akhir kustom, Anda dapat memprediksi kapasitas instans DB yang digunakan untuk setiap koneksi. Saat Anda menggunakan titik akhir kustom, Anda biasanya tidak menggunakan titik akhir pembaca untuk klaster tersebut.

Contoh berikut mengilustrasikan titik akhir kustom untuk sebuah instans DB di sebuah klaster DB Aurora MySQL.

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

Titik akhir instans

Titik akhir instans melakukan koneksi ke sebuah instans DB tertentu di dalam klaster Aurora. Setiap instans DB dalam sebuah klaster DB memiliki titik akhir yang unik. Jadi, ada satu titik akhir instans untuk instans DB primer saat ini di klaster DB, dan ada satu titik akhir untuk setiap Replika Aurora di klaster DB tersebut.

Titik akhir instans menyediakan kontrol langsung atas koneksi ke klaster DB untuk skenario yang tidak memungkinkan penggunaan titik akhir klaster atau titik akhir pembaca. Misalnya, aplikasi klien Anda mungkin membutuhkan penyeimbangan beban yang lebih terperinci berdasarkan jenis beban kerja. Dalam hal ini, Anda dapat mengonfigurasi beberapa klien untuk melakukan koneksi ke Replika Aurora yang berbeda dalam klaster DB untuk mendistribusikan beban kerja baca. Untuk mengetahui contoh yang menggunakan titik akhir instans untuk meningkatkan kecepatan koneksi setelah failover untuk Aurora PostgreSQL, lihat Failover cepat dengan Amazon Aurora PostgreSQL. Untuk mengetahui contoh yang menggunakan titik akhir instans untuk meningkatkan

kecepatan koneksi setelah failover untuk Aurora MySQL, lihat Dukungan failover Connector/J MariaDB – kasus Amazon Aurora.

Contoh berikut mengilustrasikan titik akhir instans untuk sebuah instans DB di sebuah klaster DB Aurora MySQL.

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

Melihat titik akhir untuk klaster Aurora

Dalam AWS Management Console, Anda melihat titik akhir klaster, titik akhir pembaca, dan setiap titik akhir kustom di halaman detail untuk setiap klaster. Anda melihat titik akhir instans di halaman detail untuk setiap instans. Saat Anda terhubung, Anda harus menambahkan nomor port terkait, setelah simbol titik dua, ke nama titik akhir yang ditampilkan di halaman detail ini.

DenganAWS CLI, Anda melihat penulis, pembaca, dan titik akhir khusus apa pun dalam output describe-db-clustersperintah. Misalnya, perintah berikut menampilkan atribut-atribut titik akhir untuk semua klaster di Kawasan AWS Anda saat ini.

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

Dengan Amazon RDS API, Anda mengambil titik akhir dengan memanggil fungsi describeDB. ClusterEndpoints

Menggunakan endpoint klaster

Karena setiap klaster Aurora memiliki satu titik akhir klaster bawaan, yang nama dan atribut lainnya dikelola oleh Aurora, Anda tidak dapat membuat, menghapus, atau mengubah jenis titik akhir ini.

Anda menggunakan titik akhir klaster saat mengoperasikan klaster Anda, melakukan operasi extract, transform, and load (ETL), atau mengembangkan dan menguji aplikasi. Titik akhir klaster terhubung ke instans primer dari klaster tersebut. Instans primer adalah satu-satunya instans DB yang memungkinkan Anda membuat tabel dan indeks, menjalankan pernyataan `INSERT`, dan melakukan operasi DDL dan DML lainnya.

Alamat IP fisik yang ditunjuk oleh titik akhir klaster akan berubah ketika mekanisme failover mempromosikan instans DB baru untuk menjadi instans primer baca/tulis untuk klaster tersebut. Jika Anda menggunakan segala bentuk penyatuan koneksi atau multiplexing lainnya, bersiaplah untuk menyiram atau mengurangi informasi DNS yang time-to-live di-cache. Melakukan hal itu memastikan

bahwa Anda tidak mencoba membentuk koneksi baca/tulis ke instans basis data yang menjadi tidak tersedia atau kini hanya baca setelah terjadi pindah saat gagal/failover.

Menggunakan titik akhir pembaca

Anda menggunakan titik akhir pembaca untuk koneksi hanya baca untuk klaster Aurora Anda. Titik akhir ini menggunakan mekanisme penyeimbangan beban untuk membantu klaster Anda menangani beban kerja sarat kueri. Titik akhir pembaca adalah titik akhir yang Anda berikan ke aplikasi yang melakukan pelaporan atau operasi hanya baca lain pada klaster.

Titik akhir pembaca menyeimbangkan beban koneksi ke Replika Aurora di klaster DB Aurora. Titik akhir ini tidak menyeimbangkan beban kueri individual. Jika Anda ingin melakukan penyeimbangan beban untuk setiap kueri guna mendistribusikan beban kerja baca untuk klaster DB, buka koneksi baru ke titik akhir pembaca untuk setiap kueri.

Setiap klaster Aurora memiliki satu titik akhir pembaca bawaan, yang nama dan atribut lainnya dikelola oleh Aurora. Anda tidak dapat membuat, menghapus, atau memodifikasi titik akhir semacam ini.

Jika klaster Anda hanya berisi instans primer dan tidak ada Replika Aurora, titik akhir pembaca akan terhubung ke instans primer. Dalam hal ini, Anda dapat melakukan operasi tulis melalui titik akhir ini.

> **Tip**
>
> Melalui Proksi RDS, Anda dapat membuat titik akhir hanya baca tambahan untuk sebuah klaster Aurora. Titik akhir ini melakukan jenis penyeimbangan beban yang sama seperti titik akhir pembaca Aurora. Aplikasi dapat terhubung kembali lebih cepat ke titik akhir proksi daripada titik akhir pembaca Aurora jika instans pembaca menjadi tidak tersedia. Titik akhir proksi juga dapat memanfaatkan fitur proksi lainnya seperti multiplexing. Untuk informasi selengkapnya, lihat Menggunakan titik akhir pembaca dengan klaster Aurora.

Menggunakan titik akhir kustom

Anda menggunakan titik akhir kustom untuk menyederhanakan manajemen koneksi saat klaster Anda berisi beberapa instans DB dengan pengaturan kapasitas dan konfigurasi yang berbeda-beda.

Sebelumnya, Anda mungkin telah menggunakan mekanisme CNAMES untuk mengatur alias Layanan Nama Domain (DNS) dari domain Anda sendiri untuk mendapatkan hasil yang sama. Dengan menggunakan titik akhir kustom, Anda tidak perlu memperbarui catatan CNAME ketika

klaster Anda membesar atau menyusut. Titik akhir kustom juga berarti bahwa Anda dapat menggunakan koneksi Keamanan Lapisan Pengangkutan/Lapisan Soket Aman (TLS/SSL).

Daripada menggunakan satu instans DB untuk setiap tujuan khusus dan melakukan koneksi titik akhir instans, Anda dapat memiliki beberapa grup instans DB khusus. Dalam hal ini, setiap grup memiliki titik akhir kustom tersendiri. Dengan cara ini, Aurora dapat melakukan penyeimbangan beban di antara semua instans yang dikhususkan untuk tugas-tugas seperti pelaporan atau menangani produksi atau kueri internal. Titik akhir kustom memberikan penyeimbangan beban dan ketersediaan tinggi untuk setiap grup instans DB di dalam klaster Anda. Jika salah satu instans DB dalam sebuah grup tidak tersedia, Aurora mengarahkan koneksi titik akhir kustom berikutnya ke salah satu instans DB yang terkait dengan titik akhir yang sama.

Topik

- Menentukan properti untuk titik akhir kustom
- Aturan keanggotaan untuk titik akhir kustom
- Mengelola titik akhir kustom

Menentukan properti untuk titik akhir kustom

Panjang maksimum untuk nama titik akhir kustom adalah 63 karakter. Format namanya adalah sebagai berikut:

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

Anda tidak dapat menggunakan kembali nama titik akhir kustom yang sama untuk lebih dari satu klaster di Wilayah AWS yang sama. Pengidentifikasi DNS pelanggan adalah pengidentifikasi unik yang terkait dengan Akun AWS Anda di Wilayah AWS tertentu.

Setiap titik akhir kustom memiliki jenis terkait yang menentukan instans DB mana yang memenuhi syarat untuk dikaitkan dengan titik akhir tersebut. Saat ini, jenisnya bisa berupa `READER`, `WRITER`, atau `ANY`. Pertimbangan berikut berlaku untuk jenis titik akhir kustom:

- Anda tidak dapat memilih jenis titik akhir kustom di AWS Management Console. Semua titik akhir kustom yang Anda buat melalui AWS Management Console memiliki jenis `ANY`.

  Anda dapat mengatur dan memodifikasi jenis titik akhir kustom menggunakan AWS CLI atau API Amazon RDS.
- Hanya instans DB pembaca yang dapat menjadi bagian dari titik akhir kustom `READER`.

- Instans DB pembaca dan penulis dapat menjadi bagian dari titik akhir kustom ANY. Aurora mengarahkan koneksi ke titik akhir klaster dengan jenis ANY ke instans DB terkait mana saja dengan probabilitas yang sama. Jenis ANY berlaku untuk klaster yang menggunakan topologi replikasi mana pun.
- Jika Anda mencoba membuat titik akhir kustom dengan jenis yang tidak sesuai berdasarkan konfigurasi replikasi untuk klaster, Aurora akan memunculkan pesan kesalahan.

Aturan keanggotaan untuk titik akhir kustom

Saat Anda menambahkan instans DB ke titik akhir kustom atau menghapusnya dari titik akhir kustom, koneksi apa pun yang ada ke instans DB tersebut tetap aktif.

Anda dapat menetapkan daftar instans DB untuk disertakan dalam, atau dikecualikan dari, sebuah titik akhir kustom. Kami menyebut daftar ini masing-masing sebagai daftar statis dan pengecualian. Anda dapat menggunakan mekanisme penyertaan/pengecualian untuk membagi kembali grup instans DB, dan untuk memastikan bahwa rangkaian titik akhir kustom tersebut telah mencakup semua instans DB dalam klaster. Setiap titik akhir kustom hanya dapat berisi salah satu jenis daftar tersebut.

Di AWS Management Console:

- Pilihannya ditunjukkan oleh kotak centang Lampirkan instans masa depan yang ditambahkan ke klaster ini. Jika Anda membiarkan kotak centang ini kosong, titik akhir kustom akan menggunakan daftar statis yang hanya berisi instans DB yang ditentukan pada halaman. Jika Anda memilih kotak centang ini, titik akhir kustom akan menggunakan daftar pengecualian. Dalam hal ini, titik akhir kustom merepresentasikan semua instans DB di dalam klaster (termasuk yang Anda tambahkan di masa mendatang) kecuali yang tidak dipilih di halaman tersebut.
- Konsol tidak memungkinkan Anda menentukan jenis titik akhir. Titik akhir kustom apa pun yang dibuat menggunakan konsol akan memiliki jenis ANY.

  Oleh karena itu, Aurora tidak mengubah keanggotaan titik akhir kustom ketika instans DB berganti peran antara penulis dan pembaca karena failover atau promosi.

Di AWS CLI dan API Amazon RDS:

- Anda dapat menentukan jenis titik akhir. Oleh karena itu, ketika jenis titik akhir diatur ke READER atau WRITER, keanggotaan titik akhir secara otomatis disesuaikan selama failover dan promosi.

Misalnya, titik akhir kustom dengan jenis `READER` mencakup Replika Aurora yang kemudian dipromosikan menjadi instans penulis. Instans penulis baru ini tidak lagi menjadi bagian dari titik akhir kustom.

- Anda dapat menambahkan anggota individu ke dan menghapusnya dari daftar setelah perannya berubah. Gunakan modify-db-cluster-endpointAWS CLIperintah atau operasi API ModifyDB ClusterEndpoint.

Anda dapat mengaitkan instans DB dengan lebih dari satu titik akhir kustom. Misalnya, anggaplah Anda menambahkan instans DB baru ke sebuah klaster, atau Aurora menambahkan instans DB secara otomatis melalui mekanisme penskalaan otomatis. Dalam hal ini, instans DB ditambahkan ke semua titik akhir kustom yang memenuhi syarat. Titik akhir mana yang ditambahkan ke instans DB akan ditentukan berdasarkan jenis titik akhir kustom `READER`, `WRITER`, atau `ANY`, ditambah daftar statis atau pengecualian yang ditentukan untuk setiap titik akhir. Misalnya, jika titik akhir tersebut mencakup daftar statis instans DB, Replika Aurora yang baru tidak ditambahkan ke titik akhir tersebut. Sebaliknya, jika titik akhir tersebut memiliki daftar pengecualian, Replika Aurora yang baru akan ditambahkan ke titik akhir, jika tidak disebutkan dalam daftar pengecualian dan perannya cocok dengan jenis titik akhir kustom tersebut.

Jika Replika Aurora menjadi tidak tersedia, Replika Aurora ini akan tetap dikaitkan dengan titik akhir kustom. Misalnya, Replika Aurora tetap menjadi bagian dari titik akhir kustom jika tidak berkondisi baik, dihentikan, di-boot ulang, dan seterusnya. Namun, Anda tidak dapat melakukan koneksi ke Replika Aurora melalui titik akhir tersebut hingga tersedia kembali.

Mengelola titik akhir kustom

Karena klaster Aurora yang baru dibuat tidak memiliki titik akhir kustom, Anda harus membuat dan mengelola objek ini sendiri. Anda melakukannya menggunakan AWS Management Console, AWS CLI, atau API Amazon RDS.

> **Note**
>
> Anda juga harus membuat dan mengelola titik akhir kustom untuk klaster Aurora yang dipulihkan dari snapshot. Titik akhir kustom tidak disertakan dalam snapshot. Anda membuatnya kembali setelah memulihkan, dan memilih nama titik akhir baru jika klaster yang dipulihkan berada di wilayah yang sama dengan yang asli.

Untuk menggunakan titik akhir kustom dari AWS Management Console, Anda perlu membuka halaman detail untuk klaster Aurora Anda dan menggunakan kontrol di bagian Titik Akhir Kustom.

Untuk menggunakan titik akhir kustom dari AWS CLI, Anda dapat menggunakan operasi ini:

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

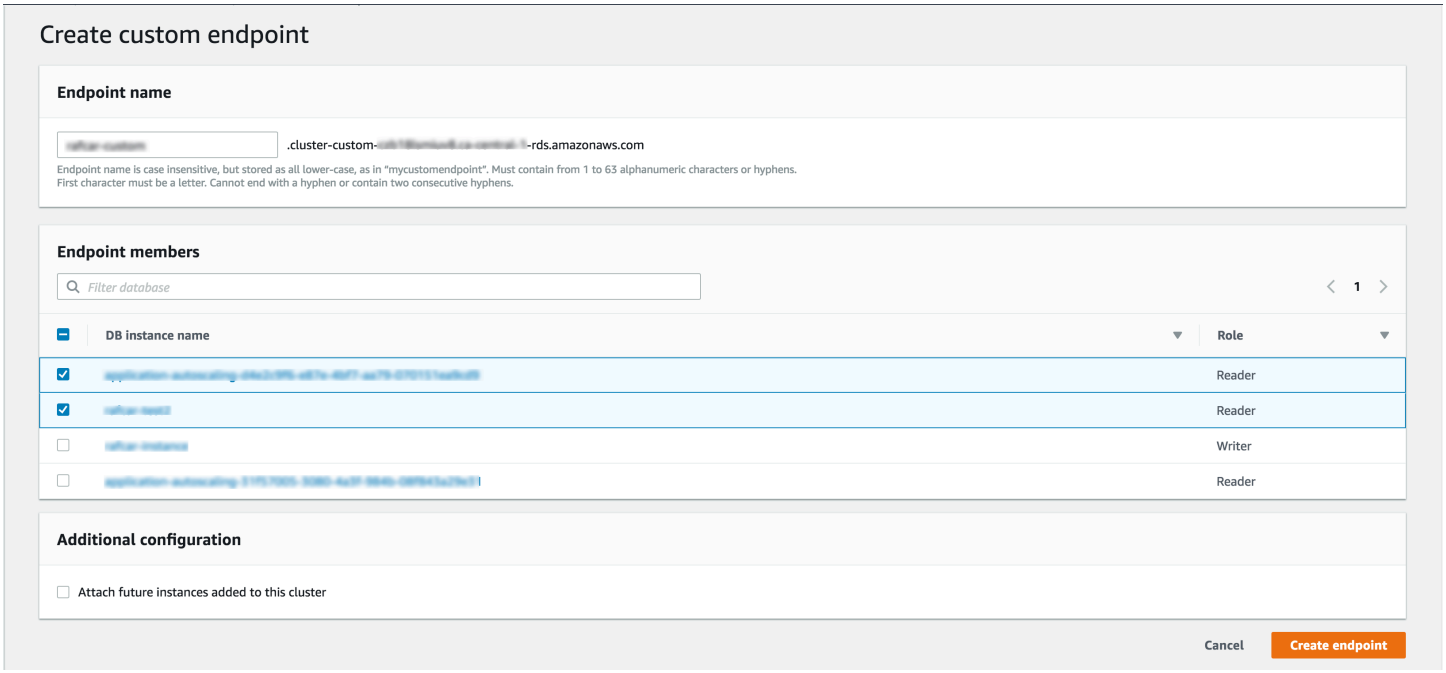
Untuk menangani titik akhir kustom melalui API Amazon RDS, Anda dapat menggunakan fungsi berikut:

- dibuatB ClusterEndpoint
- DijelaskanB ClusterEndpoints
- ModifyDB ClusterEndpoint
- DihapusB ClusterEndpoint

Membuat titik akhir kustom

Konsol

Untuk membuat titik akhir kustom dengan AWS Management Console, buka halaman detail klaster dan pilih tindakan `Create custom endpoint` di bagian Titik Akhir. Pilih nama untuk titik akhir kustom, yang unik untuk ID pengguna dan wilayah Anda. Untuk memilih daftar instans DB yang tetap sama bahkan saat ukuran klaster bertambah, kosongkan kotak centang Lampirkan instans masa depan yang ditambahkan ke klaster ini. Saat Anda memilih kotak centang tersebut, titik akhir kustom akan secara dinamis menambahkan instans baru saat Anda menambahkannya ke klaster.



Anda tidak dapat memilih jenis titik akhir kustom ANY atau READER di AWS Management Console. Semua titik akhir kustom yang Anda buat melalui AWS Management Console memiliki jenis ANY.

AWS CLI

Untuk membuat endpoint kustom denganAWS CLI, jalankan create-db-cluster-endpointperintah.

Perintah berikut membuat titik akhir kustom yang dilampirkan ke klaster tertentu. Awalnya, titik akhir ini dikaitkan dengan semua instans Replika Aurora di dalam klaster. Perintah berikutnya mengaitkannya dengan serangkaian instans DB tertentu dalam klaster.

Untuk Linux, macOS, atau Unix:

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

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

Untuk Windows:

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

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

API RDS

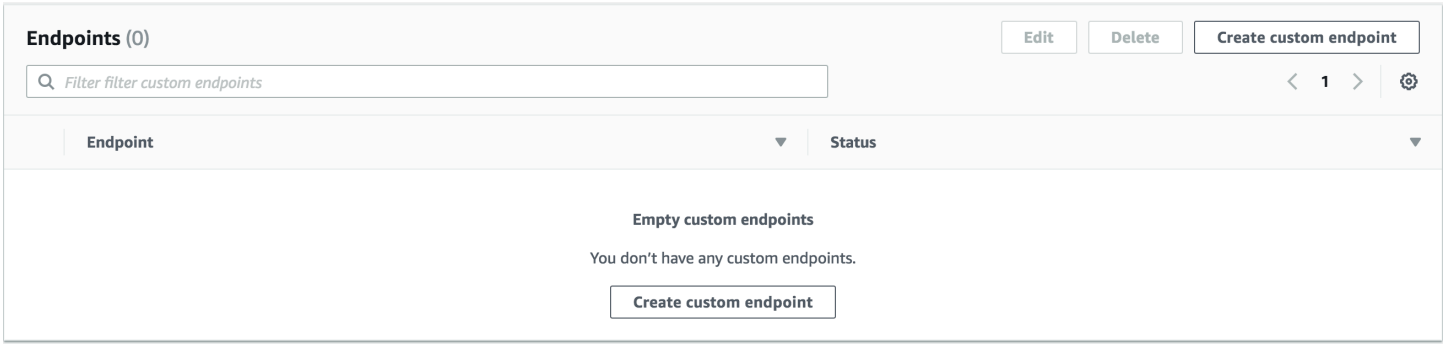
Untuk membuat endpoint kustom dengan RDS API, jalankan operasi ClusterEndpointcreateDB.

Melihat titik akhir kustom

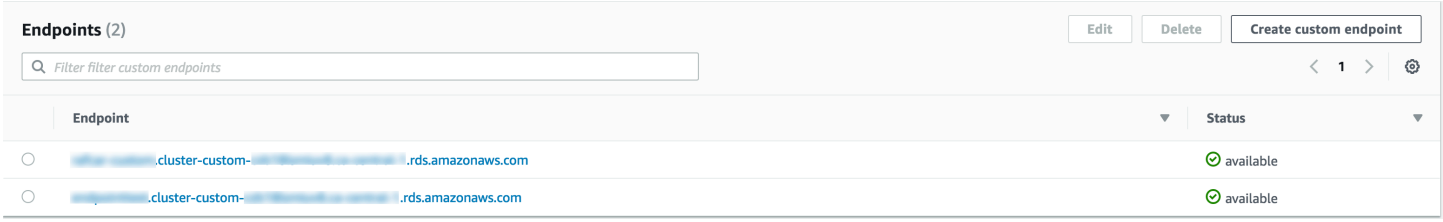
Konsol

Untuk melihat titik akhir kustom dengan AWS Management Console, kunjungi halaman detail klaster untuk klaster tersebut dan lihat di bagian Titik Akhir. Bagian ini berisi informasi tentang titik akhir kustom saja. Detail untuk titik akhir bawaan tercantum dalam bagian Detail utama. Untuk melihat detail titik akhir kustom tertentu, pilih nama titik akhir guna menampilkan halaman detail untuk titik akhir tersebut.

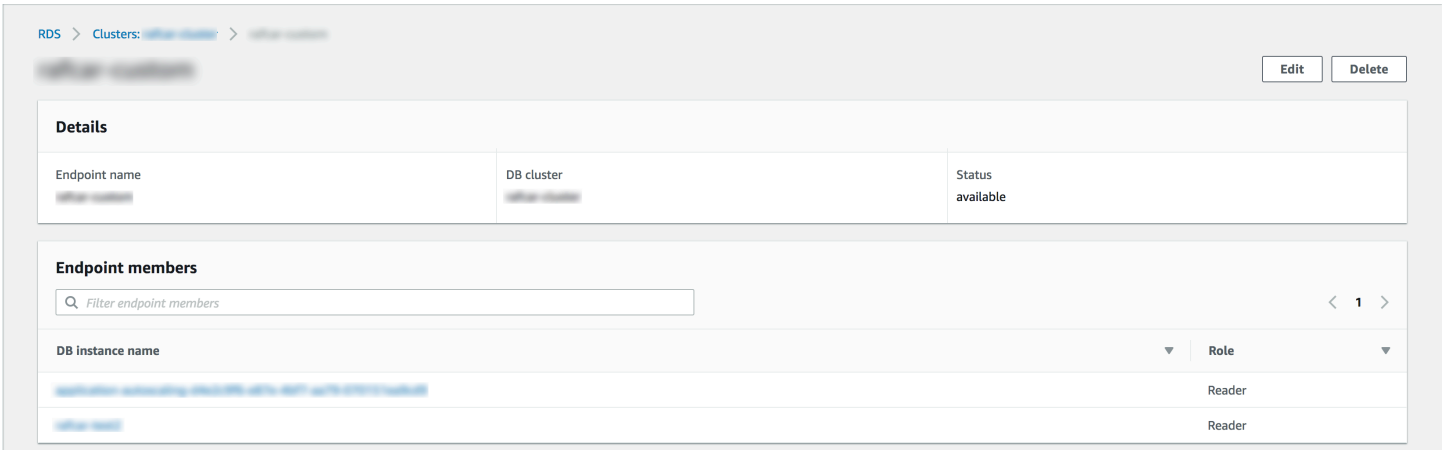
Tangkapan layar berikut menunjukkan bahwa daftar titik akhir kustom untuk klaster Aurora awalnya kosong.



Setelah Anda membuat beberapa titik akhir kustom untuk klaster tersebut, titik akhir ini ditampilkan di bagian Titik Akhir.



Dengan mengklik halaman detail, akan terlihat instans DB mana yang saat ini dikaitkan dengan titik akhir tersebut.



Untuk melihat detail tambahan tentang apakah instans DB yang baru ditambahkan ke klaster juga secara otomatis ditambahkan ke titik akhir, buka halaman Edit untuk titik akhir.

AWS CLI

Untuk melihat titik akhir kustom denganAWS CLI, jalankan describe-db-cluster-endpointsperintah.

Perintah berikut akan menampilkan titik akhir kustom yang terkait dengan klaster tertentu di wilayah tertentu. Output-nya mencakup titik akhir bawaan dan titik akhir kustom.

Untuk Linux, macOS, atau Unix:

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

Untuk Windows:

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

Hal berikut menunjukkan beberapa output sampel dari perintah `describe-db-cluster-endpoints`. `EndpointType` yang bernilai `WRITER` atau `READER` menunjukkan titik akhir baca/tulis dan titik akhir hanya baca bawaan untuk klaster tersebut. `EndpointType` yang bernilai `CUSTOM` menunjukkan titik akhir yang Anda buat dan pilih untuk instans DB terkait. Salah satu titik akhir memiliki bidang `StaticMembers` yang terisi, yang menunjukkan bahwa titik akhir tersebut dikaitkan dengan rangkaian instans DB spesifik. Titik akhir lainnya memiliki bidang `ExcludedMembers` yang terisi, menunjukkan bahwa titik akhir ini dikaitkan dengan semua instans DB selain yang tercantum dalam `ExcludedMembers`. Jenis kedua dari titik akhir kustom ini akan bertambah untuk mengakomodasi instans baru saat Anda menambahkannya ke klaster.

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

```
   "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
   "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:powers-of-2"
  },
  {
   "CustomEndpointType": "ANY",
   "DBClusterEndpointIdentifier": "eight-and-higher",
   "ExcludedMembers": [
     "custom-endpoint-demo-04",
     "custom-endpoint-demo-02",
     "custom-endpoint-demo-07",
     "custom-endpoint-demo-05",
     "custom-endpoint-demo-03",
     "custom-endpoint-demo-06",
     "custom-endpoint-demo-01"
   ],
   "DBClusterIdentifier": "custom-endpoint-demo",
   "Status": "available",
   "EndpointType": "CUSTOM",
   "Endpoint": "eight-and-higher.cluster-custom-123456789012.ca-
central-1.rds.amazonaws.com",
   "StaticMembers": [],
   "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHYQKFU6J6NV5FHU",
   "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:eight-and-higher"
  }
 ]
}
```

API RDS

Untuk melihat titik akhir kustom dengan RDS API, jalankan operasi ClusterEndpointsDescribeDB.html.

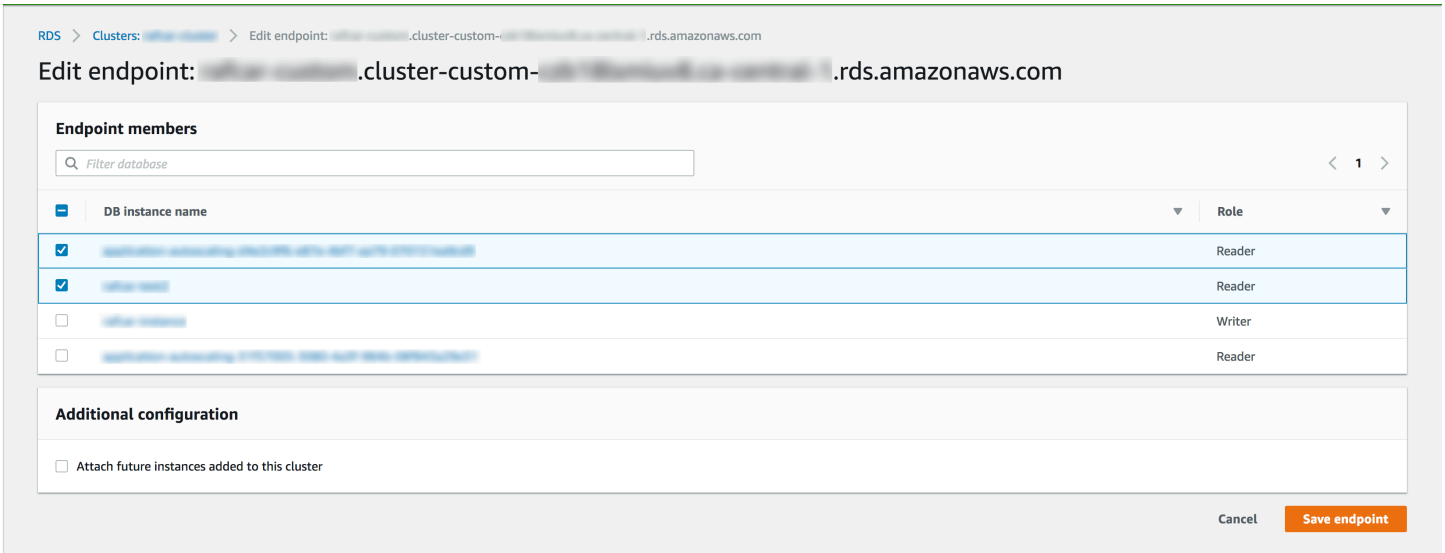
Mengedit titik akhir kustom

Anda dapat mengubah mengedit titik akhir kustom untuk mengubah instans DB mana yang dikaitkan dengan titik akhir tersebut. Anda juga dapat mengubah titik akhir di antara daftar statis dan daftar pengecualian. Jika Anda membutuhkan detail selengkapnya tentang properti titik akhir ini, lihat Aturan keanggotaan untuk titik akhir kustom.

Anda dapat terus terhubung ke dan menggunakan titik akhir kustom saat perubahan dari tindakan edit sedang berlangsung.

Konsol

Untuk mengedit titik akhir kustom dengan AWS Management Console, Anda dapat memilih titik akhir di halaman detail klaster, atau memunculkan halaman detail untuk titik akhir tersebut, dan memilih tindakan Edit.



AWS CLI

Untuk mengedit titik akhir kustom denganAWS CLI, jalankan modify-db-cluster-endpointperintah.

Perintah berikut akan mengubah rangkaian instans DB yang diterapkan ke titik akhir kustom dan secara opsional beralih di antara perilaku daftar statis atau daftar pengecualian. Parameter `--static-members` dan `--excluded-members` mengambil daftar pengidentifikasi instans DB yang dipisahkan oleh spasi.

Untuk Linux, macOS, atau Unix:

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

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

```
  --region region_name
```

Untuk Windows:

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

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

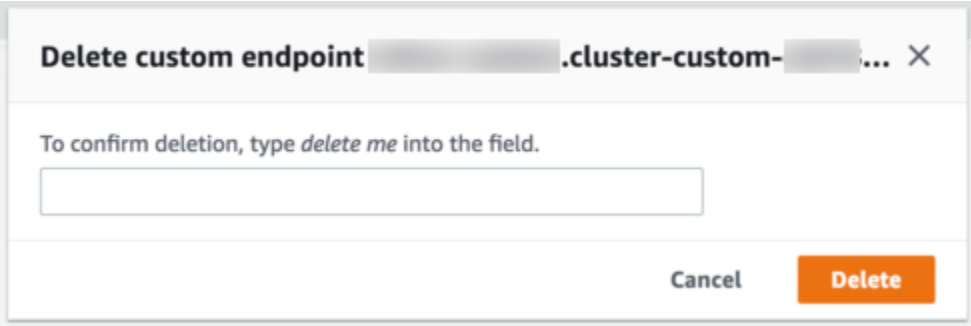
API RDS

Untuk mengedit titik akhir kustom dengan RDS API, jalankan operasi ModifyDB ClusterEndpoint .html.

Menghapus titik akhir kustom

Konsol

Untuk menghapus titik akhir kustom dengan AWS Management Console, buka halaman detail klaster, pilih titik akhir kustom yang sesuai, dan pilih tindakan Hapus.



AWS CLI

Untuk menghapus titik akhir kustom denganAWS CLI, jalankan delete-db-cluster-endpointperintah.

Perintah berikut akan menghapus titik akhir kustom. Anda tidak perlu menentukan klaster terkait, tetapi Anda harus menentukan wilayah.

Untuk Linux, macOS, atau Unix:

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

Untuk Windows:

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

API RDS

Untuk menghapus titik akhir kustom dengan RDS API, jalankan operasi ClusterEndpointDeleteDB.

nd-to-end AWS CLIContoh E untuk titik akhir kustom

Tutorial berikut menggunakan contoh AWS CLI dengan sintaksis Unix shell untuk menunjukkan bahwa Anda dapat menentukan klaster dengan beberapa instans DB “kecil” dan beberapa instans DB “besar”, lalu membuat titik akhir kustom untuk terhubung ke setiap rangkaian instans DB. Untuk menjalankan perintah yang serupa pada sistem Anda sendiri, Anda harus memahami cara kerja klaster Aurora dan penggunaan AWS CLI untuk memasukkan nilai Anda sendiri untuk parameter seperti wilayah, grup subnet, atau grup keamanan VPC.

Contoh ini menunjukkan langkah pengaturan awal: membuat klaster Aurora dan menambahkan instans DB ke dalamnya. Ini adalah klaster heterogen, yang berarti tidak semua instans DB memiliki kapasitas yang sama. Sebagian besar instans menggunakan kelas instans AWS `db.r4.4xlarge`, tetapi dua instans DB terakhir menggunakan `db.r4.16xlarge`. Masing-masing sampel perintah `create-db-instance` ini mencetak output-nya ke layar dan menyimpan salinan JSON dalam file untuk pemeriksaan di lain waktu.

```
aws rds create-db-cluster --db-cluster-identifier custom-endpoint-demo --engine aurora-
mysql \
     --engine-version 8.0.mysql_aurora.3.02.0 --master-username $MASTER_USER --manage-
master-user-password \
     --db-subnet-group-name $SUBNET_GROUP  --vpc-security-group-ids $VPC_SECURITY_GROUP
 \
     --region $REGION

for i in 01 02 03 04 05 06 07 08
do
```

```
  aws rds create-db-instance --db-instance-identifier custom-endpoint-demo-${i} \
     --engine aurora --db-cluster-identifier custom-endpoint-demo --db-instance-class
 db.r4.4xlarge \
     --region $REGION \
     | tee custom-endpoint-demo-${i}.json
done

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

Instans yang lebih besar dicadangkan untuk jenis kueri pelaporan khusus. Agar instans tersebut tidak dipromosikan menjadi instans primer, contoh berikut ini mengubah tingkat promosinya ke prioritas terendah. Contoh ini menentukan opsi `--manage-master-user-password` untuk menghasilkan kata sandi pengguna master dan mengelolanya di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. Alternatifnya, Anda dapat menggunakan opsi `--master-password` untuk menentukan dan mengelola kata sandi sendiri.

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

Misalnya, Anda ingin menggunakan dua instans “lebih besar” hanya untuk kueri yang membutuhkan banyak sumber daya. Untuk melakukannya, Anda dapat membuat titik akhir hanya baca kustom. Kemudian Anda dapat menambahkan daftar statis anggota sehingga titik akhir hanya terhubung ke instans DB tersebut. Instans tersebut sudah berada di tingkat promosi terendah, sehingga tidak mungkin dipromosikan menjadi instans primer. Jika salah satunya dipromosikan menjadi instans utama, instans tersebut menjadi tidak dapat dijangkau melalui titik akhir ini karena kita menetapkan jenis `READER` dan bukan jenis `ANY`.

Contoh berikut menunjukkan perintah pembuatan dan modifikasi titik akhir, serta sampel output JSON yang menunjukkan status awal dan status yang dimodifikasi untuk titik akhir kustom tersebut.

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

$ aws rds modify-db-cluster-endpoint --db-cluster-endpoint-identifier big-instances \
    --static-members custom-endpoint-demo-09 custom-endpoint-demo-10 --region $REGION
{
    "EndpointType": "CUSTOM",
    "ExcludedMembers": [],
    "DBClusterEndpointIdentifier": "big-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-
W7PE3TLLFNSHXQKFU6J6NV5FHU",
    "CustomEndpointType": "READER",
    "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:big-instances",
    "StaticMembers": [
        "custom-endpoint-demo-10",
        "custom-endpoint-demo-09"
    ],
    "Status": "modifying",
    "Endpoint": "big-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
    "DBClusterIdentifier": "custom-endpoint-demo"
}
```

Titik akhir READER default untuk klaster dapat terhubung ke instans DB “kecil” atau “besar”, sehingga tidak praktis untuk memprediksi performa dan skalabilitas kueri ketika klaster menjadi sibuk. Untuk membagi beban kerja secara merata di antara sejumlah rangkaian instans DB, Anda dapat

mengabaikan titik akhir READER default dan membuat titik akhir kustom kedua yang terhubung ke semua instans DB lainnya. Contoh berikut melakukannya dengan membuat titik akhir kustom, lalu menambahkan daftar pengecualian. Instans DB lain yang Anda tambahkan ke klaster di masa mendatang akan secara otomatis ditambahkan ke titik akhir ini. Jenis ANY berarti bahwa titik akhir ini terkait dengan total delapan instans: instans primer dan tujuh Replika Aurora lainnya. Jika contohnya menggunakan jenis READER, titik akhir kustom hanya akan dikaitkan dengan tujuh Replika Aurora.

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

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

```
    "DBClusterIdentifier": "custom-endpoint-demo",
    "Status": "modifying"
}
```

Contoh berikut memeriksa status titik akhir untuk klaster ini. Klaster ini masih memiliki titik akhir klaster aslinya, dengan `EndPointType` yang bernilai `WRITER`, yang masih akan Anda gunakan untuk operasi administrasi, ETL, dan operasi tulis lainnya. Klaster ini masih memiliki titik akhir `READER` aslinya, yang tidak akan Anda gunakan karena setiap koneksi yang diterimanya dapat diarahkan ke instans DB "kecil" atau "besar". Dengan titik akhir kustom, perilaku tersebut menjadi dapat diprediksi, dengan koneksi yang dipastikan akan menggunakan salah satu instans DB “kecil” atau “besar” berdasarkan titik akhir yang Anda tentukan.

```
$ aws rds describe-db-cluster-endpoints --region $REGION
{
    "DBClusterEndpoints": [
        {
            "EndpointType": "WRITER",
            "Endpoint": "custom-endpoint-demo.cluster-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
            "EndpointType": "READER",
            "Endpoint": "custom-endpoint-demo.cluster-ro-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
            "Endpoint": "small-instances.cluster-custom-c7tj4example.ca-
central-1.rds.amazonaws.com",
            "CustomEndpointType": "ANY",
            "DBClusterEndpointArn": "arn:aws:rds:ca-central-1:111122223333:cluster-
endpoint:small-instances",
            "ExcludedMembers": [
                "custom-endpoint-demo-09",
                "custom-endpoint-demo-10"
            ],
            "DBClusterEndpointResourceIdentifier": "cluster-
endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
            "DBClusterIdentifier": "custom-endpoint-demo",
```

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

Contoh terakhir menunjukkan koneksi basis data berurutan ke titik akhir kustom terhubung dengan berbagai instans DB dalam klaster Aurora. Titik akhir `small-instances` selalu terhubung ke instans DB `db.r4.4xlarge`, yang merupakan host bernomor lebih rendah dalam klaster ini.

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

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

```
| @@aurora_server_id      |
+-------------------------+
| custom-endpoint-demo-07 |
+-------------------------+

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

Titik akhir `big-instances` selalu terhubung ke instans DB `db.r4.16xlarge`, yang merupakan dua host dengan nomor tertinggi dalam klaster ini.

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

$ mysql -h big-instances.cluster-custom-c7tj4example.ca-central-1.rds.amazonaws.com -u
 $MYUSER -p
mysql> select @@aurora_server_id;
+-------------------------+
| @@aurora_server_id      |
+-------------------------+
| custom-endpoint-demo-09 |
+-------------------------+
```

Menggunakan titik akhir instans

Setiap instans DB dalam klaster Aurora memiliki titik akhir instans bawaan, yang nama dan atribut lainnya dikelola oleh Aurora. Anda tidak dapat membuat, menghapus, atau memodifikasi titik akhir semacam ini. Anda mungkin sudah paham tentang titik akhir instans jika Anda menggunakan Amazon RDS. Namun, dengan Aurora, Anda biasanya lebih sering menggunakan titik akhir penulis dan pembaca dibandingkan titik akhir instans.

Dalam operasi day-to-day Aurora, cara utama Anda menggunakan titik akhir instans adalah dengan mendiagnosis masalah kapasitas atau kinerja yang memengaruhi satu instance tertentu dalam klaster Aurora. Saat terhubung dengan instans tertentu, Anda dapat memeriksa variabel status, metrik, dan sebagainya. Dengan melakukan hal tersebut, Anda dapat menentukan apa yang terjadi untuk instans itu yang berbeda dengan apa yang terjadi untuk instans lain dalam klaster.

Dalam kasus penggunaan tingkat lanjut, Anda mungkin mengonfigurasi beberapa instans DB secara berbeda dari yang lain. Dalam hal ini, gunakan titik akhir instans untuk terhubung langsung ke instans yang lebih kecil, lebih besar, atau yang memiliki karakteristik yang berbeda dari yang lain. Selain itu, siapkan prioritas failover agar instans DB khusus ini menjadi pilihan terakhir untuk mengambil alih sebagai instans primer. Kami menyarankan Anda menggunakan titik akhir kustom dan bukan titik akhir instans dalam kasus tersebut. Tindakan tersebut akan menyederhanakan pengelolaan koneksi dan ketersediaan yang tinggi saat Anda menambahkan lebih banyak instans DB ke klaster Anda.

Bagaimana titik akhir Aurora berfungsi dengan ketersediaan tinggi

Untuk klaster yang mementingkan ketersediaan tinggi, gunakan titik akhir penulis untuk koneksi baca/tulis atau koneksi tujuan umum dan titik akhir pembaca untuk koneksi hanya baca. Titik akhir penulis dan pembaca mengelola failover instans DB secara lebih baik daripada titik akhir instans. Tidak seperti titik akhir instans, titik akhir penulis dan pembaca secara otomatis mengubah instans DB untuk koneksi jika instans DB di klaster Anda menjadi tidak tersedia.

Jika instans DB primer dari klaster DB gagal, Aurora secara otomatis melakukan failover ke instans DB primer yang baru. Hal tersebut dilakukan dengan mempromosikan Replika Aurora yang sudah ada ke instans DB primer yang baru, atau membuat instans DB primer yang baru. Jika terjadi failover, Anda dapat menggunakan titik akhir penulis untuk melakukan koneksi kembali ke instans DB primer yang baru dipromosikan atau dibuat, atau menggunakan titik akhir pembaca untuk melakukan koneksi kembali ke salah satu Replika Aurora di klaster DB. Selama failover, titik akhir pembaca mungkin akan mengarahkan koneksi ke instans DB primer yang baru dari klaster DB untuk waktu yang singkat setelah Replika Aurora dipromosikan menjadi instans DB primer yang baru.

Jika Anda merancang logika aplikasi Anda sendiri untuk mengelola koneksi untuk titik akhir instans, Anda dapat secara manual atau programatis menemukan serangkaian instans DB yang tersedia dalam klaster DB. Gunakan describe-db-clustersAWS CLIperintah atau operasi DescribedBClusters RDS API untuk menemukan cluster DB dan endpoint pembaca, instans DB, apakah instans DB adalah pembaca, dan tingkatan promosinya. Anda kemudian dapat mengonfirmasi kelas instansnya setelah failover, dan melakukan koneksi ke titik akhir instans yang sesuai.

Untuk informasi selengkapnya tentang failover, lihat Toleransi kesalahan untuk klaster DB Aurora.

Kelas instans DB Aurora

Kelas instans DB menentukan komputasi dan kapasitas memori instans DB Amazon Aurora. Kelas instans DB yang Anda butuhkan tergantung pada kebutuhan daya dan memori pemrosesan Anda.

Sebuah kelas instans DB terdiri dari jenis dan ukuran kelas instans DB. Misalnya, db.r6g adalah tipe kelas instans DB yang dioptimalkan memori yang didukung oleh prosesor Graviton2. AWS Dalam jenis kelas instans db.r6g, db.r6g.2xlarge adalah kelas instans DB. Ukuran kelas ini adalah 2xlarge.

Untuk informasi selengkapnya tentang harga kelas instans, lihat Harga Amazon RDS.

Topik

- Jenis kelas instans DB
- Mesin DB yang didukung untuk kelas instans DB
- Menentukan dukungan kelas instans DB di Wilayah AWS
- Spesifikasi perangkat keras kelas instans DB untuk Aurora

Jenis kelas instans DB

Amazon Aurora mendukung kelas instans DB untuk kasus penggunaan berikut:

- Aurora Serverless v2
- Memori yang dioptimalkan
- Performa yang dapat melonjak
- Optimized Reads

Untuk informasi selengkapnya tentang jenis instans Amazon EC2, lihat Jenis instans di dokumentasi Amazon EC2.

Jenis kelas instans Aurora Serverless v2

Jenis Aurora Serverless v2 berikut tersedia:

- db.serverless – Jenis kelas instans DB khusus yang digunakan oleh Aurora Serverless v2. Aurora menyesuaikan komputasi, memori, dan sumber daya jaringan secara dinamis saat beban kerja berubah. Untuk detail penggunaan, lihat Menggunakan Aurora Serverless v2.

Jenis kelas instans memori yang dioptimalkan

Rangkaian X dengan memori yang dioptimalkan mendukung kelas instans berikut:

- db.x2g - Kelas instans yang dioptimalkan untuk aplikasi intensif memori dan didukung oleh prosesor Graviton2. AWS Kelas instans ini menawarkan biaya rendah per GiB memori.

  Anda dapat memodifikasi instans DB untuk menggunakan salah satu kelas instans DB yang didukung oleh prosesor AWS Graviton2. Untuk melakukannya, selesaikan langkah yang sama seperti modifikasi instans DB lainnya.

Rangkaian R dengan memori yang dioptimalkan mendukung jenis kelas instans berikut:

- db.r7g - Kelas instans yang didukung oleh prosesor Graviton3. AWS Kelas instans ini ideal untuk menjalankan beban kerja yang memerlukan banyak memori .

  Anda dapat memodifikasi instans DB untuk menggunakan salah satu kelas instans DB yang didukung oleh prosesor AWS Graviton3. Untuk melakukannya, selesaikan langkah yang sama seperti modifikasi instans DB lainnya.
- db.r6g - Kelas instans yang didukung oleh prosesor Graviton2. AWS Kelas instans ini ideal untuk menjalankan beban kerja yang memerlukan banyak memori .

  Anda dapat memodifikasi instans DB untuk menggunakan salah satu kelas instans DB yang didukung oleh prosesor AWS Graviton2. Untuk melakukannya, selesaikan langkah yang sama seperti modifikasi instans DB lainnya.
- db.r6i – Kelas instans yang didukung oleh prosesor Intel Xeon Scalable Generasi ke-3. Kelas instans ini bersertifikat SAP dan ideal untuk beban kerja yang memerlukan banyak memori dalam basis data sumber terbuka, seperti MySQL dan PostgreSQL.
- db.r4 – Kelas instans ini hanya didukung untuk versi Aurora PostgreSQL 11 dan 12. Untuk semua klaster DB Aurora PostgreSQL yang menggunakan kelas instans db.r4 DB, sebaiknya Anda meningkatkan ke kelas instans generasi yang lebih tinggi sesegera mungkin.

  Kelas instans db.r4 tidak tersedia untuk konfigurasi penyimpanan klaster Aurora I/O-Optimized.
- db.r3 – Kelas instans yang menyediakan optimasi memori.

  Amazon Aurora telah memulai end-of-life proses untuk kelas instans db.r3 DB menggunakan jadwal berikut, yang mencakup rekomendasi peningkatan. Untuk semua klaster DB Aurora MySQL

yang menggunakan kelas instans DB db.r3, sebaiknya Anda meningkatkan ke kelas instans DB db.r5 atau yang lebih tinggi sesegera mungkin.

Tindakan atau rekomendasi	Tanggal
Anda tidak dapat lagi membuat klaster DB Aurora MySQL yang menggunakan kelas instans DB db.r3.	Sekarang
Amazon Aurora memulai peningkatan otomatis untuk klaster DB Aurora MySQL yang menggunakan kelas instans DB db.r3 ke kelas instans DB db.r5 yang setara atau lebih tinggi.	31 Januari 2023

Jenis kelas instans performa yang dapat melonjak

Jenis kelas instans DB performa yang dapat melonjak berikut tersedia:

- db.t4g - Kelas instance tujuan umum yang didukung oleh prosesor Graviton2 berbasis ARM. AWS Kelas instans ini memberikan performa harga yang lebih baik daripada kelas instans DB performa yang dapat melonjak sebelumnya untuk serangkaian beban kerja tujuan umum yang dapat melonjak. Instans Amazon RDS db.t4g dikonfigurasi untuk mode Tidak Terbatas. Artinya, instans tersebut dapat melampaui garis dasar dalam jangka waktu 24 jam dengan biaya tambahan.

  Anda dapat memodifikasi instans DB untuk menggunakan salah satu kelas instans DB yang didukung oleh prosesor AWS Graviton2. Untuk melakukannya, selesaikan langkah yang sama seperti modifikasi instans DB lainnya.
- db.t3 – Kelas instans yang memberikan tingkat performa dasar, dengan kemampuan untuk melonjak hingga penggunaan CPU penuh. Instans db.t3 dikonfigurasi untuk mode Tidak Terbatas. Kelas instans ini memberikan kapasitas komputasi yang lebih besar dibandingkan kelas instans db.t2 sebelumnya. Produk ini didukung oleh Nitro System AWS , kombinasi perangkat keras khusus dan hypervisor ringan. Sebaiknya gunakan kelas instans ini hanya untuk server pengujian dan pengembangan, atau server non-produksi lainnya.
- db.t2 – Kelas instans yang memberikan tingkat performa dasar, dengan kemampuan untuk melonjak hingga penggunaan CPU penuh. Instans db.t2 dikonfigurasi untuk mode Tidak Terbatas. Sebaiknya gunakan kelas instans ini hanya untuk server pengujian dan pengembangan, atau server non-produksi lainnya.

Kelas instans db.t2 tidak tersedia untuk konfigurasi penyimpanan klaster Aurora I/O-Optimized.

> **Note**
>
> Sebaiknya gunakan kelas instans DB T hanya untuk server pengembangan, pengujian, atau non-produksi lainnya. Rekomendasi lebih mendetail untuk kelas instans T dapat dilihat di Menggunakan kelas instans T untuk pengembangan dan pengujian.

Untuk spesifikasi perangkat keras kelas instans DB, lihat Spesifikasi perangkat keras kelas instans DB untuk Aurora.

Jenis kelas instans Optimized Reads

Jenis kelas instans Optimized Reads berikut tersedia:

- db.r6gd - Kelas instans yang didukung oleh prosesor Graviton2. AWS Kelas instans ini ideal untuk menjalankan beban kerja intensif memori dan menawarkan penyimpanan tingkat blok SSD berbasis NVME lokal untuk aplikasi yang membutuhkan penyimpanan lokal berkecepatan tinggi dan latensi rendah.
- db.r6id – Kelas instans yang didukung prosesor Intel Xeon Scalable Generasi ke-3. Kelas instans ini bersertifikat SAP dan ideal untuk beban kerja yang menggunakan banyak memori. Kelas instans tersebut menawarkan memori maksimum 1 TiB dan hingga 7,6 TB penyimpanan SSD berbasis NVMe default.

Mesin DB yang didukung untuk kelas instans DB

Di tabel berikut, Anda dapat menemukan detail tentang kelas instans Amazon Aurora DB yang didukung untuk mesin DB Aurora.

Kelas instans	Aurora MySQL	Aurora PostgreSQL
db.serverless – Aurora Serverless v2 kelas instans dengan penskalaan kapasitas otomatis		
db.serverless	Lihat Aurora Serverless v2	Lihat Aurora Serverless v2

Kelas instans	Aurora MySQL	Aurora PostgreSQL
db.x2g — kelas instance yang dioptimalkan untuk memori yang didukung oleh prosesor Graviton2 AWS		
db.x2g.16xlarge	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi
db.x2g.12xlarge	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi
db.x2g.8xlarge	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi
db.x2g.4xlarge	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi
db.x2g.2xlarge	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi
db.x2g.xlarge	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi

Kelas instans	Aurora MySQL	Aurora PostgreSQL
db.x2g.large	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi
db.r6gd - Kelas instance Reads yang Dioptimalkan yang didukung oleh prosesor Graviton2 AWS		
db.r6gd.16xlarge	Tidak	15.4 dan yang lebih tinggi, 14.9 dan yang lebih tinggi
db.r6gd.12xlarge	Tidak	15.4 dan yang lebih tinggi, 14.9 dan yang lebih tinggi
db.r6gd.8xlarge	Tidak	15.4 dan yang lebih tinggi, 14.9 dan yang lebih tinggi
db.r6gd.4xlarge	Tidak	15.4 dan yang lebih tinggi, 14.9 dan yang lebih tinggi
db.r6gd.2xlarge	Tidak	15.4 dan yang lebih tinggi, 14.9 dan yang lebih tinggi
db.r6gd.xlarge	Tidak	15.4 dan yang lebih tinggi, 14.9 dan yang lebih tinggi
db.r6id – Kelas instans Optimized Reads		
db.r6id.32xlarge	Tidak	15.4 dan yang lebih tinggi, 14.9 dan yang lebih tinggi
db.r6id.24xlarge	Tidak	15.4 dan yang lebih tinggi, 14.9 dan yang lebih tinggi
db.r7g — kelas instance yang dioptimalkan untuk memori yang didukung oleh prosesor Graviton3 AWS		

Kelas instans	Aurora MySQL	Aurora PostgreSQL
db.r7g.16xlarge	2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.7 dan yang lebih tinggi, 13.10 dan yang lebih tinggi
db.r7g.12xlarge	2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.7 dan yang lebih tinggi, 13.10 dan yang lebih tinggi
db.r7g.8xlarge	2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.7 dan yang lebih tinggi, 13.10 dan yang lebih tinggi
db.r7g.4xlarge	2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.7 dan yang lebih tinggi, 13.10 dan yang lebih tinggi
db.r7g.2xlarge	2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.7 dan yang lebih tinggi, 13.10 dan yang lebih tinggi
db.r7g.xlarge	2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.7 dan yang lebih tinggi, 13.10 dan yang lebih tinggi
db.r7g.large	2.12.0 dan yang lebih tinggi, 3.03.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.7 dan yang lebih tinggi, 13.10 dan yang lebih tinggi
db.r6g — kelas instance yang dioptimalkan untuk memori yang didukung oleh prosesor Graviton2 AWS		
db.r6g.16xlarge	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi

Kelas instans	Aurora MySQL	Aurora PostgreSQL
db.r6g.12xlarge	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi
db.r6g.8xlarge	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi
db.r6g.4xlarge	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi
db.r6g.2xlarge	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi
db.r6g.xlarge	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi
db.r6g.large	2.09.2 dan yang lebih tinggi, 2.10.0, dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.8 dan yang lebih tinggi, 11.9, 11.12 dan yang lebih tinggi
db.r6i – kelas instans dengan memori yang dioptimalkan		
db.r6i.32xlarge	2.11.0 dan yang lebih tinggi, 3.02.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.5 dan yang lebih tinggi, 12.9 dan yang lebih tinggi

Kelas instans	Aurora MySQL	Aurora PostgreSQL
db.r6i.24xlarge	2.11.0 dan yang lebih tinggi, 3.02.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.5 dan yang lebih tinggi, 12.9 dan yang lebih tinggi
db.r6i.16xlarge	2.11.0 dan yang lebih tinggi, 3.02.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.5 dan yang lebih tinggi, 12.9 dan yang lebih tinggi
db.r6i.12xlarge	2.11.0 dan yang lebih tinggi, 3.02.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.5 dan yang lebih tinggi, 12.9 dan yang lebih tinggi
db.r6i.8xlarge	2.11.0 dan yang lebih tinggi, 3.02.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.5 dan yang lebih tinggi, 12.9 dan yang lebih tinggi
db.r6i.4xlarge	2.11.0 dan yang lebih tinggi, 3.02.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.5 dan yang lebih tinggi, 12.9 dan yang lebih tinggi
db.r6i.2xlarge	2.11.0 dan yang lebih tinggi, 3.02.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.5 dan yang lebih tinggi, 12.9 dan yang lebih tinggi
db.r6i.xlarge	2.11.0 dan yang lebih tinggi, 3.02.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.5 dan yang lebih tinggi, 12.9 dan yang lebih tinggi
db.r6i.large	2.11.0 dan yang lebih tinggi, 3.02.1 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.5 dan yang lebih tinggi, 12.9 dan yang lebih tinggi
db.r5 – kelas instans dengan memori yang dioptimalkan		
db.r5.24xlarge	Semua versi 2.x; 3.01.0 dan yang lebih tinggi	Semua versi yang tersedia saat ini

Kelas instans	Aurora MySQL	Aurora PostgreSQL
db.r5.16xlarge	Semua versi 2.x; 3.01.0 dan yang lebih tinggi	Semua versi yang tersedia saat ini
db.r5.12xlarge	Semua versi 2.x; 3.01.0 dan yang lebih tinggi	Semua versi yang tersedia saat ini
db.r5.8xlarge	Semua versi 2.x; 3.01.0 dan yang lebih tinggi	Semua versi yang tersedia saat ini
db.r5.4xlarge	Semua versi 2.x; 3.01.0 dan yang lebih tinggi	Semua versi yang tersedia saat ini
db.r5.2xlarge	Semua versi 2.x; 3.01.0 dan yang lebih tinggi	Semua versi yang tersedia saat ini
db.r5.xlarge	Semua versi 2.x; 3.01.0 dan yang lebih tinggi	Semua versi yang tersedia saat ini
db.r5.large	Semua versi 2.x; 3.01.0 dan yang lebih tinggi	Semua versi yang tersedia saat ini
db.r4 – kelas instans dengan memori yang dioptimalkan		
db.r4.16xlarge	Semua versi 2.x; tidak didukung di 3.01.0 dan yang lebih tinggi	Tidak
db.r4.8xlarge	Semua versi 2.x; tidak didukung di 3.01.0 dan yang lebih tinggi	Tidak
db.r4.4xlarge	Semua versi 2.x; tidak didukung di 3.01.0 dan yang lebih tinggi	Tidak
db.r4.2xlarge	Semua versi 2.x; tidak didukung di 3.01.0 dan yang lebih tinggi	Tidak
db.r4.xlarge	Semua versi 2.x; tidak didukung di 3.01.0 dan yang lebih tinggi	Tidak

Kelas instans	Aurora MySQL	Aurora PostgreSQL
db.r4.large	Semua versi 2.x; tidak didukung di 3.01.0 dan yang lebih tinggi	Tidak
db.t4g — kelas instance berkinerja pecah yang ditenagai oleh prosesor Graviton2 AWS		
db.t4g.2xlarge	Tidak	Tidak
db.t4g.xlarge	Tidak	Tidak
db.t4g.large	2.11.1 dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.7 dan yang lebih tinggi, 11.12 dan yang lebih tinggi
db.t4g.medium	2.11.1 dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.7 dan yang lebih tinggi, 11.12 dan yang lebih tinggi
db.t4g.small	Tidak	Tidak
db.t3 – kelas instans performa yang dapat melonjak		
db.t3.2xlarge	Tidak	Tidak
db.t3.xlarge	Tidak	Tidak
db.t3.large	2.11.1 dan yang lebih tinggi, 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.7 dan yang lebih tinggi, 11.12 dan yang lebih tinggi
db.t3.medium	Semua versi 2.x; 3.01.0 dan yang lebih tinggi	15.2 dan yang lebih tinggi, 14.3 dan yang lebih tinggi, 13.3 dan yang lebih tinggi, 12.7 dan yang lebih tinggi, 11.12 dan yang lebih tinggi

Kelas instans	Aurora MySQL	Aurora PostgreSQL
db.t3.small	Semua versi 2.x; tidak didukung di 3.01.0 dan yang lebih tinggi	Tidak
db.t3.micro	Tidak	Tidak
db.t2 – kelas instans performa yang dapat melonjak		
db.t2.medium	Semua versi 2.x; tidak didukung di 3.01.0 dan yang lebih tinggi	Tidak
db.t2.small	Semua versi 2.x; tidak didukung di 3.01.0 dan yang lebih tinggi	Tidak

Menentukan dukungan kelas instans DB di Wilayah AWS

Untuk menentukan kelas instans DB yang didukung oleh mesin DB di Wilayah AWS tertentu, Anda dapat menggunakan beberapa pendekatan. Anda dapat menggunakan AWS Management Console, halaman Harga Amazon RDS, atau perintah describe-orderable-db-instance-options AWS CLI .

> **Note**
>
> Ketika Anda melakukan operasi dengan AWS Management Console, secara otomatis menampilkan kelas instans DB yang didukung untuk mesin DB tertentu, versi mesin DB, dan Wilayah AWS. Contoh operasi yang dapat Anda lakukan termasuk membuat dan mengubah instans DB.

Daftar Isi

- Menggunakan halaman harga Amazon RDS untuk menentukan dukungan kelas instans DB di Wilayah AWS
- Menggunakan AWS CLI untuk menentukan dukungan kelas instans DB di Wilayah AWS
  - Menyusun daftar kelas instans DB yang didukung oleh versi mesin DB tertentu di Wilayah AWS
  - Menyusun daftar versi mesin DB yang mendukung kelas instans DB tertentu di Wilayah AWS

Menggunakan halaman harga Amazon RDS untuk menentukan dukungan kelas instans DB di Wilayah AWS

Anda dapat menggunakan halaman Harga Amazon Aurora untuk menentukan kelas instans DB yang didukung oleh masing-masing mesin DB di Wilayah AWS tertentu.

Untuk menggunakan halaman harga guna menentukan kelas instans DB yang didukung oleh masing-masing mesin DB di sebuah Wilayah

1. Buka Harga Amazon Aurora.
2. Pilih mesin Amazon Aurora di bagian Kalkulator Harga AWS .
3. Di Pilih Wilayah, pilih Wilayah AWS.
4. Di Opsi Konfigurasi Klaster, pilih opsi konfigurasi.
5. Gunakan bagian instans yang kompatibel untuk melihat kelas instans DB yang didukung.
6. (Opsional) Pilih opsi lain di kalkulator, lalu pilih Simpan dan lihat ringkasan atau Simpan dan tambahkan layanan.

Menggunakan AWS CLI untuk menentukan dukungan kelas instans DB di Wilayah AWS

Anda dapat menggunakan AWS CLI untuk menentukan kelas instans DB mana yang didukung untuk mesin DB tertentu dan versi mesin DB dalam file Wilayah AWS.

Untuk menggunakan AWS CLI contoh berikut, masukkan nilai yang valid untuk mesin DB, versi mesin DB, kelas instans DB, dan Wilayah AWS. Tabel berikut menunjukkan nilai-nilai mesin DB yang valid.

Nama mesin	Nilai mesin di perintah CLI	Informasi selengkapnya tentang versi
Kompatibel dengan MySQL 5.7 dan Aurora yang kompatibel dengan 8.0	`aurora-mysql`	Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 2 dan Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 3 di Catatan Rilis Aurora MySQL

Nama mesin	Nilai mesin di perintah CLI	Informasi selengkapnya tentang versi
Aurora PostgreSQL	`aurora-postgresql`	Catatan Rilis untuk Aurora PostgreSQL

Untuk informasi tentang Wilayah AWS nama, lihatAWS Daerah.

Contoh berikut menunjukkan bagaimana menentukan dukungan kelas instance DB dalam Wilayah AWS menggunakan AWS CLI perintah describe-orderable-db-instance-options.

Topik

- Menyusun daftar kelas instans DB yang didukung oleh versi mesin DB tertentu di Wilayah AWS
- Menyusun daftar versi mesin DB yang mendukung kelas instans DB tertentu di Wilayah AWS

Menyusun daftar kelas instans DB yang didukung oleh versi mesin DB tertentu di Wilayah AWS

Untuk daftar kelas instans DB yang didukung oleh versi mesin DB tertentu dalam Wilayah AWS, jalankan perintah berikut.

Untuk Linux, macOS, atau Unix:

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

Untuk Windows:

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

Output juga menunjukkan mode mesin yang didukung untuk setiap kelas instans DB.

Sebagai contoh, perintah berikut menyusun daftar kelas instans DB yang didukung untuk versi 13.6 mesin DB Aurora PostgreSQL di AS Timur (Virginia Utara).

Untuk Linux, macOS, atau Unix:

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

Untuk Windows:

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

Menyusun daftar versi mesin DB yang mendukung kelas instans DB tertentu di Wilayah AWS

Untuk menyusun daftar versi mesin DB yang mendukung kelas instans DB tertentu di Wilayah AWS, jalankan perintah berikut.

Untuk Linux, macOS, atau Unix:

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

Untuk Windows:

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

```
    --region region
```

Output juga menunjukkan mode mesin yang didukung untuk setiap versi mesin DB.

Sebagai contoh, perintah berikut menyusun daftar versi mesin DB Aurora PostgreSQL yang mendukung kelas instans DB db.r5.large di AS Timur (Virginia Utara).

Untuk Linux, macOS, atau Unix:

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

Untuk Windows:

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

Spesifikasi perangkat keras kelas instans DB untuk Aurora

Terminologi berikut digunakan untuk menjelaskan spesifikasi perangkat keras untuk kelas instans DB:

vCPU

Jumlah unit pemrosesan pusat (CPU) virtual. CPU virtual adalah unit kapasitas yang dapat Anda gunakan untuk membandingkan kelas instans DB. Alih-alih membeli atau menyewa prosesor tertentu untuk digunakan selama beberapa bulan atau tahun, Anda menyewa kapasitas per jam. Tujuan kami adalah menyediakan jumlah kapasitas CPU yang konsisten dan spesifik, dalam batas perangkat keras sebenarnya yang mendasarinya.

ECU

Ukuran relatif daya pemrosesan integer dari instans Amazon EC2. Agar mempermudah developer membandingkan kapasitas CPU antara berbagai kelas instans, kami telah mendefinisikan Unit

Komputasi Amazon EC2. Jumlah CPU yang dialokasikan ke instans tertentu dinyatakan dalam Unit Komputasi EC2 ini. Satu ECU saat ini menyediakan kapasitas CPU yang setara dengan prosesor 1.0–1.2 GHz 2007 Opteron atau 2007 Xeon.

Memori (GiB)

RAM, dalam gibibyte, dialokasikan ke instans DB. Sering kali ada rasio yang konsisten antara memori dan vCPU. Sebagai contoh, ambil kelas instans db.r4, yang memiliki rasio memori terhadap vCPU serupa dengan kelas instans db.r5. Namun, untuk sebagian besar kasus penggunaan, kelas instans db.r5 memberikan performa yang lebih baik dan lebih konsisten dibandingkan kelas instans db.r4.

Maks. Bandwidth EBS (Mbps)

Bandwidth EBS maksimum dalam megabit per detik. Bagi dengan 8 untuk mendapatkan throughput yang diharapkan dalam megabyte per detik.

> **Note**
>
> Angka ini mengacu pada bandwidth I/O untuk penyimpanan lokal dalam instans DB. Ini tidak berlaku untuk komunikasi dengan volume klaster Aurora.

Bandwith jaringan

Kecepatan jaringan relatif terhadap kelas instans DB lainnya.

Pada tabel berikut, Anda dapat menemukan detail perangkat keras tentang kelas instans DB Amazon RDS untuk Aurora.

Untuk informasi tentang dukungan mesin DB Aurora untuk setiap kelas instans DB, lihat Mesin DB yang didukung untuk kelas instans DB.

Kelas instans	vCPU	ECU	Memori (GiB)	Bandwidth maksimum (mbps) penyimpanan lokal	Performa jaringan (Gbps)
db.x2g – kelas instans dengan memori yang dioptimalkan					

Kelas instans	vCPU	ECU	Memori (GiB)	Bandwidth maksimum (mbps) penyimpanan lokal	Performa jaringan (Gbps)
db.x2g.16xlarge	64	—	1024	19.000	25
db.x2g.12xlarge	48	—	768	14,250	20
db.x2g.8xlarge	32	—	512	9.500	12
db.x2g.4xlarge	16	—	256	4,750	Hingga 10
db.x2g.2xlarge	8	—	128	Hingga 4.750	Hingga 10
db.x2g.xlarge	4	—	64	Hingga 4.750	Hingga 10
db.x2g.large	2	—	32	Hingga 4.750	Hingga 10
db.r7g – kelas instans dengan memori yang dioptimalkan yang didukung oleh prosesor AWS Graviton3					
db.r7g.16xlarge	64	—	512	20.000	30
db.r7g.12xlarge	48	—	384	15.000	22.5
db.r7g.8xlarge	32	—	256	10.000	15
db.r7g.4xlarge	16	—	128	Hingga 10.000	Hingga 15
db.r7g.2xlarge	8	—	64	Hingga 10.000	Hingga 15
db.r7g.xlarge	4	—	32	Hingga 10.000	Hingga 12,5
db.r7g.large	2	—	16	Hingga 10.000	Hingga 12,5

Kelas instans	vCPU	ECU	Memori (GiB)	Bandwidth maksimum (mbps) penyimpanan lokal	Performa jaringan (Gbps)
db.r6g – kelas instans dengan memori yang dioptimalkan yang didukung oleh prosesor AWS Graviton2					
db.r6g.16xlarge	64	—	512	19.000	25
db.r6g.12xlarge	48	—	384	13.500	20
db.r6g.8xlarge	32	—	256	9.000	12
db.r6g.4xlarge	16	—	128	4,750	Hingga 10
db.r6g.2xlarge	8	—	64	Hingga 4.750	Hingga 10
db.r6g.xlarge	4	—	32	Hingga 4.750	Hingga 10
db.r6g.large	2	—	16	Hingga 4.750	Hingga 10
db.r6i – kelas instans dengan memori yang dioptimalkan					
db.r6i.32xlarge	128	—	1,024	40.000	50
db.r6i.24xlarge	96	—	768	30.000	37,5
db.r6i.16xlarge	64	—	512	20.000	25
db.r6i.12xlarge	48	—	384	15.000	18.75
db.r6i.8xlarge	32	—	256	10.000	12,5
db.r6i.4xlarge	16	—	128	Hingga 10.000	Hingga 12,5
db.r6i.2xlarge	8	—	64	Hingga 10.000	Hingga 12,5

Kelas instans	vCPU	ECU	Memori (GiB)	Bandwidth maksimum (mbps) penyimpanan lokal	Performa jaringan (Gbps)
db.r6i.xlarge	4	—	32	Hingga 10.000	Hingga 12,5
db.r6i.large	2	—	16	Hingga 10.000	Hingga 12,5
db.r5 – kelas instans dengan memori yang dioptimalkan					
db.r5.24xlarge	96	347	768	19.000	25
db.r5.16xlarge	64	264	512	13.600	20
db.r5.12xlarge	48	173	384	9.500	12
db.r5.8xlarge	32	132	256	6,800	10
db.r5.4xlarge	16	71	128	4,750	Hingga 10
db.r5.2xlarge	8	38	64	Hingga 4.750	Hingga 10
db.r5.xlarge	4	19	32	Hingga 4.750	Hingga 10
db.r5.large	2	10	16	Hingga 4.750	Hingga 10
db.r4 – kelas instans dengan memori yang dioptimalkan					
db.r4.16xlarge	64	195	488	14.000	25
db.r4.8xlarge	32	99	244	7.000	10
db.r4.4xlarge	16	53	122	3.500	Hingga 10
db.r4.2xlarge	8	27	61	1.700	Hingga 10
db.r4.xlarge	4	13,5	30,5	850	Hingga 10

Kelas instans	vCPU	ECU	Memori (GiB)	Bandwidth maksimum (mbps) penyimpanan lokal	Performa jaringan (Gbps)
db.r4.large	2	7	15.25	425	Hingga 10
db.t4g – kelas instans performa yang dapat melonjak					
db.t4g.large	2	—	8	Hingga 2.780	Hingga 5
db.t4g.medium	2	—	4	Hingga 2.085	Hingga 5
db.t3 – kelas instans performa yang dapat melonjak					
db.t3.large	2	Variabel	8	Hingga 2.048	Hingga 5
db.t3.medium	2	Variabel	4	Hingga 1.536	Hingga 5
db.t3.small	2	Variabel	2	Hingga 1.536	Hingga 5
db.t2 – kelas instans performa yang dapat melonjak					
db.t2.medium	2	Variabel	4	—	Sedang
db.t2.small	1	Variabel	2	—	Rendah

Penyimpanan dan keandalan Amazon Aurora

Di bagian berikut ini, Anda dapat mempelajari tentang subsistem penyimpanan Aurora. Aurora menggunakan arsitektur penyimpanan terdistribusi dan bersama yang merupakan faktor penting dalam performa, skalabilitas, dan keandalan klaster Aurora.

Topik

- Gambaran umum penyimpanan Amazon Aurora
- Apa saja konten volume klaster
- Konfigurasi penyimpanan untuk klaster DB Amazon Aurora

- Cara penyimpanan Aurora berubah ukuran secara otomatis
- Cara penyimpanan data Aurora ditagih
- Keandalan Amazon Aurora

Gambaran umum penyimpanan Amazon Aurora

Data Aurora disimpan di volume klaster, yang merupakan volume virtual tunggal yang menggunakan solid state drive (SSD). Volume klaster terdiri dari salinan data di tiga Zona Ketersediaan dalam satu Wilayah AWS. Karena data direplikasi secara otomatis di Zona ketersediaan, data Anda sangat durabel dengan kemungkinan kehilangan data yang kecil. Replikasi ini juga mempertahankan ketersediaan basis data Anda selama failover. Hal itu terjadi karena salinan data sudah ada di Zona Ketersediaan lain dan terus melayani permintaan data ke instans DB dalam klaster DB Anda. Jumlah replikasi bergantung pada jumlah instans DB dalam klaster Anda.

Aurora menggunakan penyimpanan lokal terpisah untuk file sementara yang tidak persisten. Ini termasuk file yang digunakan untuk tujuan seperti mengurutkan set data besar selama pemrosesan kueri, dan membuat indeks. Lihat informasi yang lebih lengkap di Batas penyimpanan sementara untuk Aurora MySQL dan Batas penyimpanan sementara untuk Aurora PostgreSQL.

Apa saja konten volume klaster

Volume klaster Aurora berisi semua data pengguna, objek skema, dan metadata internal seperti tabel sistem dan log biner. Misalnya, Aurora menyimpan semua tabel, indeks, objek besar biner (BLOB), prosedur tersimpan, dan seterusnya untuk klaster Aurora dalam volume klaster.

Arsitektur penyimpanan bersama Aurora membuat data Anda terpisah dari instans DB di dalam klaster. Misalnya, Anda dapat menambahkan instans DB dengan cepat karena Aurora tidak membuat salinan data tabel yang baru. Sebagai gantinya, instans DB terhubung ke volume bersama yang sudah berisi semua data Anda. Anda dapat menghapus instans DB dari suatu klaster tanpa menghapus data dasar dari klaster tersebut. Aurora hanya akan menghapus data jika Anda menghapus seluruh klaster tersebut.

Konfigurasi penyimpanan untuk klaster DB Amazon Aurora

Amazon Aurora memiliki dua konfigurasi penyimpanan klaster DB:

- Aurora I/O-Optimized – Peningkatan performa harga dan prediktabilitas untuk aplikasi intensif I/O. Anda hanya membayar penggunaan dan penyimpanan klaster DB Anda, tanpa biaya tambahan untuk operasi I/O baca dan tulis.

  Aurora I/O-Optimized adalah pilihan terbaik ketika pengeluaran I/O Anda adalah 25% atau lebih dari total pengeluaran basis data Aurora Anda.

  Anda dapat memilih Aurora I/O-Optimized saat Anda membuat atau memodifikasi klaster DB dengan versi mesin DB yang mendukung konfigurasi klaster Aurora I/O-Optimized. Anda dapat beralih dari Aurora I/O-Optimized ke Aurora Standard kapan saja.
- Aurora Standard – Harga hemat biaya untuk banyak aplikasi dengan penggunaan I/O sedang. Selain penggunaan dan penyimpanan klaster DB, Anda juga membayar tarif standar per 1 juta permintaan untuk operasi I/O.

  Aurora Standard adalah pilihan terbaik ketika pengeluaran I/O Anda kurang dari 25% dari total pengeluaran basis data Aurora Anda.

  Anda dapat beralih dari Aurora Standard ke Aurora I/O-Optimized sekali setiap 30 hari. Tidak ada downtime saat Anda beralih dari Aurora Standard keAurora I/O-Optimized, atau dari Aurora I/O-Optimized keAurora Standard.

Untuk informasi tentang Wilayah AWS dan dukungan versi, lihat Konfigurasi penyimpanan klaster Aurora.

Untuk informasi selengkapnya tentang harga konfigurasi penyimpanan Amazon Aurora, lihat Harga Amazon Aurora.

Untuk informasi tentang memilih konfigurasi penyimpanan saat membuat klaster DB, lihat Membuat klaster DB. Untuk informasi tentang memodifikasi konfigurasi penyimpanan untuk klaster DB, lihat Pengaturan untuk Amazon Aurora.

Cara penyimpanan Aurora berubah ukuran secara otomatis

Volume klaster Aurora secara otomatis bertambah seiring peningkatan jumlah data dalam basis data Anda. Ukuran maksimum untuk volume klaster Aurora adalah 128 tebibyte (TiB) atau 64 TiB, bergantung pada versi mesin DB. Untuk detail tentang ukuran maksimum untuk versi tertentu, lihat Batas ukuran Amazon Aurora. Penskalaan penyimpanan otomatis ini dikombinasikan dengan subsistem penyimpanan beperforma tinggi dan sangat terdistribusi. Hal ini menjadikan Aurora

sebagai pilihan yang baik untuk data perusahaan Anda yang penting ketika keandalan dan ketersediaan yang tinggi menjadi tujuan utama Anda.

Untuk menampilkan status volume, lihat Menampilkan status volume untuk klaster DB Aurora MySQL atau Menampilkan status volume untuk klaster DB Aurora PostgreSQL. Untuk cara menyeimbangkan biaya penyimpanan dengan prioritas lain, Penskalaan penyimpanan jelaskan cara memantau `AuroraVolumeBytesLeftTotal` metrik `VolumeBytesUsed` Amazon Aurora dan masuk. CloudWatch

Saat data Aurora dihapus, ruang yang dialokasikan untuk data tersebut dikosongkan. Contoh penghapusan data termasuk menghapus atau memotong tabel. Pengurangan otomatis penggunaan penyimpanan ini membantu Anda meminimalkan biaya penyimpanan.

> **Note**
>
> Perilaku pembatasan penyimpanan dan perubahan ukuran dinamis yang dibahas di sini berlaku untuk tabel persisten dan data lain yang disimpan dalam volume klaster.
> Untuk Aurora PostgreSQL, data tabel sementara disimpan dalam instans DB lokal.
> Untuk Aurora MySQL versi 2, data tabel sementara disimpan secara default dalam volume klaster untuk instans penulis dan dalam penyimpanan lokal untuk instans pembaca. Untuk informasi selengkapnya, lihat Mesin penyimpanan untuk tabel sementara di disk.
> Untuk Aurora MySQL versi 3, data tabel sementara disimpan dalam instans DB lokal atau dalam volume klaster. Untuk informasi selengkapnya, lihat Perilaku tabel sementara baru di Aurora MySQL versi 3.
> Ukuran maksimum tabel sementara yang berada di penyimpanan lokal dibatasi oleh ukuran penyimpanan lokal maksimum instans DB. Ukuran penyimpanan lokal bergantung pada kelas instans yang Anda gunakan. Lihat informasi yang lebih lengkap di Batas penyimpanan sementara untuk Aurora MySQL dan Batas penyimpanan sementara untuk Aurora PostgreSQL.

Beberapa fitur penyimpanan, seperti ukuran maksimum volume klaster dan pengubahan ukuran otomatis saat data dihapus, bergantung pada versi Aurora klaster Anda. Untuk informasi selengkapnya, lihat Penskalaan penyimpanan. Anda juga dapat mempelajari cara menghindari masalah penyimpanan dan cara memantau penyimpanan yang dialokasikan dan mengosongkan ruang dalam klaster Anda.

Cara penyimpanan data Aurora ditagih

Meskipun volume klaster Aurora dapat bertambah hingga 128 tebibyte (TiB), Anda hanya dikenai biaya untuk ruang yang Anda gunakan dalam volume klaster Aurora. Pada versi Aurora sebelumnya, volume klaster dapat menggunakan kembali ruang yang dikosongkan saat Anda menghapus data, tetapi ruang penyimpanan yang dialokasikan tidak akan pernah berkurang. Saat data Aurora dihapus, seperti menghapus tabel atau basis data, keseluruhan ruang yang dialokasikan berkurang sebesar jumlah yang sebanding. Dengan demikian, Anda dapat mengurangi biaya penyimpanan dengan menghapus tabel, indeks, basis data, dan sebagainya yang tidak lagi Anda butuhkan.

> **Tip**
>
> Untuk versi terdahulu tanpa fitur pengubahan ukuran dinamis, pengaturan ulang penggunaan penyimpanan untuk suatu klaster memerlukan dump logis dan pemulihan ke klaster baru. Operasi tersebut dapat memakan waktu lama untuk volume data yang besar. Jika Anda mengalami situasi ini, pertimbangkan untuk meng-upgrade klaster Anda ke versi yang mendukung pengubahan ukuran volume dinamis.

Untuk informasi tentang versi Aurora yang mendukung pengubahan ukuran dinamis, dan cara meminimalkan biaya penyimpanan dengan memantau penggunaan penyimpanan untuk klaster Anda, lihat Penskalaan penyimpanan. Untuk informasi tentang penagihan penyimpanan cadangan Aurora, lihat Memahami penggunaan penyimpanan cadangan Amazon Aurora. Untuk informasi harga tentang penyimpanan data Aurora, lihat Harga Amazon RDS for Aurora.

Keandalan Amazon Aurora

Aurora dirancang untuk menjadi andal, durabel, dan toleran terhadap kesalahan. Anda dapat menentukan arsitektur klaster DB Aurora Anda untuk meningkatkan ketersediaan dengan melakukan hal-hal seperti menambahkan Replika Aurora dan menempatkannya di Zona Ketersediaan yang berbeda-beda, dan Aurora juga menyertakan beberapa fitur otomatis yang menjadikannya solusi basis data yang andal.

Topik

- Perbaikan penyimpanan otomatis
- Cache halaman yang dapat bertahan
- Pemulihan dari pengaktifan ulang yang tidak direncanakan

Perbaikan penyimpanan otomatis

Karena Aurora menyimpan beberapa salinan data Anda di tiga Zona Ketersediaan, kemungkinan kehilangan data akibat kegagalan disk dapat diminimalkan secara signifikan. Aurora secara otomatis mendeteksi kegagalan dalam volume disk yang membentuk volume klaster. Saat sebuah segmen volume disk gagal, Aurora segera memperbaiki segmen tersebut. Saat Aurora memperbaiki segmen disk, Aurora menggunakan data dalam volume lain yang membentuk volume klaster untuk memastikan bahwa data dalam segmen yang diperbaiki sudah diperbarui. Akibatnya, Aurora menghindari kehilangan data dan mengurangi kebutuhan untuk melakukan point-in-time pemulihan untuk memulihkan dari kegagalan disk.

Cache halaman yang dapat bertahan

Di Aurora, cache halaman instans DB dikelola dalam proses terpisah dari basis data, yang memungkinkan cache halaman bertahan secara terpisah dari basis data. (Cache halaman juga disebut pool buffer InnoDB di Aurora MySQL dan cache buffer pada Aurora PostgreSQL).

Jika terjadi kegagalan basis data yang tidak terduga, cache halaman tetap berada di memori, sehingga halaman data saat ini tetap dalam kondisi “warm” di cache halaman saat basis data diaktifkan kembali. Hal ini memberikan peningkatan performa dengan meniadakan kebutuhan kueri awal untuk mengeksekusi operasi I/O untuk melakukan “warming” cache halaman.

Untuk Aurora MySQL, perilaku cache halaman saat boot ulang dan gagal adalah sebagai berikut:

- Versi yang lebih lama dari 2.10 – Ketika instans DB penulis melakukan boot ulang, cache halaman pada instans penulis bertahan, tetapi instans DB pembaca kehilangan cache halamannya.
- Versi 2.10 dan lebih tinggi – Anda dapat melakukan boot ulang instans penulis tanpa melakukan boot ulang instans pembaca.
  - Jika instans pembaca tidak melakukan boot ulang saat instans penulis melakukan boot ulang, instans tersebut tidak kehilangan cache halamannya.
  - Jika instans pembaca melakukan boot ulang saat instans penulis melakukan boot ulang, instans tersebut kehilangan cache halamannya.
- Ketika instans pembaca melakukan boot ulang, halaman cache pada instans penulis dan pembaca sama-sama bertahan.
- Ketika klaster DB gagal, efeknya mirip dengan ketika instans penulis melakukan boot ulang. Pada instans penulis baru (sebelumnya instans pembaca) cache halaman bertahan, tetapi pada instans pembaca (sebelumnya instans penulis), cache halaman tidak bertahan.

Untuk Aurora PostgreSQL, Anda dapat menggunakan manajemen cache klaster untuk mempertahankan cache halaman dari instans pembaca yang ditetapkan yang menjadi instans penulis setelah failover. Untuk informasi selengkapnya, lihat Pemulihan cepat setelah failover dengan manajemen cache klaster untuk Aurora PostgreSQL.

Pemulihan dari pengaktifan ulang yang tidak direncanakan

Aurora dirancang untuk pulih dari pengaktifan ulang yang tidak direncanakan hampir seketika dan terus melayani data aplikasi Anda tanpa log biner. Aurora pulih secara asinkron pada thread paralel, sehingga basis data Anda terbuka dan tersedia segera setelah pengaktifan ulang yang tidak direncanakan.

Lihat informasi yang lebih lengkap di Toleransi kesalahan untuk klaster DB Aurora dan Optimisasi untuk mengurangi waktu pengaktifan ulang basis data..

Berikut ini adalah pertimbangan untuk logging biner dan pemulihan pengaktifan ulang yang tidak direncanakan di Aurora MySQL:

- Mengaktifkan log biner di Aurora secara langsung memengaruhi waktu pemulihan setelah pengaktifan ulang yang tidak direncanakan, karena hal itu memaksa instans DB untuk melakukan pemulihan log biner.
- Jenis logging biner yang digunakan memengaruhi ukuran dan efisiensi logging. Untuk jumlah aktivitas basis data yang sama, beberapa format mencatat log untuk lebih banyak informasi dibandingkan yang lain dalam log biner. Pengaturan berikut untuk parameter `binlog_format` menghasilkan jumlah data log yang berbeda:
  - `ROW` – Data log paling banyak
  - `STATEMENT` – Data log paling sedikit
  - `MIXED` – Data log dalam jumlah sedang yang biasanya memberikan kombinasi integritas data dan performa yang terbaik

  Jumlah data log biner memengaruhi waktu pemulihan. Jika ada lebih banyak data yang dicatat lognya dalam log biner, instans DB harus memproses lebih banyak data selama pemulihan, yang meningkatkan waktu pemulihan.
- Untuk mengurangi overhead komputasi dan meningkatkan waktu pemulihan dengan logging biner, Anda dapat menggunakan binlog yang disempurnakan. Binlog yang disempurnakan mempersingkat waktu pemulihan basis data hingga 99%. Untuk informasi selengkapnya, lihat Menyiapkan binlog yang ditingkatkan.

- Aurora tidak memerlukan log biner untuk mereplikasi data dalam cluster DB atau untuk melakukan point-in-time restore (PITR).
- Jika Anda tidak membutuhkan log biner untuk replikasi eksternal (atau log stream biner eksternal), kami sarankan Anda mengatur parameter `binlog_format` ke `OFF` untuk menonaktifkan log biner. Hal ini mengurangi waktu pemulihan.

Untuk informasi selengkapnya tentang logging biner dan replikasi Aurora, lihat Replikasi dengan Amazon Aurora. Untuk informasi tentang implikasi tipe replikasi MySQL yang berbeda-beda, lihat Advantages and disadvantages of statement-based and row-based replication dalam dokumentasi MySQL.

Keamanan Amazon Aurora

Keamanan untuk Amazon Aurora dikelola di tiga tingkat:

- Untuk mengontrol siapa yang dapat melakukan tindakan manajemen Amazon RDS di klaster DB dan instans DB Aurora, Anda perlu menggunakan AWS Identity and Access Management (IAM). Jika Anda terhubung ke AWS menggunakan kredensial IAM, akun AWS Anda harus memiliki kebijakan IAM yang memberikan izin yang diperlukan untuk menjalankan operasi manajemen Amazon RDS. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Amazon Aurora.

  Jika Anda menggunakan IAM untuk mengakses konsol Amazon RDS, Anda harus masuk terlebih dahulu ke AWS Management Console dengan kredensial pengguna IAM Anda, lalu membuka konsol Amazon RDS di https://console.aws.amazon.com/rds.
- Klaster DB Aurora harus dibuat di cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. Untuk mengontrol perangkat dan instans Amazon EC2 mana yang dapat membuka koneksi ke titik akhir dan port instans DB untuk klaster DB Aurora di VPC, Anda perlu menggunakan grup keamanan VPC. Anda dapat membuat koneksi titik akhir dan port ini menggunakan Keamanan Lapisan Pengangkutan (TLS)/Lapisan Soket Aman (SSL). Selain itu, aturan firewall di perusahaan Anda dapat mengontrol apakah perangkat yang berjalan di perusahaan Anda dapat membuka koneksi ke instans DB. Untuk informasi selengkapnya tentang VPC, lihat Amazon VPC dan Amazon Aurora.
- Untuk mengautentikasi login dan izin untuk klaster DB Amazon Aurora, Anda dapat menggunakan salah satu pendekatan berikut, atau kombinasi dari beberapa pendekatan.

- Anda dapat menggunakan pendekatan yang sama seperti dengan instans DB mandiri MySQL atau PostgreSQL.

  Teknik untuk mengautentikasi login dan izin untuk instans DB mandiri MySQL atau PostgreSQL, seperti menggunakan perintah SQL atau memodifikasi tabel skema basis data, juga dapat dilakukan dengan Aurora. Untuk informasi selengkapnya, lihat Keamanan dengan Amazon Aurora MySQL atau Keamanan dengan Amazon Aurora PostgreSQL.

- Anda dapat menggunakan autentikasi basis data IAM.

  Dengan autentikasi basis data IAM, Anda mengautentikasi klaster DB Aurora Anda dengan menggunakan pengguna atau peran IAM dan token autentikasi. Token autentikasi adalah nilai unik yang dihasilkan dengan menggunakan proses penandatanganan Signature Versi 4. Dengan menggunakan autentikasi basis data IAM, Anda dapat menggunakan kredensial yang sama untuk mengontrol akses ke sumber daya AWS dan basis data Anda. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.

- Sekarang Anda dapat menggunakan autentikasi Kerberos untuk Aurora PostgreSQL dan Aurora MySQL.

  Sekarang Anda dapat menggunakan Kerberos untuk mengautentikasi pengguna saat mereka terhubung ke klaster DB Aurora PostgreSQL dan Aurora MySQL. Dalam hal ini, klaster DB Anda beroperasi dengan AWS Directory Service for Microsoft Active Directory untuk mengaktifkan autentikasi Kerberos. AWS Directory Service for Microsoft Active Directory disebut juga AWS Managed Microsoft AD. Dengan menyimpan semua kredensial Anda di direktori yang sama, Anda dapat menghemat waktu dan tenaga. Anda memiliki tempat terpusat untuk menyimpan dan mengelola kredensial untuk beberapa klaster DB. Penggunaan direktori juga dapat meningkatkan profil keamanan Anda secara keseluruhan. Untuk informasi selengkapnya, lihat Menggunakan autentikasi Kerberos dengan Aurora PostgreSQL dan Menggunakan autentikasi Kerberos untuk Aurora MySQL.

Untuk informasi tentang konfigurasi keamanan, lihat Keamanan dalam Amazon Aurora.

Menggunakan SSL dengan klaster DB Aurora

Klaster DB Amazon Aurora mendukung koneksi Lapisan Soket Aman (SSL) dari aplikasi menggunakan proses dan kunci publik yang sama seperti instans DB Amazon RDS. Untuk informasi selengkapnya, lihat Keamanan dengan Amazon Aurora MySQL, Keamanan dengan Amazon Aurora PostgreSQL, atau Menggunakan TLS/SSL dengan Aurora Serverless v1.

Ketersediaan yang tinggi untuk Amazon Aurora

Arsitektur Amazon Aurora melibatkan pemisahan penyimpanan dan komputasi. Aurora memiliki beberapa fitur ketersediaan tinggi yang berlaku untuk data dalam klaster DB Anda. Data tetap aman meskipun beberapa atau semua instans DB di klaster tidak tersedia. Fitur ketersediaan tinggi lainnya berlaku untuk instans DB. Fitur-fitur ini membantu memastikan bahwa satu atau lebih instans DB siap menangani permintaan basis data dari aplikasi Anda.

Topik

- Ketersediaan data Aurora yang tinggi
- Ketersediaan yang tinggi untuk instans DB Aurora
- Ketersediaan yang tinggi di seluruh Wilayah AWS dengan basis data global Aurora
- Toleransi kesalahan untuk klaster DB Aurora
- Ketersediaan yang tinggi dengan Proksi Amazon RDS

Ketersediaan data Aurora yang tinggi

Aurora menyimpan salinan data dalam klaster basis data di beberapa Zona Ketersediaan dalam satu Wilayah AWS. Aurora menyimpan salinan ini terlepas jika instans dalam klaster DB mencakup beberapa Zona Ketersediaan. Untuk informasi selengkapnya tentang Aurora, lihat Mengelola klaster DB Amazon Aurora.

Ketika data ditulis ke instans DB primer, Aurora secara sinkron mereplikasi data di seluruh Zona Ketersediaan ke enam simpul penyimpanan yang terkait dengan volume klaster Anda. Melakukan hal tersebut akan memberikan redundansi data, menghilangkan I/O freeze, dan meminimalkan lonjakan latensi selama pencadangan sistem. Menjalankan instans DB dengan ketersediaan tinggi dapat meningkatkan ketersediaan selama pemeliharaan sistem terencana, dan membantu melindungi basis data Anda terhadap kegagalan dan gangguan Zona Ketersediaan. Untuk informasi selengkapnya tentang Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan.

Ketersediaan yang tinggi untuk instans DB Aurora

Setelah Anda membuat instans primer (penulis), Anda dapat membuat hingga 15 Replika Aurora baca-saja. Replika Aurora juga dikenal sebagai instans pembaca.

Selama day-to-day operasi, Anda dapat menurunkan beberapa pekerjaan untuk aplikasi intensif baca dengan menggunakan instance pembaca untuk memproses kueri. `SELECT` Ketika masalah

memengaruhi instans utama, salah satu dari instans pembaca ini mengambil alih sebagai instans utama. Mekanisme ini dikenal sebagai failover. Banyak fitur Aurora berlaku pada mekanisme failover. Misalnya, Aurora mendeteksi masalah basis data dan mengaktifkan mekanisme failover secara otomatis jika diperlukan. Aurora juga memiliki fitur yang mengurangi waktu failover untuk menyelesaikannya. Melakukan hal ini meminimalkan waktu tidak tersedianya basis data untuk menulis selama failover.

Aurora dirancang untuk pulih secepat mungkin, dan jalur tercepat menuju pemulihan sering kali dimulai ulang atau gagal ke instans DB yang sama. Restart lebih cepat dan melibatkan lebih sedikit overhead daripada failover.

Untuk menggunakan string koneksi yang tetap sama bahkan ketika failover menaikkan instans primer baru, Anda terhubung ke titik akhir klaster. Titik akhir klaster selalu mewakili instans utama saat ini di klaster. Untuk informasi selengkapnya tentang titik akhir klaster, lihat Manajemen koneksi Amazon Aurora.

> **Tip**
>
> Dalam masing-masing Wilayah AWS, Availability Zones (AZ) mewakili lokasi yang berbeda satu sama lain untuk memberikan isolasi jika terjadi pemadaman. Kami menyarankan agar Anda mendistribusikan instans utama dan instans pembaca dalam klaster DB Anda di beberapa Availability Zone untuk meningkatkan ketersediaan klaster DB Anda. Dengan demikian, masalah yang memengaruhi seluruh Availability Zone tidak menyebabkan pemadaman bagi klaster Anda.
>
> Anda dapat mengatur cluster DB multi-AZ dengan membuat pilihan sederhana saat Anda membuat cluster. Anda dapat menggunakan AWS Management Console, AWS CLI, atau API Amazon RDS. Anda juga dapat mengonversi cluster Aurora DB yang ada menjadi cluster DB multi-AZ dengan menambahkan instans DB pembaca baru dan menentukan Availability Zone yang berbeda.

Ketersediaan yang tinggi di seluruh Wilayah AWS dengan basis data global Aurora

Untuk ketersediaan tinggi di beberapa Wilayah AWS, Anda dapat mengatur database global Aurora. Setiap database global Aurora mencakup beberapa Wilayah AWS, memungkinkan pembacaan global latensi rendah dan pemulihan bencana dari pemadaman di seluruh dunia. Wilayah AWS

Aurora secara otomatis menangani replikasi semua data dan pembaruan dari primer Wilayah AWS ke masing-masing Wilayah sekunder. Untuk informasi selengkapnya, lihat Menggunakan basis data global Amazon Aurora.

Toleransi kesalahan untuk klaster DB Aurora

Klaster DB Aurora toleran terhadap kesalahan secara default. Volume cluster mencakup beberapa Availability Zone (AZ) dalam satu Wilayah AWS, dan setiap Availability Zone berisi salinan data volume cluster. Fungsionalitas ini berarti bahwa klaster DB Anda dapat menoleransi kesalahan dari Zona Ketersediaan tanpa kehilangan data dan hanya berupa gangguan layanan yang singkat.

Jika instans DB primer saat ini pada suatu klaster DB gagal, Aurora secara otomatis melakukan failover ke instans DB primer yang baru.

- Dengan menaikkan Replika Aurora yang sudah ada ke instans utama yang baru
- Dengan membuat instans primer baru

Jika klaster DB memiliki satu atau lebih Replika Aurora, maka Replika Aurora dipromosikan ke instans primer selama peristiwa failover. Peristiwa kegagalan mengakibatkan interupsi singkat, selama operasi baca dan tulis gagal dengan pengecualian. Namun, layanan biasanya dipulihkan dalam waktu kurang dari 60 detik, dan sering kali kurang dari 30 detik. Untuk meningkatkan ketersediaan klaster DB Anda, kami sarankan Anda membuat setidaknya satu Replika Aurora atau lebih di dua Zona Ketersediaan yang berbeda.

> **Tip**
>
> Di Aurora MySQL 2.10 dan yang lebih tinggi, Anda dapat meningkatkan ketersediaan selama failover dengan menyediakan lebih dari satu instans DB pembaca dalam sebuah klaster. Di Aurora MySQL 2.10 dan yang lebih tinggi, Aurora memulai ulang hanya instans DB penulis dan instans pembaca yang gagal. Instans pembaca lain dalam klaster ini tetap tersedia selama failover untuk terus memproses kueri melalui koneksi ke titik akhir pembaca.
> Anda juga dapat meningkatkan ketersediaan selama failover dengan menggunakan Proksi RDS dengan klaster DB Aurora Anda. Untuk informasi selengkapnya, lihat Ketersediaan yang tinggi dengan Proksi Amazon RDS.

Anda dapat menyesuaikan urutan di mana Replika Aurora Anda dinaikkan ke instans utama setelah kegagalan dengan menetapkan masing-masing replika sebagai prioritas. Prioritas berkisar dari

0 untuk prioritas tertinggi hingga 15 untuk prioritas terendah. Jika instans primer gagal, Amazon RDS mempromosikan Replika Aurora dengan prioritas yang paling tinggi untuk instans primer baru. Anda dapat mengubah prioritas dari Replika Aurora kapan saja. Memodifikasi prioritas tidak memicu failover.

Lebih dari satu Replika Aurora dapat memiliki prioritas yang sama, yang menghasilkan tingkat promosi. Jika dua atau lebih Replika Aurora memiliki prioritas yang sama, maka Amazon RDS menaikkan replika dengan ukuran paling besar. Jika dua atau lebih Replika Aurora memiliki prioritas yang sama, maka Amazon RDS menaikkan replika bebas dengan tingkat promosi yang sama.

Jika klaster DB tidak berisi Replika Aurora, maka instans primer dibuat ulang di AZ yang sama selama peristiwa kegagalan. Peristiwa kegagalan mengakibatkan interupsi yang operasi baca dan tulisnya gagal dengan pengecualian. Layanan dipulihkan ketika instans primer baru dibuat, yang biasanya memakan waktu kurang dari 10 menit. Mempromosikan Replika Aurora ke instans primer jauh lebih cepat daripada membuat instans primer baru.

Misalkan instans primer di klaster Anda tidak tersedia karena pemadaman yang memengaruhi keseluruhan AZ. Dalam hal ini, cara untuk menjadikan instans primer baru online tergantung pada apakah klaster Anda menggunakan konfigurasi Multi-AZ:

- Jika klaster atau Aurora Serverless v2 yang tersedia berisi instans pembaca apa pun di AZ lain, Aurora menggunakan mekanisme failover untuk mempromosikan salah satu instans pembaca menjadi instans primer baru.
- Jika klaster atau Aurora Serverless v2 yang tersedia hanya berisi satu instans DB, atau jika instans primer dan semua instans pembaca berada di AZ yang sama, pastikan untuk secara manual membuat satu atau beberapa instans DB baru di AZ lain.
- Jika klaster Anda menggunakan Aurora Serverless v1, Aurora secara otomatis membuat instans DB baru di AZ lain. Namun, proses ini melibatkan penggantian host, sehingga membutuhkan waktu lebih lama dari failover.

> **Note**
>
> Amazon Aurora juga mendukung replikasi dengan basis data MySQL eksternal, atau instans DB RDS MySQL. Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner).

Ketersediaan yang tinggi dengan Proksi Amazon RDS

Dengan Proksi RDS, Anda dapat membangun aplikasi yang dapat secara transparan mentolerir kegagalan basis data tanpa perlu menulis kode penanganan kegagalan yang kompleks. Proksi RDS secara otomatis merutekan lalu lintas ke instans basis data baru sekaligus mempertahankan koneksi aplikasi. Proksi RDS juga melewati cache Sistem Nama Domain (DNS) untuk mengurangi waktu failover hingga 66% untuk basis data Multi-AZ Aurora. Lihat informasi yang lebih lengkap di Menggunakan Proksi Amazon RDS untuk Aurora.

Replikasi dengan Amazon Aurora

Ada beberapa opsi replikasi dengan Aurora. Setiap klaster DB Aurora memiliki replikasi bawaan di antara beberapa instans DB dalam klaster yang sama. Anda juga dapat mengatur replikasi dengan klaster Aurora sebagai sumber atau target. Ketika mereplikasi data ke dalam atau ke luar dari klaster Aurora, Anda dapat memilih antara fitur bawaan seperti basis data global Aurora atau mekanisme replikasi biasa untuk mesin MySQL atau PostgreSQL DB. Anda dapat memilih opsi yang sesuai berdasarkan fitur mana yang memberikan kombinasi yang tepat antara ketersediaan, kepraktisan, dan performa yang tinggi untuk kebutuhan Anda. Bagian berikut ini menjelaskan bagaimana dan kapan harus memilih masing-masing teknik.

Topik

- Replika Aurora
- Replikasi dengan Aurora MySQL
- Replikasi dengan Aurora PostgreSQL

Replika Aurora

Ketika Anda membuat instans DB kedua, ketiga, dan seterusnya dalam klaster DB yang disediakan Aurora, Aurora secara otomatis mengatur replikasi dari instans DB penulis ke semua instans DB lainnya. Instans DB lainnya ini bersifat hanya baca dan dikenal sebagai Replika Aurora. Kami juga menyebutnya sebagai instans pembaca ketika membahas cara-cara bagi Anda untuk menggabungkan instans DB penulis dan pembaca dalam sebuah klaster.

Replika Aurora memiliki dua tujuan utama. Anda dapat menerbitkan kueri ke replika ini untuk menskalakan operasi baca untuk aplikasi Anda. Anda biasanya melakukannya dengan menghubungkan ke titik akhir pembaca pada klaster. Dengan demikian, Aurora dapat menyebarkan beban untuk koneksi hanya baca ke sebanyak Replika Aurora yang Anda miliki di klaster. Replika

Aurora juga membantu meningkatkan ketersediaan. Jika instans penulis dalam klaster menjadi tidak tersedia, Aurora secara otomatis mempromosikan salah satu instans pembaca untuk menggantikannya sebagai penulis baru.

Sebuah klaster DB Aurora dapat berisi hingga 15 Replika Aurora. Replika Aurora dapat didistribusikan ke seluruh Zona Ketersediaan yang dicakup oleh klaster DB di sebuah Wilayah AWS .

Data dalam klaster DB Anda memiliki fitur ketersediaan tinggi dan keandalannya sendiri, yang terpisah dari instans DB di klaster. Jika Anda belum memahami fitur penyimpanan Aurora, lihat Gambaran umum penyimpanan Amazon Aurora. Volume klaster DB secara fisik terdiri atas beberapa salinan data untuk klaster DB. Instans primer dan Replika Aurora di klaster DB melihat data dalam volume klaster sebagai volume logis tunggal.

Akibatnya, semua Replika Aurora menghasilkan data yang sama untuk hasil kueri dengan sedikit lag replika. Lag ini biasanya kurang dari 100 milidetik setelah instans primer menulis pembaruan. Lag replika bervariasi bergantung pada laju perubahan basis data. Artinya, selama periode saat sejumlah besar operasi tulis terjadi untuk basis data, Anda mungkin melihat peningkatan lag replika.

> **Note**
>
> Aurora Replica restart, ketika kehilangan komunikasi dengan instans DB penulis selama lebih dari 60 detik dalam versi PostgreSQL Aurora berikut:
>
> - 14.6 dan versi yang lebih lama
> - 13.9 dan versi yang lebih lama
> - 12.13 dan versi yang lebih lama
> - Semua Aurora PostgreSQL 11 versi

Replika Aurora berfungsi dengan baik untuk penskalaan baca karena ditujukan sepenuhnya untuk operasi baca pada volume klaster Anda. Operasi tulis dikelola oleh instans primer. Karena volume klaster dibagikan di antara semua instans dalam klaster DB Aurora Anda, pekerjaan tambahan untuk mereplikasi salinan data untuk setiap Replika Aurora menjadi minimal.

Untuk meningkatkan ketersediaan, Anda dapat menggunakan Replika Aurora sebagai target failover. Artinya, jika instans primer gagal, Replika Aurora dipromosikan menjadi instans primer. Akan ada interupsi singkat saat permintaan baca dan tulis yang dibuat ke instans primer mengalami kegagalan dengan pengecualian.

Mempromosikan Replika Aurora melalui failover jauh lebih cepat daripada membuat ulang instans primer. Jika klaster DB Aurora Anda tidak menyertakan Replika Aurora, klaster DB Anda tidak akan tersedia selama instans DB Anda melakukan pemulihan dari kegagalan.

Ketika failover terjadi, beberapa Replika Aurora mungkin di-boot ulang, bergantung pada versi mesin DB. Misalnya, di Aurora MySQL 2.10 dan lebih tinggi, Aurora akan mengaktifkan ulang hanya instans DB penulis dan target failover selama failover. Untuk informasi selengkapnya tentang perilaku boot ulang versi mesin DB Aurora yang berbeda-beda, lihat Mem-boot ulang klaster DB Amazon Aurora atau instans DB Amazon Aurora. Untuk informasi tentang apa yang terjadi pada cache halaman saat boot ulang atau failover, lihat Cache halaman yang dapat bertahan.

Untuk skenario ketersediaan tinggi, kami sarankan Anda membuat satu atau beberapa Replika Aurora. Replika ini harus berasal dari kelas instans DB yang sama dengan instans primer dan berada di Zona Ketersediaan yang berbeda untuk klaster DB Aurora Anda. Untuk informasi selengkapnya tentang Replika Aurora sebagai target failover, lihat Toleransi kesalahan untuk klaster DB Aurora.

Anda tidak dapat membuat Replika Aurora terenkripsi untuk klaster DB Aurora yang tidak terenkripsi. Anda tidak dapat membuat Replika Aurora yang tidak terenkripsi untuk klaster DB Aurora terenkripsi.

> **Tip**
>
> Anda dapat menggunakan Replika Aurora dalam klaster Aurora sebagai satu-satunya bentuk replikasi untuk mempertahankan ketersediaan tinggi data Anda. Anda juga dapat menggabungkan replikasi Aurora bawaan dengan jenis replikasi lainnya. Tindakan tersebut dapat membantu menambah tingkat ketersediaan tinggi dan distribusi geografis data Anda.

Untuk detail tentang cara membuat Replika Aurora, lihat Menambahkan Replika Aurora ke klaster DB.

Replikasi dengan Aurora MySQL

Selain Replika Aurora, Anda memiliki opsi berikut untuk replikasi dengan Aurora MySQL:

- Aurora MySQL DB cluster di Wilayah yang berbeda. AWS
  - Anda dapat mereplikasi data di beberapa Wilayah menggunakan basis data global Aurora. Untuk detailnya, lihat Ketersediaan yang tinggi di seluruh Wilayah AWS dengan basis data global Aurora

  - Anda dapat membuat replika baca Aurora dari cluster DB MySQL Aurora di AWS Wilayah yang berbeda, dengan menggunakan replikasi log biner MySQL (binlog). Setiap klaster dapat memiliki hingga lima replika baca yang dibuat dengan cara ini, masing-masing di Wilayah berbeda.
- Dua klaster DB Aurora MySQL di Wilayah yang sama, dengan menggunakan replikasi log biner (binlog) MySQL.
- Instans DB RDS for MySQL sebagai data sumber dan klaster DB Aurora MySQL, dengan membuat replika baca Aurora dari instans DB RDS for MySQL. Biasanya, pendekatan ini digunakan untuk migrasi ke Aurora MySQL, bukan replikasi berkelanjutan.

Untuk informasi selengkapnya tentang replikasi dengan Aurora MySQL, lihat Replikasi dengan Amazon Aurora MySQL.

Replikasi dengan Aurora PostgreSQL

Selain Replika Aurora, Anda memiliki opsi berikut untuk replikasi dengan Aurora PostgreSQL:

- Klaster DB primer Aurora di suatu Wilayah dan hingga lima klaster DB sekunder hanya baca di Wilayah yang berbeda menggunakan basis data global Aurora. Aurora PostgreSQL tidak mendukung Replika Aurora lintas Wilayah. Namun, Anda dapat menggunakan database global Aurora untuk menskalakan kemampuan baca klaster PostgreSQL DB Aurora Anda ke lebih dari satu Wilayah dan untuk memenuhi tujuan ketersediaan. AWS Untuk informasi selengkapnya, lihat Menggunakan basis data global Amazon Aurora.
- Dua klaster DB Aurora PostgreSQL di Wilayah yang sama, menggunakan fitur replika logis PostgreSQL.
- Instans DB RDS for PostgreSQL sebagai sumber data dan klaster DB Aurora PostgreSQL, dengan membuat replika baca Aurora dari instans DB PostgreSQL. Biasanya, pendekatan ini digunakan untuk migrasi ke Aurora PostgreSQL, bukan replikasi berkelanjutan.

Untuk informasi selengkapnya tentang replikasi dengan Aurora PostgreSQL, lihat Replikasi dengan Amazon Aurora PostgreSQL.

Penagihan instans DB untuk Aurora

Instans yang disediakan Amazon RDS dalam klaster Amazon Aurora ditagih berdasarkan komponen berikut:

- Jam instans DB (per jam) – Berdasarkan kelas instans DB dari instans DB (misalnya, db.t2.small atau db.m4.large). Harga dicantumkan per jam, tetapi tagihan dihitung turun menjadi detik dan menunjukkan waktu dalam bentuk desimal. Penggunaan RDS ditagih dalam setiap kenaikan 1 detik, dengan minimal 10 menit. Untuk informasi selengkapnya, lihat Kelas instans DB Aurora.
- Penyimpanan (per GiB per bulan) – Kapasitas penyimpanan yang telah Anda sediakan untuk instans DB Anda. Jika Anda menskalakan kapasitas penyimpanan yang telah Anda sediakan dalam bulan tertentu, tagihan Anda akan diprorata. Untuk informasi selengkapnya, lihat Penyimpanan dan keandalan Amazon Aurora.
- Permintaan input/output (I/O) (per 1 juta permintaan) – Total jumlah permintaan I/O penyimpanan yang telah Anda buat dalam siklus penagihan, hanya untuk konfigurasi klaster DB Aurora Standard.

  Untuk informasi selengkapnya tentang penagihan I/O Amazon Aurora, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.
- Penyimpanan cadangan (per GiB per bulan) – Penyimpanan cadangan adalah penyimpanan yang terkait dengan cadangan basis data otomatis dan setiap snapshot basis data aktif yang telah Anda ambil. Meningkatkan periode retensi cadangan atau mengambil snapshot basis data tambahan akan meningkatkan penyimpanan cadangan yang digunakan oleh basis data Anda. Tagihan per detik tidak berlaku untuk penyimpanan cadangan (diukur dalam GB-bulan).

  Untuk informasi selengkapnya, lihat Mencadangkan dan memulihkan klaster DB Amazon Aurora.
- Transfer data (per GB) – Transfer data yang masuk dan keluar dari instans DB Anda dari atau ke internet dan Wilayah AWS lainnya.

Amazon RDS menyediakan opsi pembelian berikut yang memungkinkan Anda mengoptimalkan biaya berdasarkan kebutuhan Anda:

- Instans Sesuai Permintaan – Bayar per jam untuk periode instans DB yang Anda gunakan. Harga dicantumkan per jam, tetapi tagihan dihitung turun menjadi detik dan menunjukkan waktu dalam bentuk desimal. Penggunaan RDS kini ditagih setiap kenaikan 1 detik, dengan minimal 10 menit.
- Instans terpesan – Memesan instans DB untuk jangka waktu satu tahun atau tiga tahun dan mendapatkan diskon yang signifikan dibandingkan dengan harga instans DB sesuai permintaan. Dengan penggunaan Instans Terpesan, Anda dapat meluncurkan, menghapus, memulai, atau menghentikan beberapa instans dalam satu jam dan mendapatkan keuntungan Instans Terpesan untuk semua instans.

- Aurora Serverless v2 – Aurora Serverless v2 menyediakan kapasitas sesuai permintaan di mana unit penagihan adalah jam unit kapasitas Aurora (ACU), bukan jam instans DB. Kapasitas Aurora Serverless v2 naik dan turun, dalam rentang yang Anda tentukan, bergantung pada muatan pada basis data Anda. Anda dapat mengkonfigurasi klaster jika semua kapasitasnya adalah Aurora Serverless v2. Atau Anda dapat mengonfigurasi kombinasi Aurora Serverless v2 dan instans yang tersedia sesuai permintaan atau terpesan. Untuk informasi selengkapnya tentang cara kerja ACU Aurora Serverless v2, lihat Cara kerja Aurora Serverless v2.

Untuk informasi tentang harga Aurora, lihat Halaman harga Aurora.

Topik

- Instans Sesuai Permintaan untuk Aurora
- Instans DB terpesan untuk Aurora

Instans Sesuai Permintaan untuk Aurora

Penagihan instans DB Amazon RDS sesuai permintaan didasarkan pada kelas instans DB (misalnya, db.t3.smal atau db.m5.large). Untuk informasi tentang harga Amazon RDS, lihat Halaman harga Amazon RDS.

Penagihan dimulai untuk instans DB segera setelah instans DB tersedia. Harga dicantumkan per jam, tetapi tagihan dihitung turun menjadi detik dan menunjukkan waktu dalam bentuk desimal. Penggunaan Amazon RDS ditagih dalam setiap kenaikan satu detik, dengan minimum 10 menit. Dalam hal perubahan konfigurasi yang dapat ditagih, seperti penghitungan skala atau kapasitas penyimpanan, Anda dikenai biaya minimum 10 menit. Penagihan berlanjut hingga instans DB berakhir, yaitu pada saat Anda menghapus instans DB atau jika instans DB gagal.

Jika tidak ingin dikenai biaya lagi untuk instans DB, Anda harus menghentikan atau menghapusnya agar tidak ada tagihan untuk instans DB tambahan per jam. Untuk informasi selengkapnya tentang status instans DB yang ditagih, lihat Melihat status instans DB di klaster Aurora.

Instans DB tang dihentikan

Ketika instans DB Anda dihentikan, Anda akan dikenai biaya untuk penyimpanan yang disediakan, termasuk IOPS yang Tersedia. Anda juga dikenai biaya untuk penyimpanan cadangan, termasuk penyimpanan untuk snapshot manual dan cadangan otomatis dalam periode retensi yang Anda tentukan. Anda tidak dikenai biaya untuk jam instans DB.

Instans DB Multi-AZ

Jika Anda menentukan bahwa instans DB harus berupa deployment Multi-AZ, Anda akan ditagih berdasarkan harga Multi-AZ yang diposting di halaman harga Amazon RDS.

Instans DB terpesan untuk Aurora

Dengan menggunakan instans DB terpesan, Anda dapat memesan instans DB untuk jangka waktu satu atau tiga tahun. Instans DB terpesan memberikan diskon yang signifikan dibandingkan harga instans DB sesuai permintaan. Instans DB terpesan bukan merupakan instans fisik, melainkan diskon penagihan yang diterapkan untuk penggunaan instans DB tertentu sesuai permintaan dalam akun Anda. Diskon untuk instans DB terpesan terikat dengan jenis instans dan Wilayah AWS.

Proses umum untuk menggunakan instans DB terpesan adalah: Pertama, dapatkan informasi tentang penawaran instans DB terpesan yang tersedia, kemudian beli penawaran instans DB terpesan, dan terakhir dapatkan informasi tentang instans DB terpesan yang ada.

Ikhtisar instans DB terpesan

Saat membeli instans DB terpesan di Amazon RDS, Anda membeli komitmen untuk mendapatkan tarif diskon, pada jenis instans DB tertentu, selama durasi instans DB terpesan. Untuk menggunakan instans DB terpesan Amazon RDS, Anda membuat instans DB baru seperti yang Anda lakukan untuk instans sesuai permintaan.

Instans DB baru yang Anda buat harus memiliki spesifikasi yang sama dengan instans DB terpesan untuk hal berikut:

- Wilayah AWS
- Mesin DB
- Jenis instans DB

Jika spesifikasi instans DB baru cocok dengan instans DB terpesan yang sudah ada untuk akun Anda, Anda akan ditagih dengan tarif diskon yang ditawarkan untuk instans DB terpesan. Jika tidak, instans DB ditagih dengan tarif sesuai permintaan.

Anda dapat memodifikasi instans DB yang Anda gunakan sebagai instans DB terpesan. Jika modifikasi sesuai dengan spesifikasi instans DB terpesan, sebagian atau seluruh diskon masih akan berlaku untuk instans DB yang dimodifikasi. Jika modifikasi berada di luar spesifikasi, seperti mengubah kelas instans, diskon tidak lagi berlaku. Untuk informasi selengkapnya, lihat Instans DB terpesan berukuran fleksibel.

Topik

- Jenis penawaran

- Fleksibilitas konfigurasi klaster DB Aurora
- Instans DB terpesan berukuran fleksibel
- Contoh penagihan instans DB terpesan Aurora
- Menghapus instans DB terpesan

Untuk informasi selengkapnya tentang instans DB terpesan, termasuk harga, lihat instans terpesan Amazon RDS.

Jenis penawaran

Instans DB terpesan tersedia dalam tiga varietas—Tanpa Uang Muka, Uang Muka Sebagian, dan Uang Muka Penuh—yang memungkinkan Anda mengoptimalkan biaya Amazon RDS berdasarkan perkiraan penggunaan Anda.

Tanpa Uang Muka

Opsi ini menyediakan akses ke instans DB terpesan tanpa harus membayar di muka. Instans DB terpesan Tanpa Uang Muka akan menagih tarif per jam yang didiskon untuk setiap jam dalam jangka waktu pemesanan, terlepas dari penggunaannya, dan tidak perlu membayar di muka. Opsi ini hanya tersedia sebagai pemesanan satu tahun.

Uang Muka Sebagian

Opsi ini mengharuskan pembayaran di muka untuk sebagian instans DB terpesan. Sisa jam dalam jangka waktu pemesanan akan ditagih dengan tarif per jam yang didiskon, terlepas dari penggunaannya. Opsi ini adalah pengganti untuk opsi Penggunaan Berat sebelumnya.

Uang Muka Penuh

Pembayaran penuh dilakukan di awal jangka waktu pemesanan, tanpa biaya lain untuk sisa jangka waktu pemesanan, terlepas dari jumlah jam yang digunakan.

Jika Anda menggunakan penagihan gabungan, semua akun dalam organisasi akan dianggap sebagai satu akun. Berarti semua akun dalam organisasi dapat menerima manfaat biaya per jam dari instans DB terpesan yang dibeli oleh akun lain. Untuk informasi selengkapnya tentang penagihan gabungan, lihat instans DB terpesan Amazon RDS dalam Panduan Pengguna Manajemen Biaya dan Penagihan AWS .

Fleksibilitas konfigurasi klaster DB Aurora

Anda dapat menggunakan instans DB terpesan Aurora dengan kedua konfigurasi klaster DB:

- Aurora I/O-Optimized – Anda hanya membayar penggunaan dan penyimpanan klaster DB Anda, tanpa biaya tambahan untuk operasi I/O baca dan tulis.
- Aurora Standard – Selain penggunaan dan penyimpanan klaster DB, Anda juga membayar tarif standar per 1 juta permintaan untuk operasi I/O.

Aurora secara otomatis memperhitungkan perbedaan harga antara konfigurasi ini. Aurora I/O-Optimized menghabiskan lebih dari 30% unit per jam yang dinormalkan dibandingkan Aurora Standard.

Untuk informasi selengkapnya tentang konfigurasi penyimpanan klaster Aurora, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora. Untuk informasi selengkapnya tentang harga untuk konfigurasi penyimpanan klaster Aurora, lihat Harga Amazon Aurora.

Instans DB terpesan berukuran fleksibel

Saat membeli instans DB terpesan, satu hal yang Anda tentukan adalah kelas instans, misalnya, db.r5.large. Untuk informasi selengkapnya tentang kelas instans DB, lihat Kelas instans DB Aurora.

Jika Anda memiliki instans DB, dan perlu menskalakan kapasitasnya ke ukuran yang lebih besar, instans DB terpesan Anda secara otomatis akan diterapkan pada instans DB yang diskalakan. Artinya, instans DB terpesan Anda secara otomatis diterapkan ke semua ukuran kelas instans DB. Instans DB cadangan yang fleksibel dengan ukuran tersedia untuk instans DB dengan mesin database yang sama. Wilayah AWS Instans DB terpesan berukuran fleksibel hanya dapat diskalakan di jenis kelas instans-nya. Misalnya, instans DB terpesan untuk db.r5.large dapat diterapkan ke db.r5.xlarge, tetapi tidak ke db.r6g.large, karena db.r5 dan db.r6g adalah jenis kelas instans yang berbeda.

Manfaat instans DB terpesan juga berlaku untuk konfigurasi Multi-AZ dan AZ-Tunggal. Fleksibilitas berarti bahwa Anda dapat berpindah dengan bebas di antara konfigurasi dengan jenis kelas instans DB yang sama. Misalnya, Anda dapat berpindah dari penerapan AZ tunggal yang berjalan pada satu instans DB besar (empat unit dinormalisasi per jam) ke penerapan multi-AZ yang berjalan pada dua instans DB sedang (2+2 = 4 unit dinormalisasi per jam).

Instans DB terpesan berukuran fleksibel tersedia untuk mesin basis data Amazon RDS berikut:

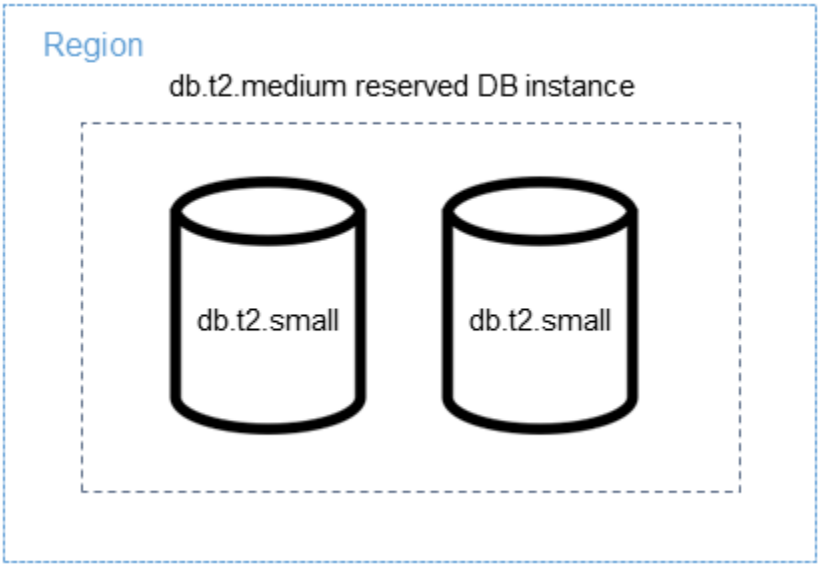
- Aurora MySQL

- Aurora PostgreSQL

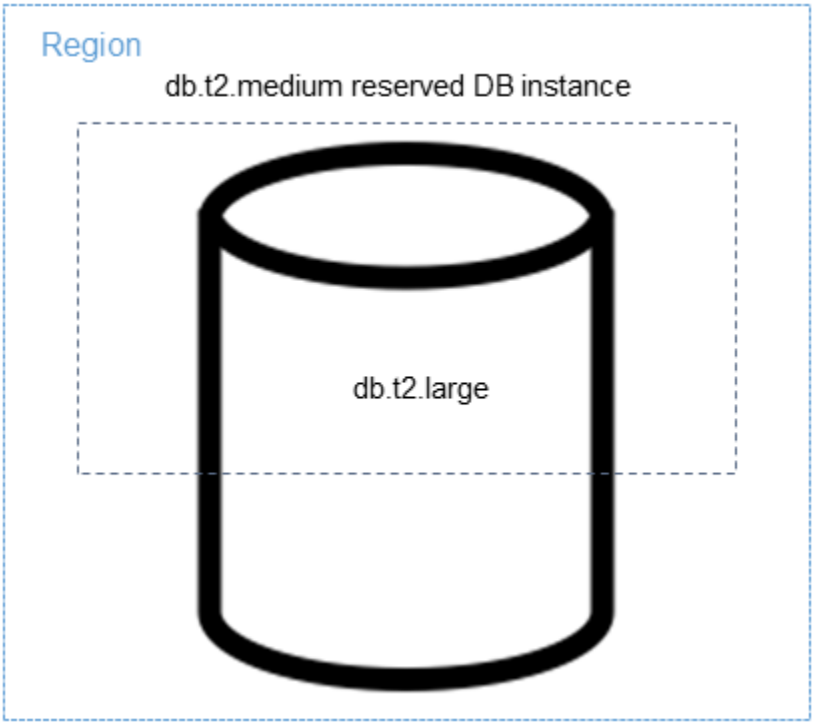
Anda dapat membandingkan penggunaan untuk ukuran instans DB terpesan yang berbeda dengan menggunakan unit per jam yang dinormalkan. Misalnya, satu unit penggunaan di dua instans DB db.r3.large setara dengan delapan unit per jam penggunaan yang dinormalkan di satu db.r3.small. Tabel berikut menunjukkan jumlah unit per jam yang dinormalkan untuk setiap ukuran instans DB.

Ukuran instans	Unit per jam yang dinormalkan untuk satu instans DB, Aurora Standard	Unit per jam yang dinormalkan untuk satu instans DB, Aurora I/O-Optimized	Unit per jam yang dinormalkan untuk tiga instans DB (penulis dan dua pembaca), Aurora Standard	Unit per jam yang dinormalkan untuk tiga instans DB (penulis dan dua pembaca), Aurora I/O-Optimized
kecil	1	1.3	3	3.9
sedang	2	2.6	6	7.8
besar	4	5.2	12	15.6
xlarge	8	10.4	24	31.2
2xlarge	16	20.8	48	62.4
4xlarge	32	41,6	96	124,8
8xlarge	64	83.2	192	249.6
12xlarge	96	124,8	288	374.4
16xlarge	128	166.4	384	499.2
24xlarge	192	249.6	576	748,8
32xlarge	256	332.8	768	998,4

Misalnya, Anda membeli instans DB terpesan `db.t2.medium`, dan Anda memiliki dua instans DB `db.t2.small` yang berjalan di akun Anda pada Wilayah AWS yang sama. Dalam hal ini, manfaat penagihan diterapkan sepenuhnya pada kedua instans.



Atau, jika Anda memiliki satu `db.t2.large` instans yang berjalan di akun Anda dalam hal yang sama Wilayah AWS, manfaat penagihan diterapkan ke 50 persen dari penggunaan instans DB.



> **Note**
>
> Kami merekomendasikan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server nonproduksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Jenis kelas instans DB.

Contoh penagihan instans DB terpesan Aurora

Contoh berikut menggambarkan harga untuk instans DB terpesan untuk klaster DB Aurora menggunakan konfigurasi klaster DB Aurora Standard dan Aurora I/O-Optimized.

Contoh menggunakan Aurora Standard

Harga untuk instans DB terpesan tidak memberikan diskon untuk biaya yang terkait dengan penyimpanan, cadangan, dan I/O. Contoh berikut menggambarkan total biaya per bulannya untuk instans DB terpesan:

- Kelas instans DB db.r5.large AZ-Tunggal terpesan untuk Aurora MySQL di AS Timur (Virginia Utara) dengan biaya sebesar $0,19 per jam, atau $138,70 per bulan
- Penyimpanan Aurora dengan biaya sebesar $0,10 per GiB per bulan (dengan asumsi $45,60 per bulan untuk contoh ini)
- Aurora I/O dengan biaya $0,20 per 1 juta permintaan (dengan asumsi $20 per bulan untuk contoh ini)
- Penyimpanan cadangan Aurora dengan biaya sebesar $0,021 per GiB per bulan (dengan asumsi $30 per bulan untuk contoh ini)

Tambahkan semua biaya ini ($138,70 + $45,60 + $20 + $30) dengan instans DB terpesan, dan total biaya per bulannya adalah $234,30.

Jika Anda memilih untuk menggunakan instans DB sesuai permintaan dan bukannya instans DB terpesan, biaya kelas instans DB db.r5.large Tunggal-AZ pada Aurora MySQL di AS Timur (Virginia Utara) adalah sebesar $0,29 per jam, atau $217,50 per bulan. Jadi, untuk instans DB sesuai permintaan, tambahkan semua opsi ini ($217,50 + $45,60 + $20 + $30), dan total biaya per bulannya adalah $313,10. Anda menghemat hampir $79 per bulan dengan menggunakan instans DB terpesan.

Contoh menggunakan klaster DB Aurora Standard dengan dua instans pembaca

Untuk menggunakan instans terpesan untuk klaster DB Aurora, cukup beli satu instans terpesan untuk setiap instans DB dalam klaster.

Memperluas contoh pertama, Anda memiliki klaster DB Aurora MySQL dengan satu instans DB penulis dan dua Replika Aurora, untuk total tiga instans DB dalam klaster. Kedua Replika Aurora tidak dikenakan biaya penyimpanan atau cadangan tambahan. Jika Anda membeli tiga instans DB terpesan Aurora MySQL db.r5.large, biaya Anda akan menjadi $234,30 (untuk instans DB penulis) + 2 * ($138,70 + $20 I/O per Replika Aurora), dengan total $551,70 per bulan.

Biaya sesuai permintaan yang sesuai untuk klaster DB Aurora MySQL dengan satu instans DB penulis dan dua Replika Aurora adalah $313,10 + 2 * ($217,50 + $20 I/O per instans) dengan total $788,10 per bulan. Anda menghemat $236,40 per bulan dengan menggunakan instans DB terpesan.

Contoh menggunakan Aurora I/O-Optimized

Anda dapat menggunakan kembali instans DB terpesan Aurora Standard Anda yang ada dengan Aurora I/O-Optimized. Untuk menggunakan sepenuhnya manfaat diskon instans terpesan Anda dengan Aurora I/O-Optimized, Anda dapat membeli 30% instans terpesan tambahan yang sama dengan instans terpesan Anda saat ini.

Tabel berikut menampilkan contoh cara memperkirakan instans terpesan tambahan saat menggunakan Aurora I/O-Optimized. Jika instans terpesan yang diperlukan adalah sebagian kecil, Anda dapat memanfaatkan fleksibilitas ukuran yang tersedia dengan instans terpesan untuk mendapatkan bilangan bulat. Dalam contoh ini, "saat ini" mengacu pada instans terpesan Aurora Standard yang Anda miliki sekarang. Instans terpesan tambahan adalah jumlah instans terpesan Aurora Standard yang harus Anda beli untuk mempertahankan diskon instans terpesan saat ini ketika menggunakan Aurora I/O-Optimized.

Kelas instans DB	Instans terpesan Aurora Standard saat ini	Instans terpesan yang diperlukan untuk Aurora I/O-Optimized	Diperlukan instans terpesan tambahan	Diperlukan instans terpesan tambahan, menggunakan fleksibilitas ukuran
db.r6g.large	10	10 * 1,3 = 13	3 * db.r6g.large	3 * db.r6g.large

Kelas instans DB	Instans terpesan Aurora Standard saat ini	Instans terpesan yang diperlukan untuk Aurora I/O-Optimized	Diperlukan instans terpesan tambahan	Diperlukan instans terpesan tambahan, menggunakan fleksibilitas ukuran
db.r6g.4xlarge	20	20 * 1,3 = 26	6 * db.r6g.4xlarge	6 * db.r6g.4xlarge
db.r6g.12xlarge	5	5 * 1,3 = 6,5	1,5 * db.r6g.12xlarge	Masing-masing satu db.r6g.12xlarge, r6g.4xlarge, dan r6g.2xlarge (0,5 * db.r6g.12xlarge = 1 * db.r6g.4xlarge + 1 * db.r6g.2xlarge)
db.r6i.24xlarge	15	15 * 1,3 = 19,5	4,5 * db.r6i.24xlarge	4 * db.r6i.24xlarge + 1 * db.r6i.12xlarge (0,5 * db.r6i.24xlarge = 1 * db.r6i.12xlarge)

Contoh menggunakan klaster DB Aurora I/O-Optimized dengan dua instans pembaca

Anda memiliki klaster DB Aurora MySQL dengan satu instans DB penulis dan dua Replika Aurora, dengan total tiga instans DB dalam klaster. Semuanya menggunakan konfigurasi klaster DB Aurora I/O-Optimized. Untuk menggunakan instans DB terpesan untuk klaster ini, Anda perlu membeli empat instans DB terpesan dengan kelas instans DB yang sama. Tiga instans DB yang menggunakan Aurora I/O-Optimized menghabiskan 3,9 unit per jam yang dinormalkan, dibandingkan dengan 3 unit

per jam yang dinormalkan untuk tiga instans DB yang menggunakan Aurora Standard. Namun, Anda menghemat biaya I/O bulanan untuk setiap instans DB.

> **Note**
>
> Harga dalam contoh ini adalah harga contoh dan mungkin tidak sesuai dengan harga aktual. Untuk informasi harga Aurora, lihat Harga Amazon Aurora.

Menghapus instans DB terpesan

Syarat untuk instans DB terpesan melibatkan komitmen satu tahun atau tiga tahun. Anda tidak dapat membatalkan instans DB terpesan. Namun, Anda dapat menghapus instans DB yang dicakup oleh diskon instans DB terpesan. Proses penghapusan instans DB yang dicakup oleh diskon instans DB terpesan sama dengan instans DB lainnya.

Anda ditagih untuk biaya di muka, terlepas dari apakah Anda menggunakan sumber daya atau tidak.

Jika Anda menghapus instans DB yang dicakup oleh diskon instans DB terpesan, Anda dapat meluncurkan instans DB lain dengan spesifikasi yang kompatibel. Dalam hal ini, Anda tetap mendapatkan tarif diskon selama jangka waktu reservasi (satu atau tiga tahun).

Bekerja dengan instans DB terpesan

Anda dapat menggunakan API AWS Management Console, the AWS CLI, dan RDS untuk bekerja dengan instans DB yang dicadangkan.

Konsol

Anda dapat menggunakan AWS Management Console untuk bekerja dengan instans DB cadangan seperti yang ditunjukkan dalam prosedur berikut.

Untuk mendapatkan harga dan informasi tentang penawaran instans DB yang tersedia

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Instans terpesan.
3. Pilih Beli Instans DB Terpesan.
4. Untuk Deskripsi produk, pilih mesin DB dan jenis lisensi.
5. Untuk Kelas instans DB, pilih kelas instans DB.

6. Untuk Opsi Deployment, pilih apakah Anda menginginkan deployment instans DB AZ-Tunggal atau Multi-AZ.

> **Note**
>
> Instans Amazon Aurora terpesan selalu memiliki opsi deployment yang ditetapkan ke Instans DB AZ-Tunggal. Namun, saat Anda membuat klaster DB Aurora, opsi deployment default adalah Buat Replika Aurora atau Pembaca di AZ (Multi-AZ) yang berbeda.
> Anda harus membeli instans DB terpesan untuk setiap instans yang Anda akan gunakan, termasuk Replika Aurora. Oleh karena itu, untuk deployment Multi-AZ di Aurora, Anda harus membeli instans DB terpesan tambahan.

7. Untuk Masa berlaku, pilih jangka waktu untuk memesan instans DB.
8. Untuk Jenis penawaran, pilih jenis penawaran.

   Setelah memilih jenis penawaran, Anda dapat melihat informasi harga.

> **Important**
>
> Pilih Batalkan untuk menghindari pembelian instans DB terpesan dan dikenakan biaya.

Setelah Anda memiliki informasi tentang penawaran instans DB terpesan yang tersedia, Anda dapat menggunakan informasi tersebut untuk membeli penawaran sebagaimana ditunjukkan dalam prosedur berikut.

Untuk membeli instans DB terpesan

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Instans terpesan.
3. Pilih Beli instans DB terpesan.
4. Untuk Deskripsi produk, pilih mesin DB dan jenis lisensi.
5. Untuk Kelas instans DB, pilih kelas instans DB.
6. Untuk Deployment Multi-AZ, pilih apakah Anda menginginkan deployment instans DB AZ-Tunggal atau Multi-AZ.

> **Note**
>
> Instans Amazon Aurora terpesan selalu memiliki opsi deployment yang ditetapkan ke Instans DB AZ-Tunggal. Saat Anda membuat klaster DB Amazon Aurora dari instans DB terpesan, klaster DB secara otomatis dibuat sebagai Multi-AZ. Pastikan untuk membeli instans DB terpesan untuk setiap instans DB yang Anda akan gunakan, termasuk Replika Aurora.

7. Untuk Masa berlaku, pilih jangka waktu yang Anda inginkan untuk pemesanan instans DB.
8. Untuk Jenis penawaran, pilih jenis penawaran.

   Setelah memilih jenis penawaran, Anda dapat melihat informasi harga.
9. (Opsional) Anda dapat menetapkan pengidentifikasi Anda sendiri ke instans DB terpesan yang Anda beli untuk membantu Anda melacaknya. Untuk ID Terpesan, ketik pengidentifikasi untuk instans DB terpesan Anda.
10. Pilih Kirim.

    Instans DB terpesan Anda dibeli, lalu ditampilkan dalam daftar Instans terpesan.

Setelah membeli instans DB terpesan, Anda dapat memperoleh informasi tentang instans DB terpesan Anda seperti yang ditunjukkan dalam prosedur berikut.

Untuk mendapatkan informasi tentang instans DB cadangan untuk akun Anda AWS

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel Navigasi, pilih Instans terpesan.

   Instans DB terpesan untuk akun Anda akan muncul. Untuk melihat informasi terperinci tentang instans DB terpesan tertentu, pilih instans tersebut dalam daftar. Kemudian, Anda dapat melihat informasi terperinci tentang instans tersebut dalam panel detail di bagian bawah konsol.

AWS CLI

Anda dapat menggunakan AWS CLI untuk bekerja dengan instans DB cadangan seperti yang ditunjukkan dalam contoh berikut.

Example mendapatkan penawaran instans DB terpesan yang tersedia

Untuk mendapatkan informasi tentang penawaran instans DB cadangan yang tersedia, hubungi perintah. AWS CLI [describe-reserved-db-instances-offerings]

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

Panggilan ini menghasilkan output serupa dengan berikut ini:

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

Setelah memiliki informasi tentang penawaran instans DB terpesan yang tersedia, Anda dapat menggunakan informasi tersebut untuk membeli penawaran.

Untuk membeli instans DB cadangan, gunakan AWS CLI perintah [purchase-reserved-db-instances-offering]dengan parameter berikut:

- `--reserved-db-instances-offering-id` – ID penawaran yang ingin Anda beli. Lihat contoh sebelumnya untuk mendapatkan ID penawaran.
- `--reserved-db-instance-id` – Anda dapat menetapkan pengindentifikasi Anda sendiri ke instans DB terpesan yang Anda beli untuk membantu melacaknya.

Example membeli instans DB terpesan

Contoh berikut membeli penawaran instans DB cadangan dengan ID *649fd0c8-cf6d-47a0-bfa6-060f8e75e95f*, dan menetapkan pengenal. *MyReservation*

Untuk Linux, macOS, atau Unix:

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

Untuk Windows:

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

Perintah tersebut mengembalikan output serupa dengan berikut ini:

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

Setelah membeli instans DB terpesan, Anda dapat memperoleh informasi tentang instans DB terpesan Anda.

Untuk mendapatkan informasi tentang instans DB cadangan untuk AWS akun Anda, hubungi AWS CLI perintah `describe-reserved-db-instances`, seperti yang ditunjukkan pada contoh berikut.

Example mendapatkan instans DB terpesan Anda

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

Perintah tersebut mengembalikan output serupa dengan berikut ini:

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

API RDS

Anda dapat menggunakan API RDS untuk bekerja dengan instans DB terpesan:

- Untuk mendapatkan informasi tentang penawaran instans DB terpesan yang tersedia, panggil operasi API Amazon RDS `DescribeReservedDBInstancesOfferings`.
- Setelah memiliki informasi tentang penawaran instans DB terpesan yang tersedia, Anda dapat menggunakan informasi tersebut untuk membeli penawaran. Panggil operasi API RDS `PurchaseReservedDBInstancesOffering` dengan parameter berikut:
  - `--reserved-db-instances-offering-id` – ID penawaran yang ingin Anda beli.
  - `--reserved-db-instance-id` – Anda dapat menetapkan pengindentifikasi Anda sendiri ke instans DB terpesan yang Anda beli untuk membantu melacaknya.
- Setelah membeli instans DB terpesan, Anda dapat memperoleh informasi tentang instans DB terpesan Anda. Panggil operasi API RDS `DescribeReservedDBInstances`.

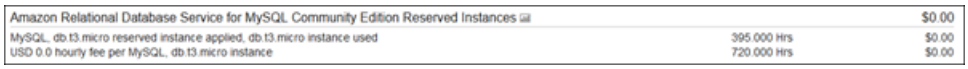
Melihat penagihan untuk instans DB terpesan Anda

Anda dapat melihat penagihan untuk instans DB terpesan Anda di Dasbor Penagihan di AWS Management Console.

Untuk melihat penagihan instans DB terpesan

1. Masuk ke AWS Management Console.
2. Dari menu akun di kanan atas, pilih Dasbor Penagihan.
3. Pilih Detail Tagihan di kanan atas dasbor.
4. Di bagian Biaya Layanan AWS , perluas Layanan Basis Data Relasional.
5. Perluas Wilayah AWS lokasi instans DB cadangan Anda, misalnya US West (Oregon).

   Instans DB terpesan Anda dan biaya per jamnya untuk bulan berjalan ditampilkan di bagian Layanan Basis Data Relasional Amazon untuk Instans Terpesan ***Mesin Basis Data***.

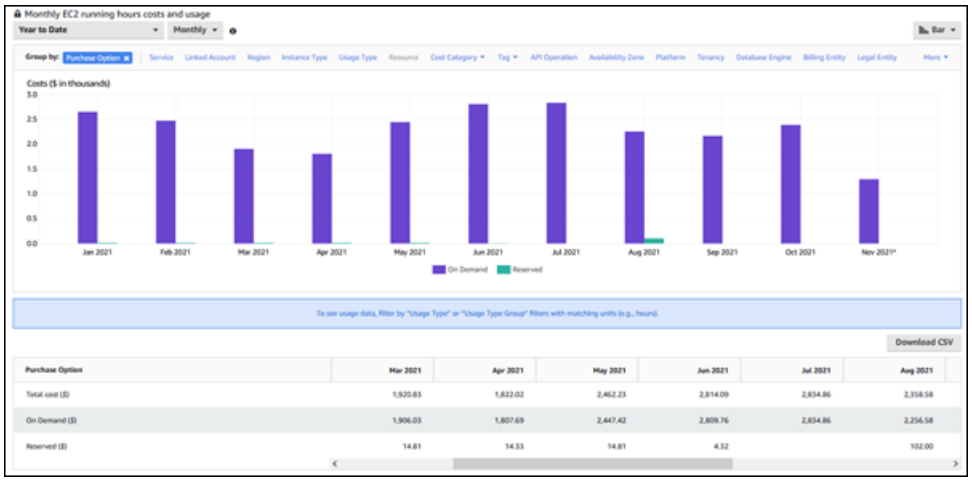
   

   Instans DB terpesan dalam contoh ini dibeli dengan opsi Uang Muka Penuh, jadi tidak ada biaya per jam.
6. Pilih ikon Penjelajah Biaya (grafik batang) di samping judul Instans Terpesan.

Penjelajah Biaya menampilkan grafik Penggunaan dan biaya waktu pengoperasian EC2 bulanan.

7. Kosongkan filter Grup Jenis Penggunaan di sebelah kanan grafik.
8. Pilih periode waktu dan unit waktu yang Anda inginkan untuk memeriksa biaya penggunaan.

Contoh berikut menunjukkan biaya penggunaan untuk instans DB sesuai permintaan dan terpesan untuk tahun berjalan menurut bulan.



Biaya instans DB terpesan dari Januari hingga Juni 2021 adalah biaya bulanan untuk instans Uang Muka Sebagian, sedangkan biaya pada Agustus 2021 adalah biaya satu kali untuk instans Uang Muka Penuh.

Diskon instans terpesan untuk instans Uang Muka Sebagian habis masa berlakunya pada Juni 2021, tetapi instans DB tidak dihapus. Setelah habis masa berlakunya, biaya dibebankan dengan tarif sesuai permintaan.

Menyiapkan lingkungan Anda untuk Amazon Aurora

Sebelum Anda menggunakan Amazon Aurora untuk pertama kali, selesaikan tugas-tugas berikut.

Topik

- Mendaftar Akun AWS
- Membuat pengguna administratif
- Memberikan akses terprogram
- Menentukan persyaratan
- Berikan akses ke klaster DB dalam VPC dengan membuat grup keamanan

Jika sudah memiliki Akun AWS, mengetahui persyaratan Aurora Anda, dan lebih memilih menggunakan default untuk grup keamanan IAM dan VPC, Anda dapat langsung beralih ke Memulai dengan Amazon Aurora.

Mendaftar Akun AWS

Jika Anda tidak memiliki Akun AWS, selesaikan langkah-langkah berikut untuk membuatnya.

Untuk mendaftar Akun AWS

1. Buka https://portal.aws.amazon.com/billing/signup.
2. Ikuti petunjuk secara online.

   Anda akan diminta untuk menerima panggilan telepon dan memasukkan kode verifikasi pada keypad telepon sebagai bagian dari prosedur pendaftaran.

   Saat Anda mendaftar Akun AWS, Pengguna root akun AWS akan dibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya dalam akun. Sebagai praktik terbaik keamanan, tetapkan akses administratif ke pengguna administratif, dan hanya gunakan pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

AWS akan mengirimkan email konfirmasi kepada Anda setelah proses pendaftaran selesai. Anda dapat melihat aktivitas akun saat ini dan mengelola akun dengan mengunjungi https://aws.amazon.com/ dan memilih Akun Saya.

Membuat pengguna administratif

Setelah mendaftar Akun AWS, amankan Pengguna root akun AWS, aktifkan AWS IAM Identity Center, dan buat sebuah pengguna administratif sehingga Anda tidak menggunakan pengguna root untuk tugas sehari-hari.

Mengamankan Pengguna root akun AWS Anda

1. Masuk ke AWS Management Console sebagai pemilik akun dengan memilih Pengguna root dan memasukkan alamat email Akun AWS Anda. Di halaman berikutnya, masukkan kata sandi Anda.

   Untuk bantuan masuk menggunakan pengguna root, lihat Masuk sebagai pengguna root dalam Panduan Pengguna AWS Sign-In.
2. Aktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

   Untuk petunjuknya, silakan lihat Mengaktifkan perangkat MFA virtual untuk pengguna root Akun AWS Anda (konsol) dalam Panduan Pengguna IAM.

Membuat pengguna administratif

1. Aktifkan Pusat Identitas IAM.

   Untuk mendapatkan petunjuk, silakan lihat Mengaktifkan AWS IAM Identity Center di Panduan Pengguna AWS IAM Identity Center.
2. Di Pusat Identitas IAM, berikan akses administratif ke sebuah pengguna administratif.

   Untuk mendapatkan tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, silakan lihat Mengonfigurasi akses pengguna dengan Direktori Pusat Identitas IAM default di Panduan Pengguna AWS IAM Identity Center.

Masuk sebagai pengguna administratif

- Untuk masuk dengan pengguna Pusat Identitas IAM, gunakan URL masuk yang dikirim ke alamat email Anda saat Anda membuat pengguna Pusat Identitas IAM.

  Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat Masuk ke portal akses AWS dalam Panduan Pengguna AWS Sign-In.

Memberikan akses terprogram

Pengguna membutuhkan akses terprogram jika mereka ingin berinteraksi dengan AWS luar dari AWS Management Console. Cara memberikan akses terprogram bergantung pada jenis pengguna yang mengakses AWS.

Untuk memberi pengguna akses terprogram, pilih salah satu opsi berikut.

Pengguna mana yang membutuhkan akses terprogram?	Untuk	Oleh
Identitas tenaga kerja (Pengguna yang dikelola di Pusat Identitas IAM)	Gunakan kredensial sementara untuk menandatangani permintaan terprogram ke AWS CLI, SDK AWS, atau API AWS.	Mengikuti petunjuk untuk antarmuka yang ingin Anda gunakan. • Untuk AWS CLI, lihat Mengonfigurasi AWS CLI untuk menggunakan AWS IAM Identity Center di Panduan Pengguna AWS Command Line Interface. • Untuk SDK AWS, alat, dan API AWS, lihat Autentikasi Pusat Identitas IAM di Panduan Referensi SDK dan Alat AWS.
IAM	Gunakan kredensial sementara untuk menandatangani permintaan terprogram ke AWS CLI, SDK AWS, atau API AWS.	Mengikuti petunjuk dalam Menggunakan kredensial sementara dengan sumber daya AWS di Panduan Pengguna IAM.
IAM	(Tidak direkomendasikan) Gunakan kredensial jangka panjang untuk menandatangani permintaan terprogram	Mengikuti petunjuk untuk antarmuka yang ingin Anda gunakan.

Pengguna mana yang membutuhkan akses terprogram?	Untuk	Oleh
	ke AWS CLI, SDK AWS, atau API AWS.	• Untuk AWS CLI, lihat Mengautentikasi menggunakan kredensial pengguna IAM di Panduan Pengguna AWS Command Line Interface. • Untuk SDK dan alat AWS, lihat Mengautentikasi menggunakan kredensial jangka panjang di Panduan Referensi SDK dan Alat AWS. • Untuk API AWS, lihat Mengelola kunci akses untuk pengguna IAM di Panduan Pengguna IAM.

Menentukan persyaratan

Blok bangunan dasar Aurora adalah klaster DB. Klaster DB dapat berisi satu atau beberapa instans DB. Klaster DB menyediakan alamat jaringan yang disebut titik akhir klaster. Aplikasi Anda terhubung ke titik akhir klaster yang dibuka oleh klaster DB kapan pun aplikasi tersebut perlu mengakses basis data yang dibuat dalam klaster DB tersebut. Informasi yang Anda tentukan saat membuat elemen konfigurasi kontrol klaster DB seperti memori, mesin dan versi basis data, konfigurasi jaringan, keamanan, serta periode pemeliharaan.

Sebelum membuat grup keamanan dan klaster DB, Anda harus mengetahui klaster DB dan kebutuhan jaringan Anda. Berikut beberapa hal penting yang perlu dipertimbangkan:

- Kebutuhan sumber daya – Apa saja kebutuhan memori dan prosesor untuk aplikasi atau layanan Anda? Anda akan menggunakan pengaturan ini saat menentukan kelas instans DB yang akan

Anda gunakan saat membuat klaster DB. Untuk spesifikasi tentang kelas instans DB, lihat Kelas instans DB Aurora.

- VPC, subnet, dan grup keamanan – Klaster DB Anda akan berada di cloud privat virtual (VPC). Aturan grup keamanan harus dikonfigurasi agar terhubung ke klaster DB. Daftar berikut menjelaskan aturan untuk setiap opsi VPC:
  - VPC Default – Jika akun AWS Anda memiliki VPC default di Wilayah AWS, VPC tersebut dikonfigurasi untuk mendukung klaster DB. Jika Anda menentukan VPC default saat membuat klaster DB:
    - Anda harus membuat grup keamanan VPC yang mengizinkan koneksi dari aplikasi atau layanan ke klaster DB Aurora. Gunakan opsi Grup Keamanan pada konsol VPC atau AWS CLI untuk membuat grup keamanan VPC. Untuk informasi, lihat Langkah 3: Buat grup keamanan VPC.
    - Anda harus menentukan grup subnet DB default. Jika ini adalah klaster DB pertama yang Anda buat di Wilayah AWS, Amazon RDS akan membuat grup subnet DB default saat membuat klaster DB.
  - VPC yang ditentukan pengguna – Jika Anda ingin menentukan VPC yang ditentukan pengguna ketika membuat klaster DB:
    - Anda harus membuat grup keamanan VPC yang mengizinkan koneksi dari aplikasi atau layanan ke klaster DB Aurora. Gunakan opsi Grup Keamanan pada konsol VPC atau AWS CLI untuk membuat grup keamanan VPC. Untuk informasi, lihat Langkah 3: Buat grup keamanan VPC.
    - VPC harus memenuhi persyaratan tertentu untuk meng-hosting klaster DB, seperti memiliki setidaknya dua subnet, yang masing-masingnya berada di zona ketersediaan terpisah. Untuk informasi, lihat Amazon VPC dan Amazon Aurora.
    - Anda harus menentukan grup subnet DB yang menentukan subnet mana di VPC tersebut yang dapat digunakan oleh klaster DB. Untuk informasi, lihat bagian Grup Subnet DB di Bekerja dengan instans DB dalam VPC.
- Ketersediaan tinggi: Apakah Anda memerlukan dukungan failover? Di Aurora, deployment Multi-AZ menciptakan instans primer dan Replika Aurora. Anda dapat mengonfigurasi instans primer dan Replika Aurora agar berada di Zona Ketersediaan yang berbeda untuk dukungan failover. Kami merekomendasikan deployment Multi-AZ untuk beban kerja produksi agar menjaga ketersediaan yang tinggi. Untuk tujuan pengembangan dan pengujian, Anda dapat menggunakan deployment Multi-AZ. Untuk informasi selengkapnya, lihat Ketersediaan yang tinggi untuk Amazon Aurora.

- Kebijakan IAM: Apakah akun AWS Anda memiliki kebijakan yang memberikan izin yang diperlukan untuk melakukan operasi Amazon RDS? Jika Anda tersambung ke AWS menggunakan kredensial IAM, akun IAM Anda harus memiliki kebijakan IAM yang memberikan izin yang diperlukan untuk menjalankan operasi Amazon RDS. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Amazon Aurora.
- Port terbuka: Port TCP/IP apa yang akan didengar oleh basis data Anda? Firewall di beberapa perusahaan mungkin memblokir koneksi ke port default untuk mesin basis data Anda. Jika firewall perusahaan Anda memblokir port default, pilih port lain untuk klaster DB baru. Perhatikan bahwa setelah membuat klaster DB yang mendengarkan di port yang Anda tentukan, Anda dapat mengubah port tersebut dengan mengubah klaster DB.
- Wilayah AWS: Wilayah AWS mana yang Anda inginkan untuk basis data Anda? Memiliki basis data yang dekat dengan aplikasi atau layanan web dapat mengurangi latensi jaringan. Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan.

Setelah mendapatkan informasi yang Anda perlukan untuk membuat grup keamanan dan klaster DB, lanjutkan ke langkah berikutnya.

Berikan akses ke klaster DB dalam VPC dengan membuat grup keamanan

Klaster DB Anda akan dibuat di VPC. Grup keamanan memberikan akses ke klaster DB dalam VPC. Grup tersebut bertindak sebagai firewall untuk klaster DB terkait, yang mengontrol lalu lintas masuk dan keluar pada tingkat klaster. Klaster DB dibuat secara default dengan satu firewall dan grup keamanan default yang mencegah akses ke klaster DB. Oleh karena itu, Anda harus menambahkan aturan ke grup keamanan yang memungkinkan Anda untuk terhubung ke klaster DB Anda. Gunakan informasi jaringan dan konfigurasi yang Anda tentukan pada langkah sebelumnya untuk membuat aturan yang memungkinkan akses ke klaster DB Anda.

Misalnya, jika memiliki aplikasi yang akan mengakses basis data pada klaster DB Anda dalam VPC, Anda harus menambahkan aturan TCP khusus yang menentukan rentang port dan alamat IP yang akan digunakan aplikasi untuk mengakses basis data. Jika memiliki aplikasi di instans Amazon EC2, Anda dapat menggunakan grup keamanan VPC yang Anda siapkan untuk instans Amazon EC2.

Anda dapat mengonfigurasi konektivitas antara instans Amazon EC2 dan kluster DB saat membuat klaster DB. Untuk informasi selengkapnya, lihat Konfigurasikan konektivitas jaringan otomatis dengan instans EC2.

> **Tip**
>
> Anda dapat menyiapkan konektivitas jaringan antara instans Amazon EC2 dan klaster DB secara otomatis saat membuat klaster DB. Untuk informasi selengkapnya, lihat Konfigurasikan konektivitas jaringan otomatis dengan instans EC2.

Untuk informasi selengkapnya tentang membuat VPC untuk digunakan dengan Aurora, lihat Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4). Untuk informasi tentang skenario umum untuk mengakses instans DB, lihat Skenario untuk mengakses klaster DB di VPC.

Untuk membuat grup keamanan VPC

1. Masuk ke AWS Management Console dan buka konsol Amazon VPC di https://console.aws.amazon.com/vpc.

   > **Note**
   >
   > Pastikan Anda berada di konsol VPC, bukan konsol RDS.

2. Di pojok kanan atas AWS Management Console, pilih Wilayah AWS tempat Anda ingin membuat grup keamanan VPC dan klaster DB Anda. Dalam daftar sumber daya Amazon VPC untuk Wilayah AWS tersebut, Anda akan melihat setidaknya satu VPC dan beberapa subnet. Jika tidak, Anda tidak memiliki VPC default di Wilayah AWS tersebut.
3. Di panel navigasi, pilih Security Groups (Grup Keamanan).
4. Pilih Buat grup keamanan.

   Halaman Membuat grup keamanan akan muncul.
5. Dalam Detail basic, masukkan Nama grup keamanan dan Deskripsi. Untuk VPC, pilih VPC tempat Anda ingin membuat klaster DB.
6. Di bagian Aturan masuk, pilih Tambahkan aturan.

   a. Untuk Tipe, pilih TCP khusus.

   b. Untuk Rentang port, ketik nilai port yang akan digunakan untuk klaster DB Anda.

   c. Untuk Sumber, pilih nama grup keamanan atau ketik rentang alamat IP (nilai CIDR) dari tempat Anda mengakses klaster DB. Jika Anda memilih IP Saya, pilihan ini mengizinkan akses ke klaster DB dari alamat IP yang terdeteksi di browser Anda.

7. Jika Anda perlu menambahkan lebih banyak alamat IP atau rentang port yang berbeda, pilih Tambahkan aturan dan masukkan informasi untuk aturan tersebut.
8. (Opsional) Dalam Aturan keluar, tambahkan aturan untuk lalu lintas keluar. Secara default, semua lalu lintas keluar akan diizinkan.
9. Pilih Buat grup keamanan.

Anda dapat menggunakan grup keamanan VPC yang baru Anda buat sebagai grup keamanan untuk klaster DB Anda saat membuatnya.

> **Note**
>
> Jika Anda menggunakan VPC default, grup subnet default yang mencakup semua subnet VPC akan dibuat untuk Anda. Saat membuat klaster DB, Anda dapat memilih VPC default dan menggunakan default untuk Grup Subnet DB.

Setelah menyelesaikan persyaratan pengaturan, Anda dapat membuat klaster DB menggunakan persyaratan dan grup keamanan Anda dengan mengikuti petunjuk di Membuat klaster DB Amazon Aurora. Untuk informasi tentang memulai dengan membuat klaster DB yang menggunakan mesin DB tertentu, lihat Memulai dengan Amazon Aurora.

Memulai dengan Amazon Aurora

Pada bagian ini, Anda dapat mengetahui cara membuat dan terhubung ke klaster Aurora DB menggunakan Amazon RDS.

Prosedur berikut adalah tutorial yang menunjukkan dasar-dasar memulai Aurora. Bagian selanjutnya memperkenalkan konsep dan prosedur Aurora lanjutan, seperti berbagai jenis titik akhir dan cara menskalakan klaster Aurora ke atas dan ke bawah.

> **Important**
>
> Sebelum dapat membuat atau terhubung ke klaster DB, pastikan untuk menyelesaikan tugas dalam Menyiapkan lingkungan Anda untuk Amazon Aurora.

Topik

- Membuat dan terhubung ke klaster DB Aurora MySQL
- Membuat dan terhubung ke klaster DB Aurora PostgreSQL
- Tutorial: Membuat server web dan klaster DB Amazon Aurora

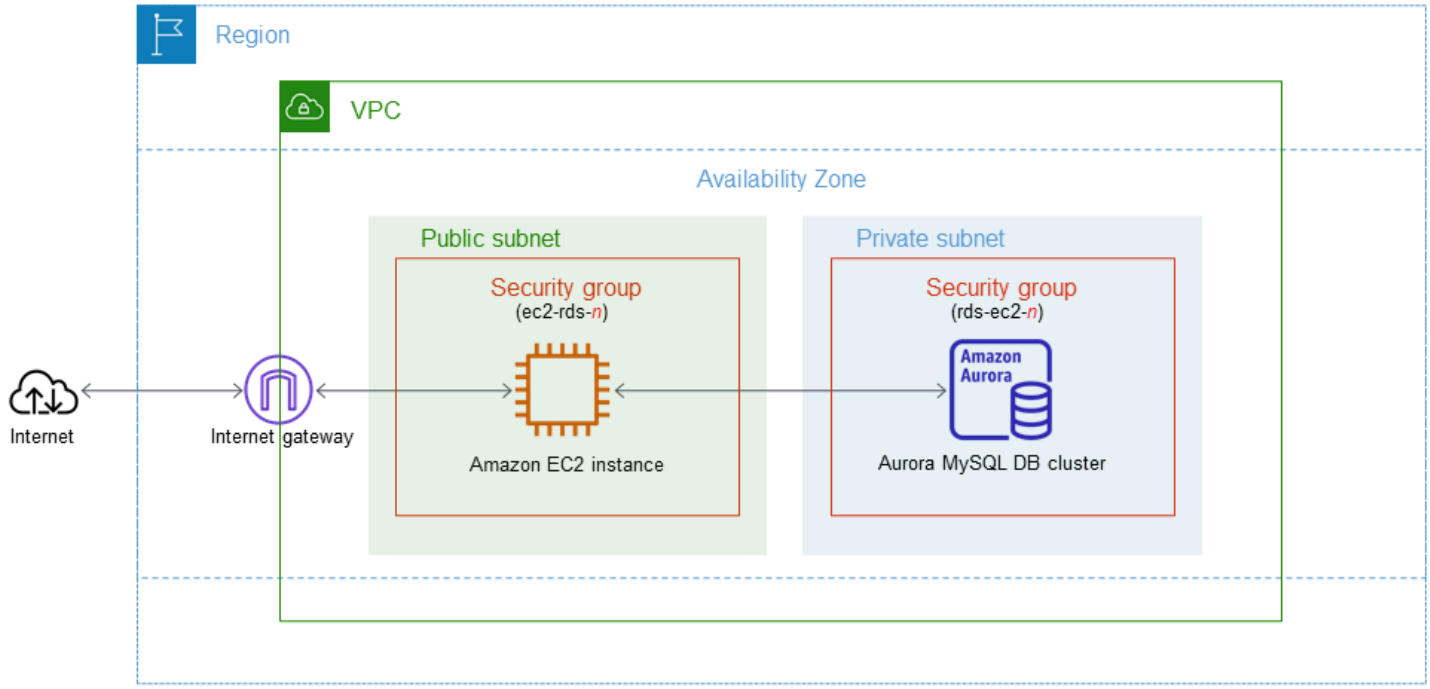
Membuat dan terhubung ke klaster DB Aurora MySQL

Tutorial ini membuat instans EC2 dan klaster DB Aurora MySQL. Tutorial ini menunjukkan cara mengakses klaster DB dari instans EC2 menggunakan klien MySQL standar. Sebagai praktik terbaik, tutorial ini membuat klaster DB privat dalam cloud privat virtual (VPC). Dalam kebanyakan kasus, sumber daya lain dalam VPC yang sama, seperti instans EC2, dapat mengakses klaster DB, tetapi sumber daya di luar VPC tidak dapat mengaksesnya.

Setelah Anda menyelesaikan tutorial, ada subnet publik dan privat di setiap Zona Ketersediaan di VPC Anda. Dalam satu Zona Ketersediaan, instans EC2 berada di subnet publik, dan instans DB berada di subnet privat.

> **Important**
>
> Tidak ada biaya untuk membuat AWS akun. Namun, dengan menyelesaikan tutorial ini, Anda mungkin dikenakan biaya untuk AWS sumber daya yang Anda gunakan. Anda dapat menghapus sumber daya ini setelah menyelesaikan tutorial jika tidak diperlukan lagi.

Diagram berikut menunjukkan konfigurasi setelah tutorial selesai.



Tutorial ini memungkinkan Anda untuk membuat sumber daya Anda dengan menggunakan salah satu metode berikut:

1. Gunakan AWS Management Console - Langkah 1: Buat instans EC2 dan Langkah 2: Membuat klaster DB Aurora MySQL
2. Gunakan AWS CloudFormation untuk membuat instance database dan instans EC2 - (Opsional) Buat VPC, instans EC2, dan cluster Aurora MySQL menggunakan AWS CloudFormation

Metode pertama menggunakan Easy create untuk membuat cluster Aurora MySQL DB pribadi dengan. AWS Management Console Di sini, Anda hanya menentukan jenis mesin DB, ukuran instans DB, dan pengidentifikasi cluster DB. Pembuatan Mudah menggunakan pengaturan default untuk opsi konfigurasi lainnya.

Saat Anda menggunakan Standard create sebagai gantinya, Anda dapat menentukan lebih banyak opsi konfigurasi saat membuat cluster DB. Opsi ini mencakup pengaturan untuk ketersediaan, keamanan, cadangan, dan pemeliharaan. Untuk membuat klaster DB publik, Anda harus menggunakan Pembuatan standar. Untuk informasi, lihat the section called “Membuat klaster DB”.

Topik

- Prasyarat
- Langkah 1: Buat instans EC2
- Langkah 2: Membuat klaster DB Aurora MySQL
- (Opsional) Buat VPC, instans EC2, dan cluster Aurora MySQL menggunakan AWS CloudFormation
- Langkah 3: Membuat klaster DB Aurora MySQL
- Langkah 4: Hapus instans EC2 dan klaster DB
- (Opsional) Hapus instans EC2 dan cluster DB yang dibuat dengan CloudFormation
- (Opsional) Menghubungkan klaster DB Anda ke fungsi Lambda

Prasyarat

Sebelum memulai, selesaikan langkah-langkah di bagian berikut:

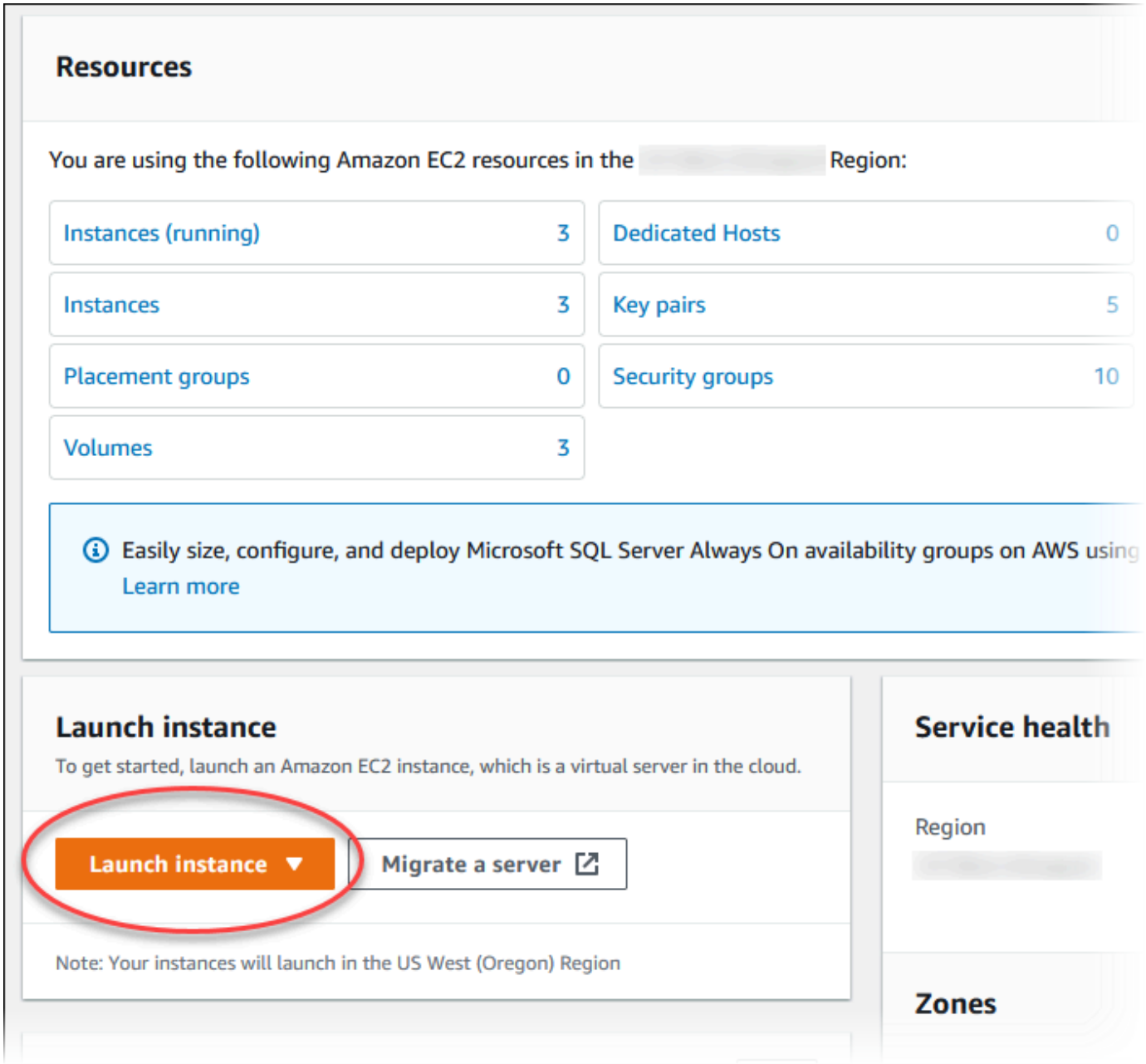
- Mendaftar Akun AWS
- Membuat pengguna administratif

Langkah 1: Buat instans EC2

Buat instans Amazon EC2 yang akan Anda gunakan untuk menghubungkan ke basis data Anda.

Untuk membuat instans EC2

1. Masuk ke AWS Management Console dan buka konsol Amazon EC2 di https://console.aws.amazon.com/ec2/.
2. Di sudut kanan atas AWS Management Console, pilih Wilayah AWS di mana Anda ingin membuat instans EC2.
3. Pilih Dasbor EC2, lalu pilih Luncurkan instans seperti yang ditampilkan dalam gambar berikut.

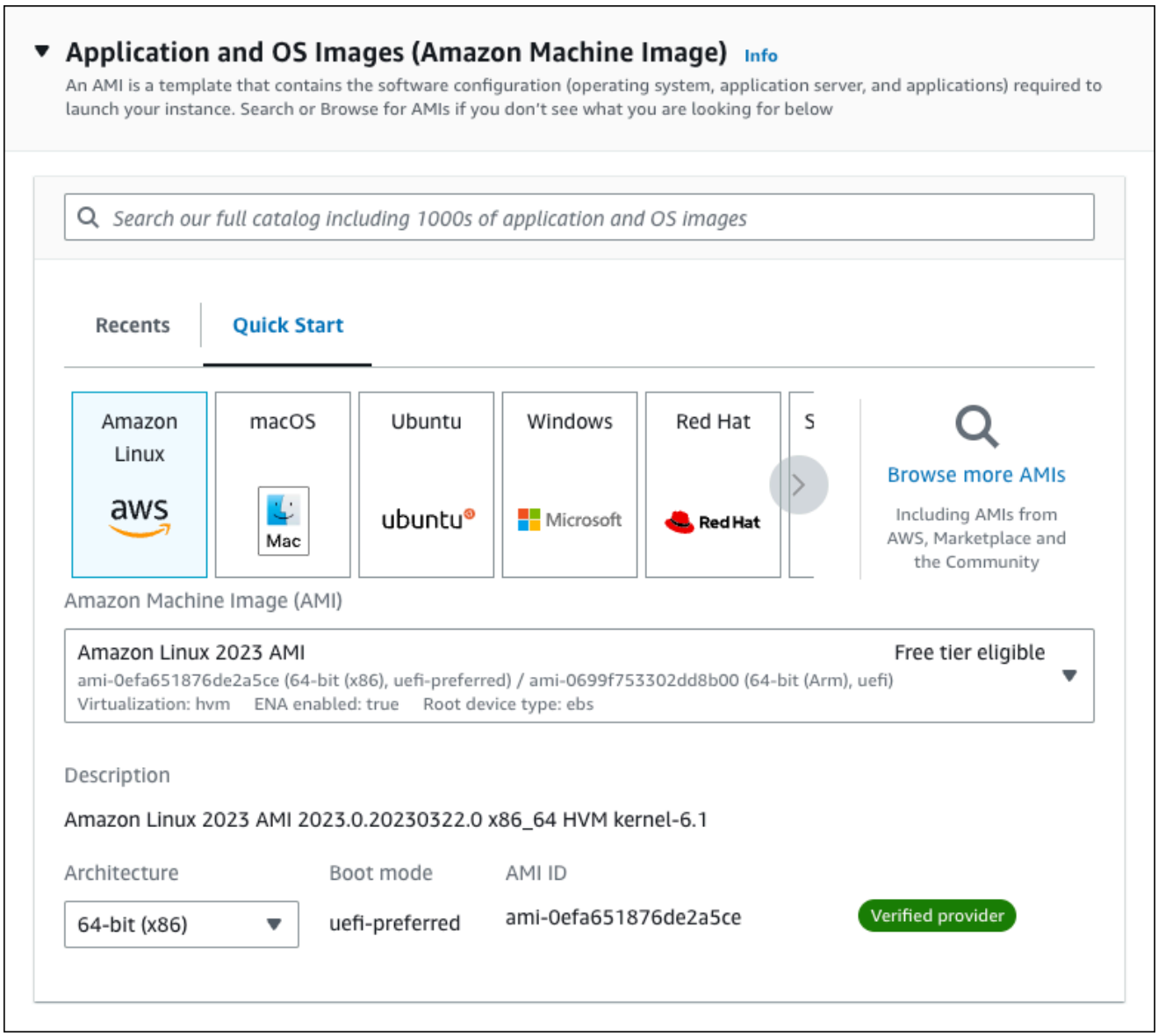


Halaman Meluncurkan instans akan terbuka.

4. Pilih pengaturan berikut di halaman Meluncurkan instans.

   a. Di bagian Nama dan tag, untuk Nama, masukkan **`ec2-database-connect`**.

   b. Di bagian Gambar Aplikasi dan OS (Amazon Machine Image), pilih Amazon Linux, lalu pilih AMI Amazon Linux 2023. Biarkan default untuk pilihan lainnya.



c. Di bagian Jenis instans, pilih t2.micro.

d. Di bagian Pasangan kunci (login), pilih nama Pasangan kunci untuk menggunakan pasangan kunci yang ada. Untuk membuat pasangan kunci baru untuk instans Amazon EC2, pilih Buat Pasangan kunci baru lalu gunakan jendela Buat pasangan kunci untuk membuatnya.

   Untuk informasi selengkapnya tentang membuat pasangan kunci baru, lihat Membuat pasangan kunci di Panduan Pengguna Amazon EC2 untuk Instans Linux.

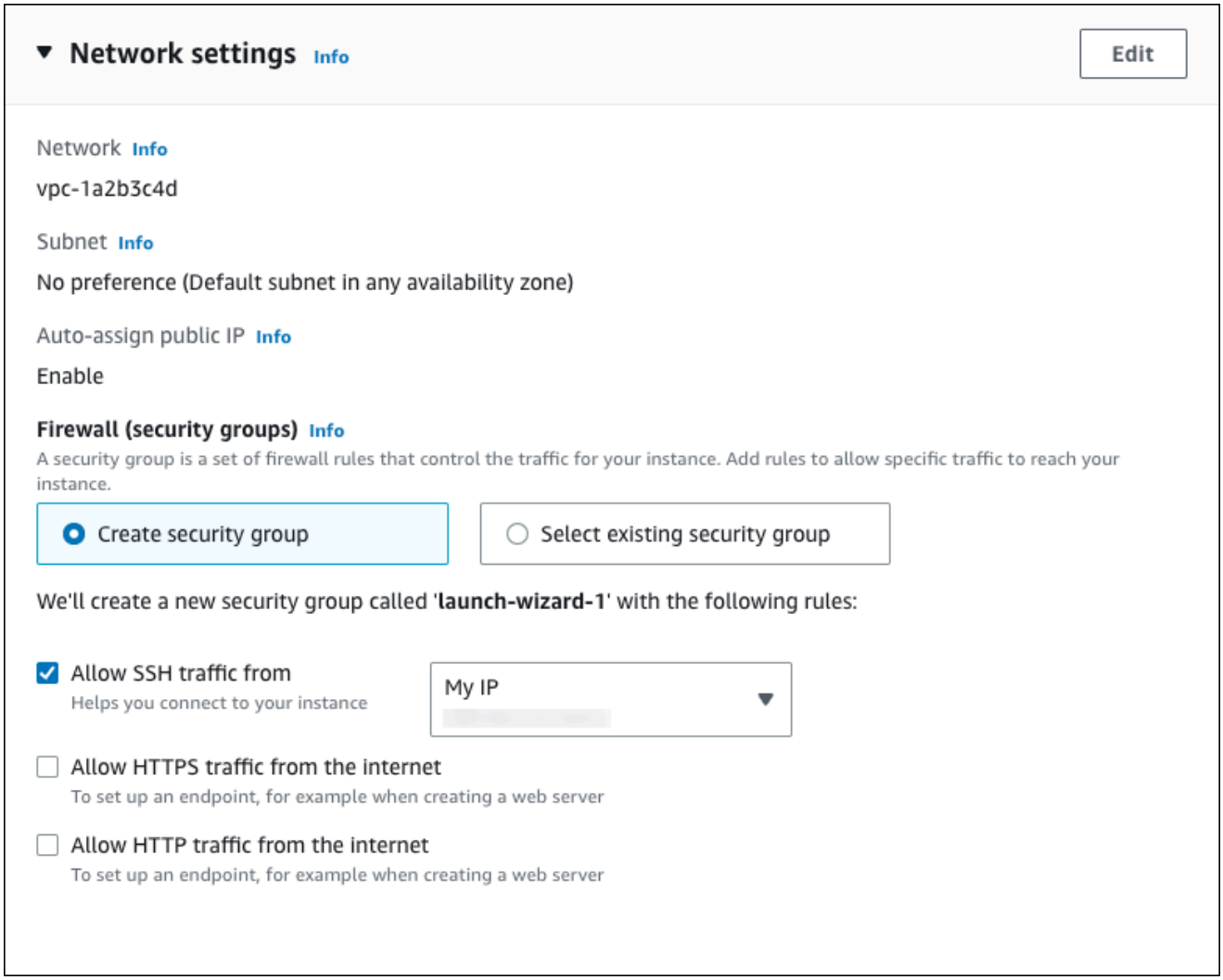
e. Untuk Izinkan lalu lintas SSH di Pengaturan jaringan, pilih sumber koneksi SSH ke instans EC2.

Anda dapat memilih IP Saya jika alamat IP yang ditampilkan benar untuk koneksi SSH. Jika tidak, Anda dapat menentukan alamat IP yang akan digunakan untuk menghubungkan ke instans EC2 di VPC Anda menggunakan Secure Shell (SSH). Untuk menentukan alamat IP publik Anda, Anda dapat membuka layanan di https://checkip.amazonaws.com di jendela atau tab browser lain. Contoh alamat IP adalah 192.0.2.1/32.

Dalam banyak kasus, Anda dapat menghubungkan melalui penyedia layanan Internet (ISP) atau dari belakang firewall Anda tanpa alamat IP statis. Jika demikian, tentukan rentang alamat IP yang digunakan oleh komputer klien.

> **Warning**
>
> Jika menggunakan `0.0.0.0/0` untuk akses SSH, Anda memungkinkan semua alamat IP untuk mengakses instans publik EC2 Anda menggunakan SSH. Hal ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, Anda hanya dapat memberikan otorisasi pada alamat IP atau rentang alamat tertentu saja untuk mengakses instans EC2 Anda menggunakan SSH.

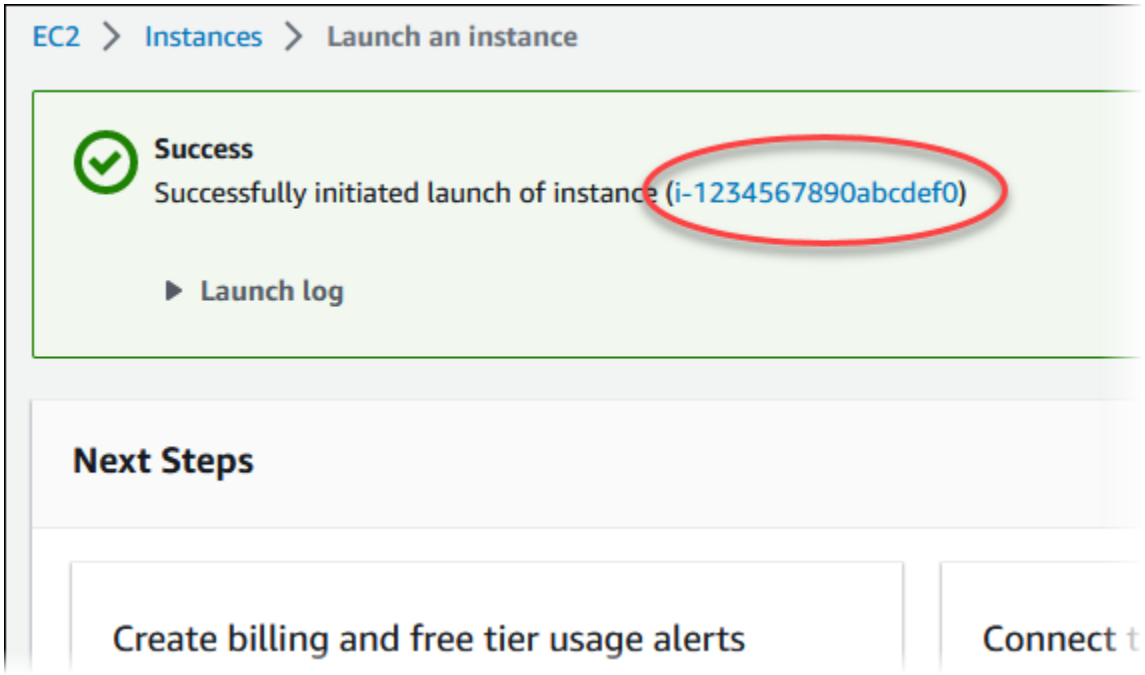
Gambar berikut menunjukkan contoh bagian Pengaturan jaringan.



f. Biarkan nilai default untuk bagian yang lainnya.

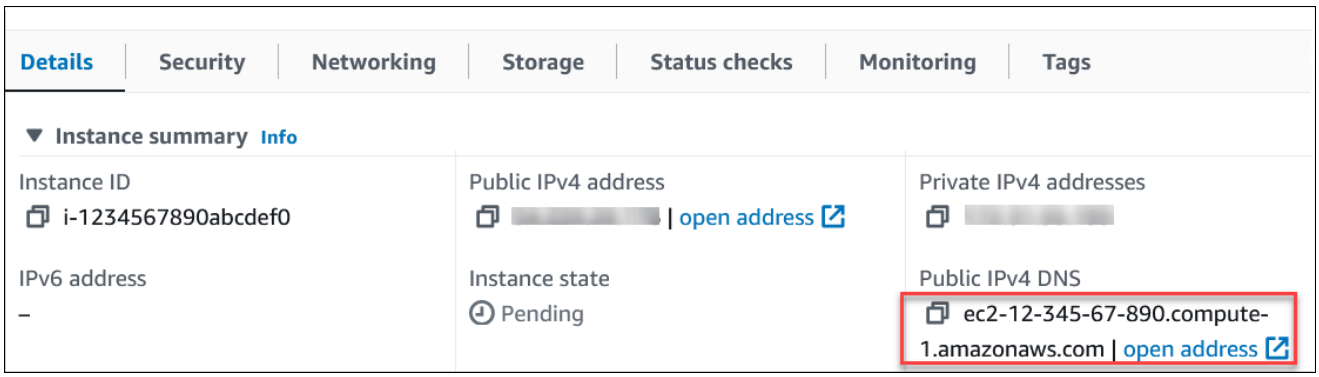
g. Tinjau ringkasan konfigurasi instans EC2 Anda di panel Ringkasan, dan setelah Anda siap, pilih Luncurkan instans.

5. Di halaman Status Peluncuran, catat pengidentifikasi untuk instans EC2 baru Anda, misalnya: `i-1234567890abcdef0`.

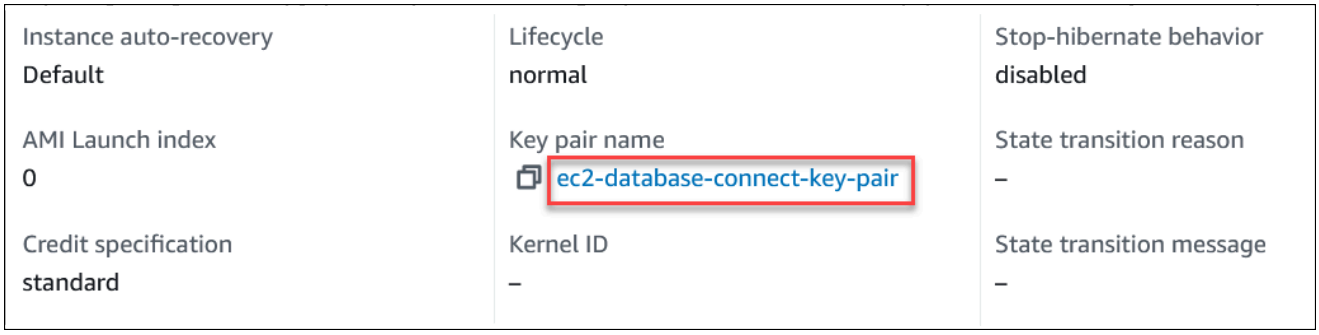


6. Pilih pengidentifikasi instans EC2 untuk membuka daftar instans EC2, lalu pilih instans EC2 Anda.
7. Di tab Detail, catat nilai-nilai berikut, yang akan Anda butuhkan saat menghubungkan menggunakan SSH:

   a. Di Ringkasan instans, catat nilai untuk DNS IPv4 Publik.



   b. Di Detail instans, catat nilai untuk Nama pasangan kunci.



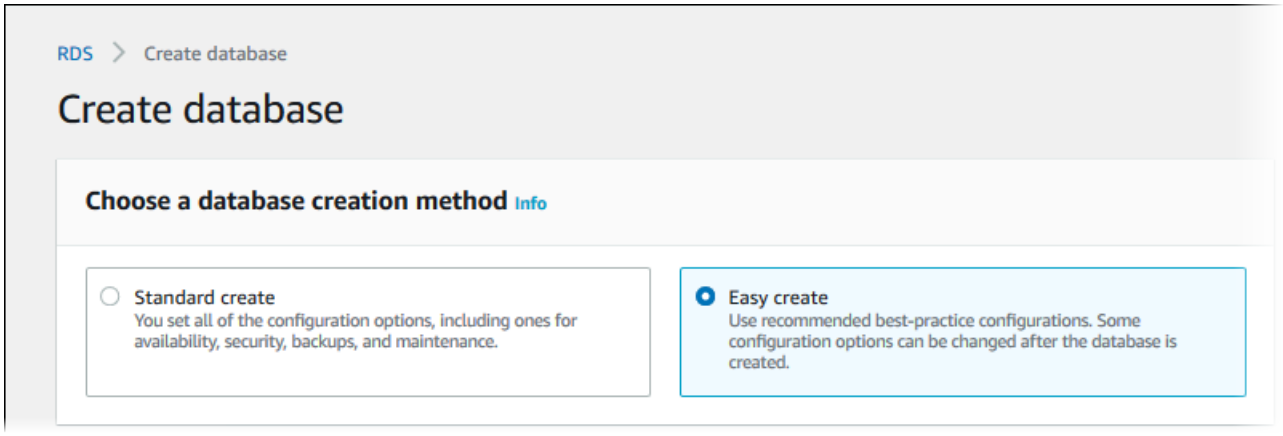
8. Tunggu hingga Status instans untuk instans EC2 Anda berstatus Berjalan sebelum melanjutkan.

Langkah 2: Membuat klaster DB Aurora MySQL

Dalam contoh ini, Anda menggunakan Pembuatan mudah untuk membuat klaster DB Aurora MySQL dengan kelas instans DB db.r6g.large.

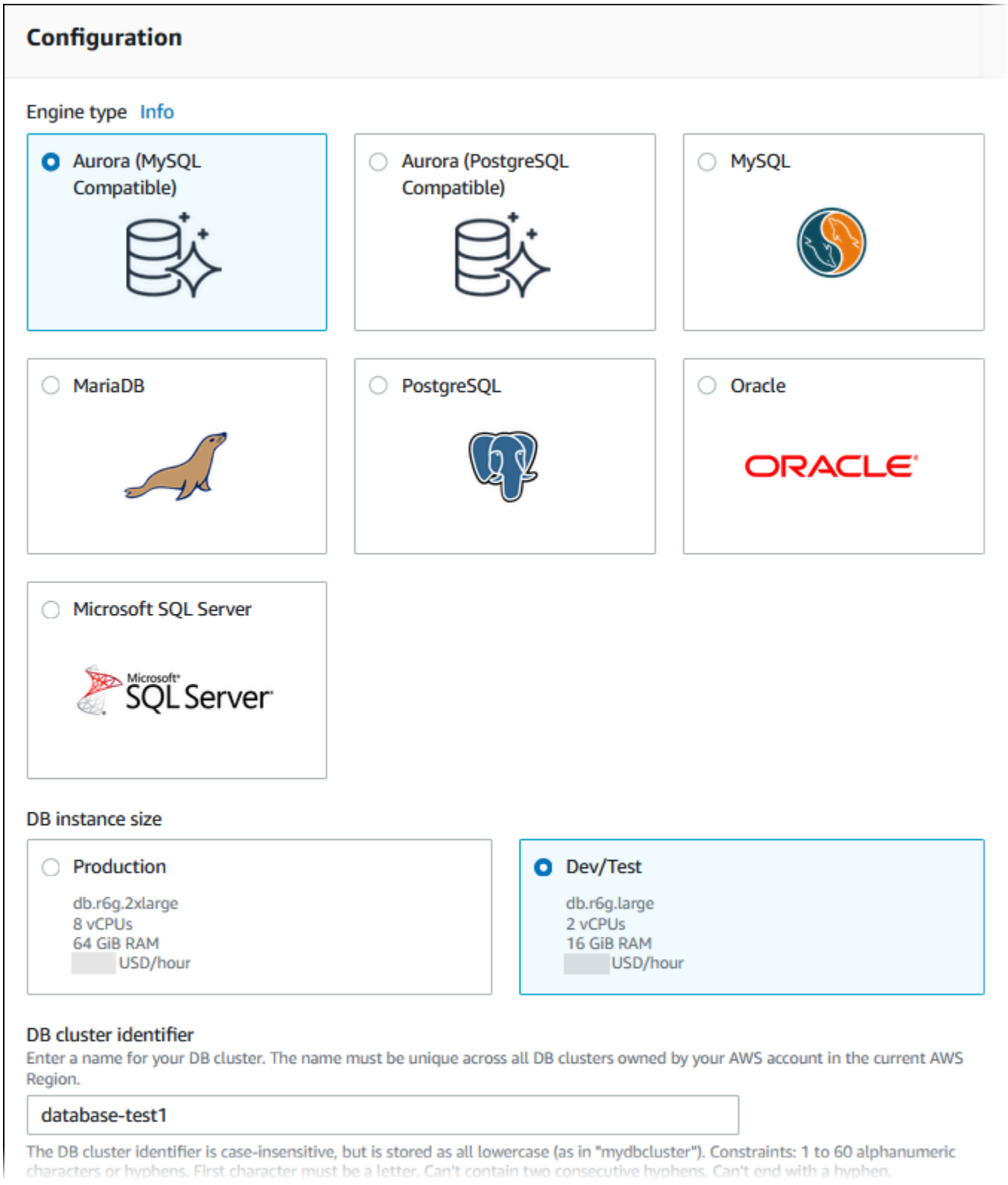
Untuk membuat klaster DB Aurora MySQL dengan Pembuatan mudah

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas konsol Amazon RDS, pilih Wilayah AWS di mana Anda ingin membuat cluster DB.
3. Di panel navigasi, pilih Basis Data.
4. Pilih Buat basis data dan pastikan Pembuatan Mudah dipilih.



5. Di Konfigurasi, pilih Aurora (Kompatibel dengan MySQL) untuk Jenis mesin.
6. Untuk Ukuran instans DB, pilih Dev/Tes.
7. Untuk pengidentifikasi klaster DB, masukkan **`database-test1`**.

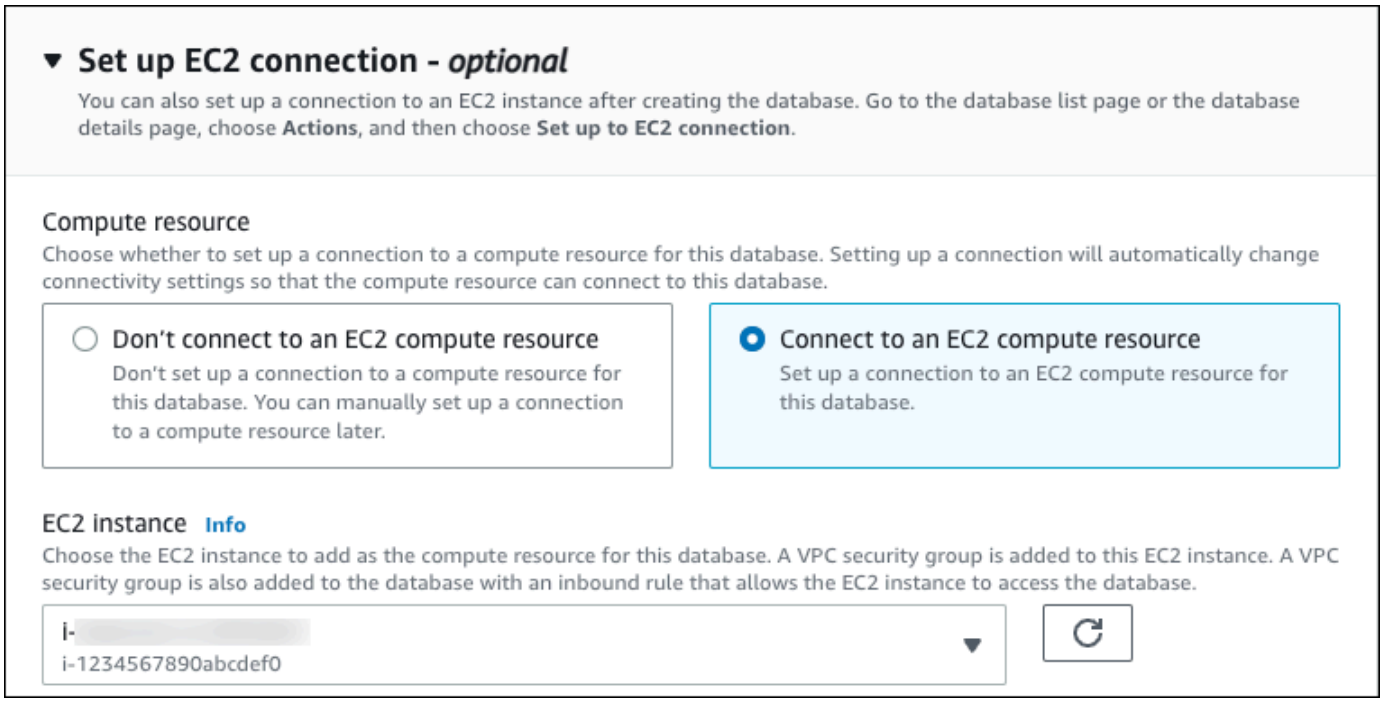
Tampilan halaman Membuat basis data seperti gambar berikut.



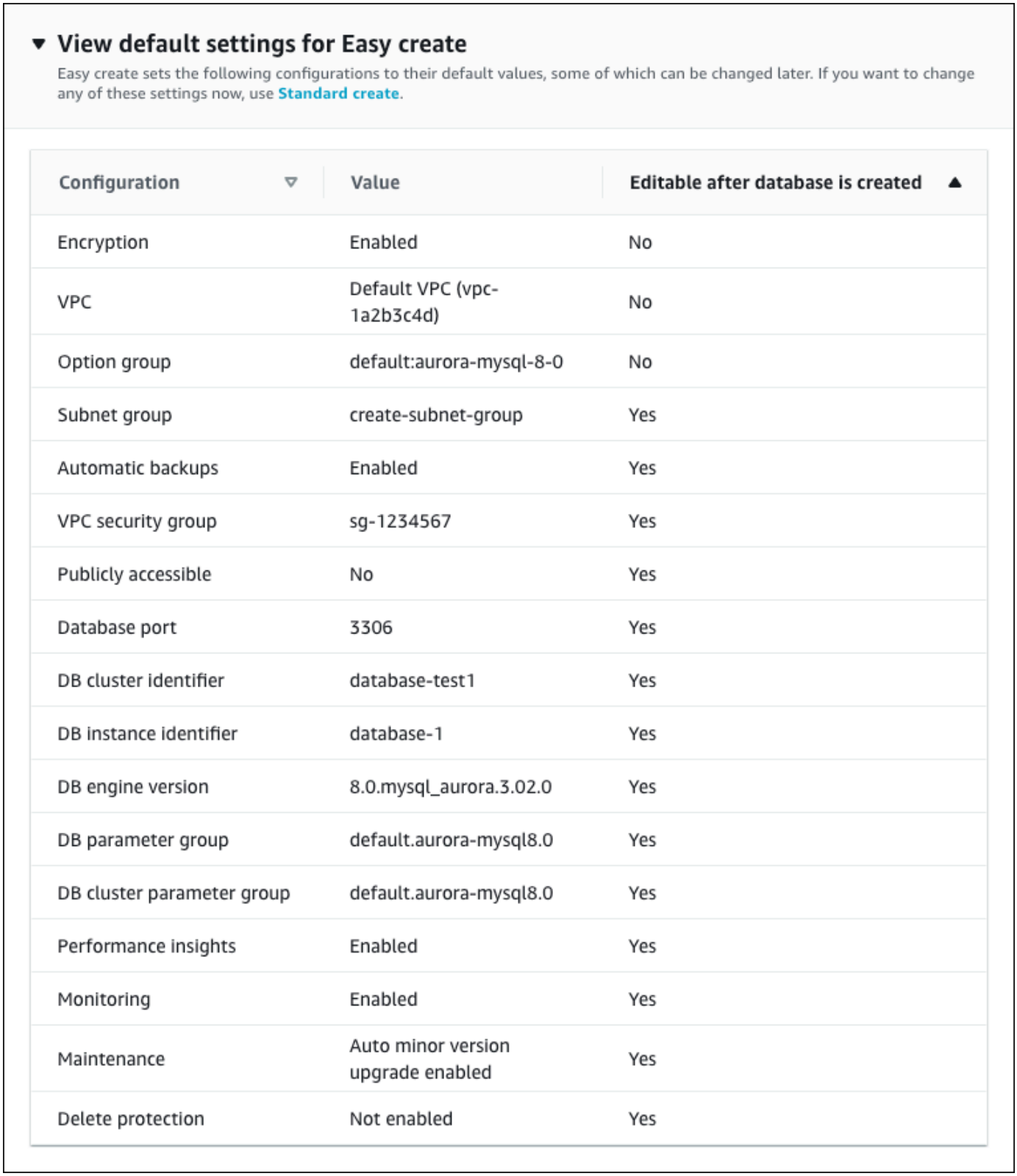
8. Untuk Nama pengguna utama, masukkan nama untuk pengguna utama, atau tetap gunakan nama default.
9. Untuk menggunakan kata sandi utama yang dibuat secara otomatis untuk klaster DB, pilih Buat kata sandi secara otomatis.

   Untuk memasukkan kata sandi utama Anda, hapus centang pada Buat kata sandi secara otomatis, lalu masukkan kata sandi yang sama dalam Kata sandi utama dan Konfirmasi kata sandi.
10. Untuk menyiapkan koneksi dengan instans EC2 yang Anda buat sebelumnya, buka Menyiapkan koneksi EC2 - opsional.

    Pilih Hubungkan ke sumber daya komputasi EC2. Pilih instans EC2 yang Anda buat sebelumnya.



11. Buka Lihat pengaturan default untuk Pembuatan Mudah.



View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:aurora-mysql-8-0	No
Subnet group	create-subnet-group	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	3306	Yes
DB cluster identifier	database-test1	Yes
DB instance identifier	database-1	Yes
DB engine version	8.0.mysql_aurora.3.02.0	Yes
DB parameter group	default.aurora-mysql8.0	Yes
DB cluster parameter group	default.aurora-mysql8.0	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

Anda dapat memeriksa pengaturan default yang digunakan dengan Pembuatan mudah. Kolom Dapat diedit setelah basis data dibuat menunjukkan opsi yang dapat Anda ubah setelah membuat basis data.

- Jika pengaturan memiliki Tidak di kolom tersebut, dan Anda menginginkan pengaturan yang berbeda, Anda dapat menggunakan Pembuatan standar untuk membuat klaster DB.
- Jika pengaturan memiliki Ya di kolom tersebut, dan Anda menginginkan pengaturan yang berbeda, Anda dapat menggunakan Pembuatan standar untuk membuat klaster DB, atau mengubah klaster DB setelah Anda membuatnya untuk mengubah pengaturan.

12. Pilih Buat basis data.

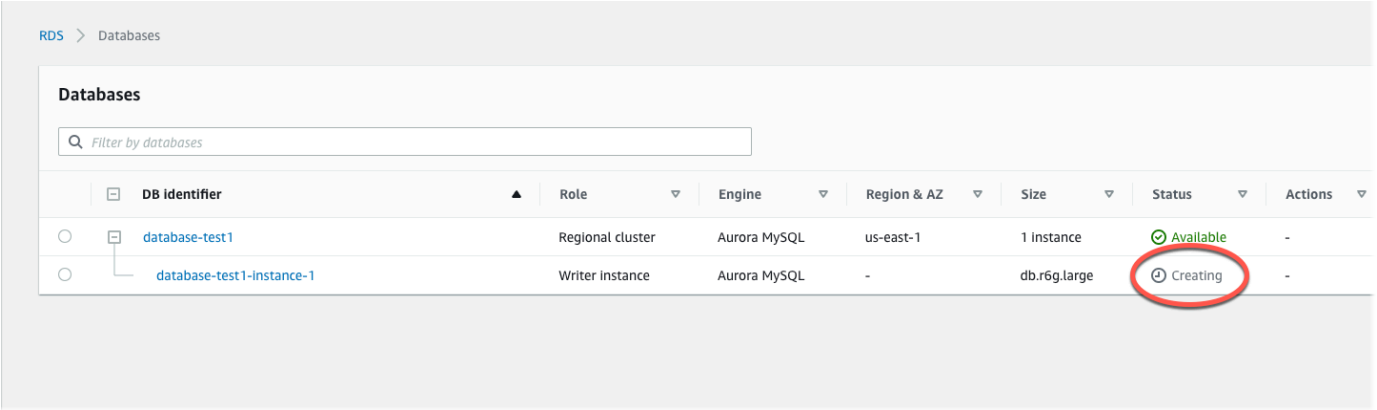
    Untuk melihat nama pengguna dan kata sandi utama untuk klaster DB, pilih Lihat detail kredensial.

    Anda dapat menggunakan nama pengguna dan kata sandi yang ditampilkan untuk terhubung ke klaster DB sebagai pengguna utama.

    > **Important**
    >
    > Anda tidak dapat melihat kata sandi pengguna utama lagi. Jika tidak mencatatnya, Anda mungkin harus mengubahnya.
    > Jika perlu mengubah kata sandi pengguna utama setelah klaster DB tersedia, Anda dapat mengubah klaster DB untuk melakukannya. Untuk informasi selengkapnya tentang cara memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

13. Dalam daftar Basis data, pilih nama klaster DB Aurora MySQL baru untuk menampilkan detailnya.

    Instans penulis memiliki status Membuat hingga klaster DB siap digunakan.



Saat status instans penulis berubah menjadi Tersedia, Anda dapat terhubung ke klaster DB. Tergantung pada kelas instans DB dan jumlah penyimpanan, diperlukan waktu hingga 20 menit sebelum klaster DB baru tersedia.

(Opsional) Buat VPC, instans EC2, dan cluster Aurora MySQL menggunakan AWS CloudFormation

Alih-alih menggunakan konsol untuk membuat VPC, instans EC2, dan cluster DB MySQL Aurora, Anda dapat AWS CloudFormation menggunakannya untuk menyediakan sumber daya dengan memperlakukan infrastruktur sebagai kode. AWS Untuk membantu Anda mengatur AWS sumber daya Anda menjadi unit yang lebih kecil dan lebih mudah dikelola, Anda dapat menggunakan fungsionalitas tumpukan AWS CloudFormation bersarang. Untuk informasi selengkapnya, lihat Membuat tumpukan di AWS CloudFormation konsol dan Bekerja dengan tumpukan bersarang.

> **Important**
>
> AWS CloudFormation gratis, tetapi sumber daya yang CloudFormation menciptakan hidup. Anda dikenakan biaya penggunaan standar untuk sumber daya ini sampai Anda menghentikannya. Total biaya akan minimal. Untuk informasi tentang bagaimana Anda dapat meminimalkan biaya apa pun, buka Tingkat AWS Gratis.

Untuk membuat sumber daya Anda menggunakan AWS CloudFormation konsol, selesaikan langkah-langkah berikut:

- Langkah 1: Unduh CloudFormation template
- Langkah 2: Konfigurasikan sumber daya Anda menggunakan CloudFormation

Unduh CloudFormation templatnya

CloudFormation Template adalah file teks JSON atau YAMG yang berisi informasi konfigurasi tentang sumber daya yang ingin Anda buat di tumpukan. Template ini juga membuat VPC dan host benteng untuk Anda bersama dengan cluster Aurora.

Untuk mengunduh file template, buka tautan berikut, template Aurora MySQL. CloudFormation

Di halaman Github, klik tombol Unduh file mentah untuk menyimpan file YAMAL template.

Konfigurasikan sumber daya Anda menggunakan CloudFormation

> **Note**
>
> Sebelum memulai proses ini, pastikan Anda memiliki pasangan Kunci untuk instans EC2 di. Akun AWS Untuk informasi selengkapnya, lihat Pasangan kunci Amazon EC2 dan instans Linux.

Ketika Anda menggunakan AWS CloudFormation template, Anda harus memilih parameter yang benar untuk memastikan sumber daya Anda dibuat dengan benar. Ikuti langkah-langkah di bawah ini:

1. Masuk ke AWS Management Console dan buka AWS CloudFormation konsol di https://console.aws.amazon.com/cloudformation.
2. Pilih Buat tumpukan.
3. Di bagian Tentukan templat, pilih Unggah file templat dari komputer Anda, lalu pilih Berikutnya.
4. Di halaman Tentukan detail tumpukan, atur parameter berikut:
   a. Tetapkan nama Stack ke AurMySQL TestStack.
   b. Di bawah Parameter, atur Availability Zone dengan memilih dua zona ketersediaan.
   c. Di bawah konfigurasi Linux Bastion Host, untuk Key Name, pilih key pair untuk login ke instans EC2 Anda.
   d. Dalam pengaturan konfigurasi Linux Bastion Host, atur rentang IP yang Diizinkan ke alamat IP Anda. Untuk terhubung ke instans EC2 di VPC Anda menggunakan Secure Shell (SSH), tentukan alamat IP publik Anda menggunakan layanan di https://checkip.amazonaws.com. Contoh alamat IP adalah 192.0.2.1/32.

> **Warning**
>
> Jika menggunakan `0.0.0.0/0` untuk akses SSH, Anda memungkinkan semua alamat IP untuk mengakses instans publik EC2 Anda menggunakan SSH. Hal ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, Anda hanya dapat memberikan otorisasi pada alamat IP atau rentang alamat tertentu saja untuk mengakses instans EC2 Anda menggunakan SSH.

e. Di bawah konfigurasi Database General, atur kelas instance Database ke db.r6g.large.

f. Tetapkan nama Database ke**`database-test1`**.

g. Untuk nama pengguna master Database, masukkan nama untuk pengguna master.

h. Atur Kelola kata sandi pengguna master DB dengan Secrets Manager `false` untuk tutorial ini.

i. Untuk kata sandi Database, tetapkan kata sandi pilihan Anda. Ingat kata sandi ini untuk langkah lebih lanjut dalam tutorial.

j. Setel penerapan Multi-AZ ke. `false`

k. Biarkan semua pengaturan lainnya sebagai nilai default. Klik Berikutnya untuk melanjutkan.

5. Di halaman Configure stack options, tinggalkan semua opsi default. Klik Berikutnya untuk melanjutkan.
6. Di halaman tumpukan Tinjauan, pilih Kirim setelah memeriksa database dan opsi host bastion Linux.

Setelah proses pembuatan tumpukan selesai, lihat tumpukan dengan nama BastionStackdan AMSNS untuk mencatat informasi yang Anda butuhkan untuk terhubung ke database. Untuk informasi selengkapnya, lihat Melihat data AWS CloudFormation tumpukan dan sumber daya di AWS Management Console.

Langkah 3: Membuat klaster DB Aurora MySQL

Anda dapat menggunakan aplikasi klien SQL standar untuk menghubungkan ke klaster DB. Dalam contoh ini, Anda terhubung ke klaster DB Aurora MySQL menggunakan klien baris perintah mysql.

Untuk menghubungkan ke klaster DB Aurora MySQL

1. Temukan titik akhir (nama DNS) dan nomor port untuk instans penulis klaster DB Anda.

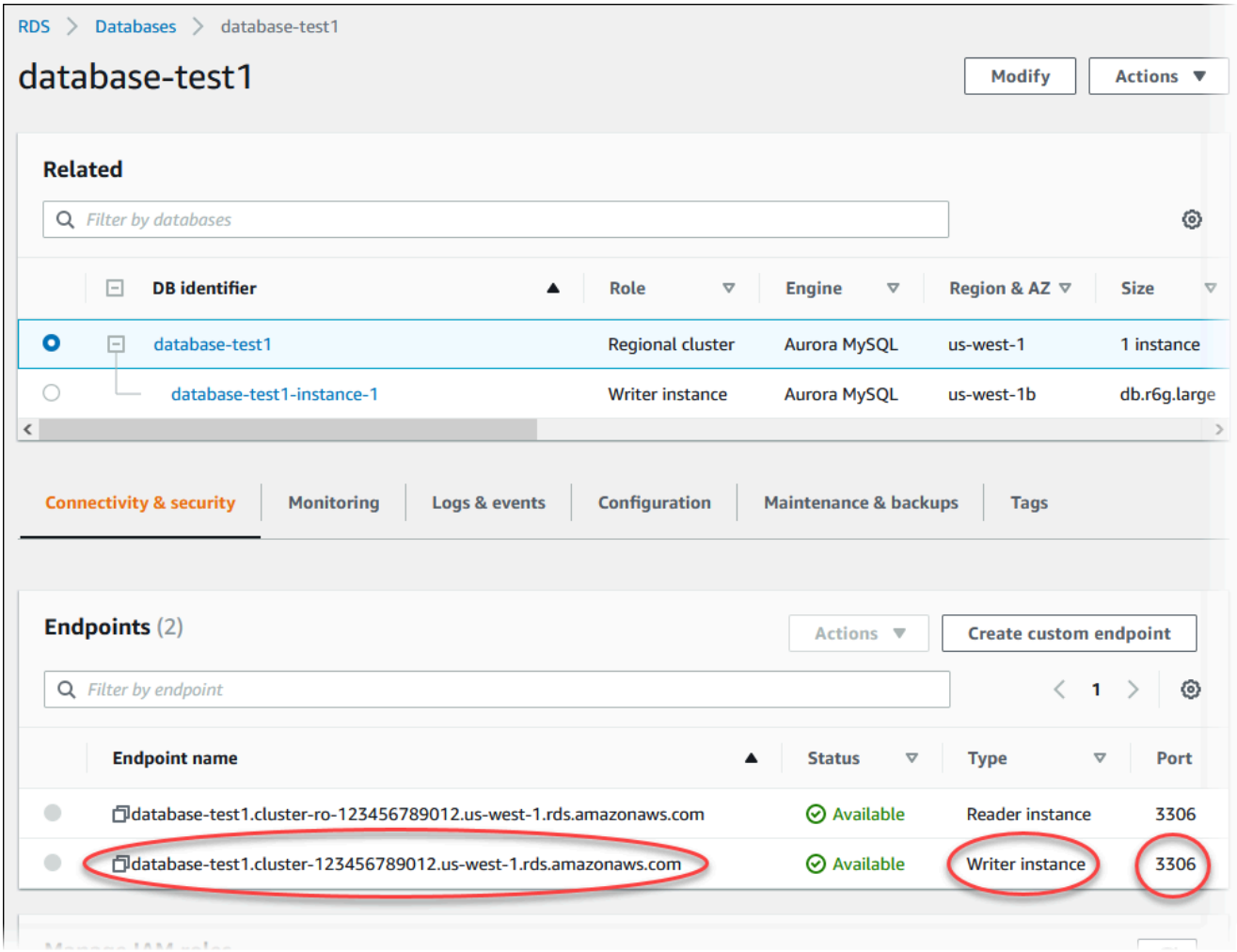
a. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

b. Di sudut kanan atas konsol Amazon RDS, pilih Wilayah AWS untuk klaster DB.

c. Di panel navigasi, pilih Basis data.

d. Pilih nama klaster DB Aurora MySQL untuk menampilkan detailnya.

e. Di tab Konektivitas & keamanan, salin titik akhir instans penulis. Perhatikan juga nomor port. Anda memerlukan titik akhir dan nomor port untuk terhubung ke klaster DB.



2. Hubungkan ke instans EC2 yang Anda buat sebelumnya dengan mengikuti langkah-langkah di Menghubungkan ke instans Linux di Panduan Pengguna Amazon EC2 untuk Instans Linux.

   Sebaiknya Anda menghubungkan ke instans EC2 menggunakan SSH. Jika utilitas klien SSH diinstal di Windows, Linux, atau Mac, Anda dapat menghubungkan ke instans menggunakan format perintah berikut:

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

Misalnya, asumsikan bahwa `ec2-database-connect-key-pair.pem` disimpan di `/dir1` di Linux, dan DNS IPv4 publik untuk instans EC2 Anda adalah `ec2-12-345-678-90.compute-1.amazonaws.com`. Perintah SSH Anda akan tampak seperti berikut:

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. Dapatkan perbaikan bug dan pembaruan keamanan terbaru dengan memperbarui perangkat lunak di instans EC2 Anda. Untuk melakukannya, gunakan perintah berikut.

> **Note**
>
> Opsi `-y` menginstal pembaruan tanpa meminta konfirmasi. Hilangkan opsi ini untuk memeriksa pembaruan sebelum menginstal.

```
sudo dnf update -y
```

4. Untuk menginstal klien baris perintah mysql dari MariaDB di Amazon Linux 2023, jalankan perintah berikut:

```
sudo dnf install mariadb105
```

5. Hubungkan ke klaster DB Aurora MySQL. Misalnya, masukkan perintah berikut. Tindakan ini memungkinkan Anda terhubung ke klaster DB Aurora MySQL menggunakan klien MySQL.

   Ganti titik akhir instans penulis dengan *`endpoint`*, dan ganti nama pengguna utama yang Anda gunakan dengan *`admin`*. Masukkan kata sandi utama yang Anda gunakan saat dimintai kata sandi.

```
mysql -h endpoint -P 3306 -u admin -p
```

   Setelah Anda memasukkan kata sandi untuk pengguna, Anda akan melihat output yang serupa dengan yang berikut ini.

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 217
Server version: 8.0.23 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

Untuk informasi lebih lanjut tentang menghubungkan ke klaster DB Aurora MySQL, lihat Menghubungkan ke klaster DB Amazon Aurora MySQL. Jika Anda tidak dapat terhubung ke klaster DB Anda, lihat Tidak dapat terhubung ke instans DB Amazon RDS.

Untuk keamanan, ini adalah praktik terbaik untuk menggunakan koneksi terenkripsi. Hanya gunakan koneksi MySQL yang tidak terenkripsi saat klien dan server berada di VPC yang sama dan jaringan tepercaya. Untuk informasi tentang cara menggunakan koneksi terenkripsi, lihat Menghubungkan dengan SSL untuk Aurora MySQL.

6. Jalankan perintah SQL.

   Misalnya, perintah SQL berikut menunjukkan tanggal dan waktu saat ini:

   ```
   SELECT CURRENT_TIMESTAMP;
   ```

Langkah 4: Hapus instans EC2 dan klaster DB

Setelah Anda terhubung ke dan menjelajahi instans EC2 dan klaster DB sampel yang Anda buat, hapus instans tersebut sehingga Anda tidak lagi dikenakan biaya untuk instans DB tersebut.

Jika Anda biasa AWS CloudFormation membuat sumber daya, lewati langkah ini dan lanjutkan ke langkah berikutnya.

Untuk menghapus instans EC2

1. Masuk ke AWS Management Console dan buka konsol Amazon EC2 di https://console.aws.amazon.com/ec2/.
2. Di panel navigasi, pilih Instans.
3. Pilih instans EC2, dan pilih Status instans, Akhiri instans.

4. Pilih Akhiri saat diminta untuk konfirmasi.

Untuk informasi selengkapnya tentang menghapus instans EC2, lihat Mengakhiri Instans Anda dalam Panduan Pengguna Amazon EC2 untuk Instans Linux.

Untuk menghapus klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data lalu pilih instans DB yang terkait dengan klaster DB.
3. Untuk Tindakan, pilih Hapus.
4. Hapus Buat snapshot akhir?.
5. Lengkapi pengakuan dan pilih Hapus.

Setelah semua instans DB yang terkait dengan klaster DB dihapus, klaster DB akan dihapus secara otomatis.

(Opsional) Hapus instans EC2 dan cluster DB yang dibuat dengan CloudFormation

Jika Anda biasa AWS CloudFormation membuat sumber daya, hapus CloudFormation tumpukan setelah Anda terhubung dan jelajahi contoh instans EC2 dan kluster DB, sehingga Anda tidak lagi dikenakan biaya untuk itu.

Untuk menghapus sumber CloudFormation daya

1. Buka AWS CloudFormation konsol.
2. Pada halaman Stacks di CloudFormation konsol, pilih tumpukan root (tumpukan tanpa nama VPCStack, BastionStack atau AMSNS).
3. Pilih Hapus.
4. Pilih Hapus tumpukan saat diminta konfirmasi.

Untuk informasi selengkapnya tentang menghapus tumpukan CloudFormation, lihat Menghapus tumpukan di AWS CloudFormation konsol di AWS CloudFormation Panduan Pengguna.

(Opsional) Menghubungkan klaster DB Anda ke fungsi Lambda

Anda juga dapat menghubungkan klaster DB Aurora MySQL ke sumber daya komputasi nirserver Lambda. Fungsi Lambda memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola infrastruktur. Fungsi Lambda juga memungkinkan Anda untuk otomatis merespons permintaan eksekusi kode pada skala apa pun, mulai dari selusin peristiwa dalam sehari hingga ratusan per detik. Lihat informasi yang lebih lengkap di Menghubungkan secara otomatis fungsi Lambda dan klaster DB Aurora.

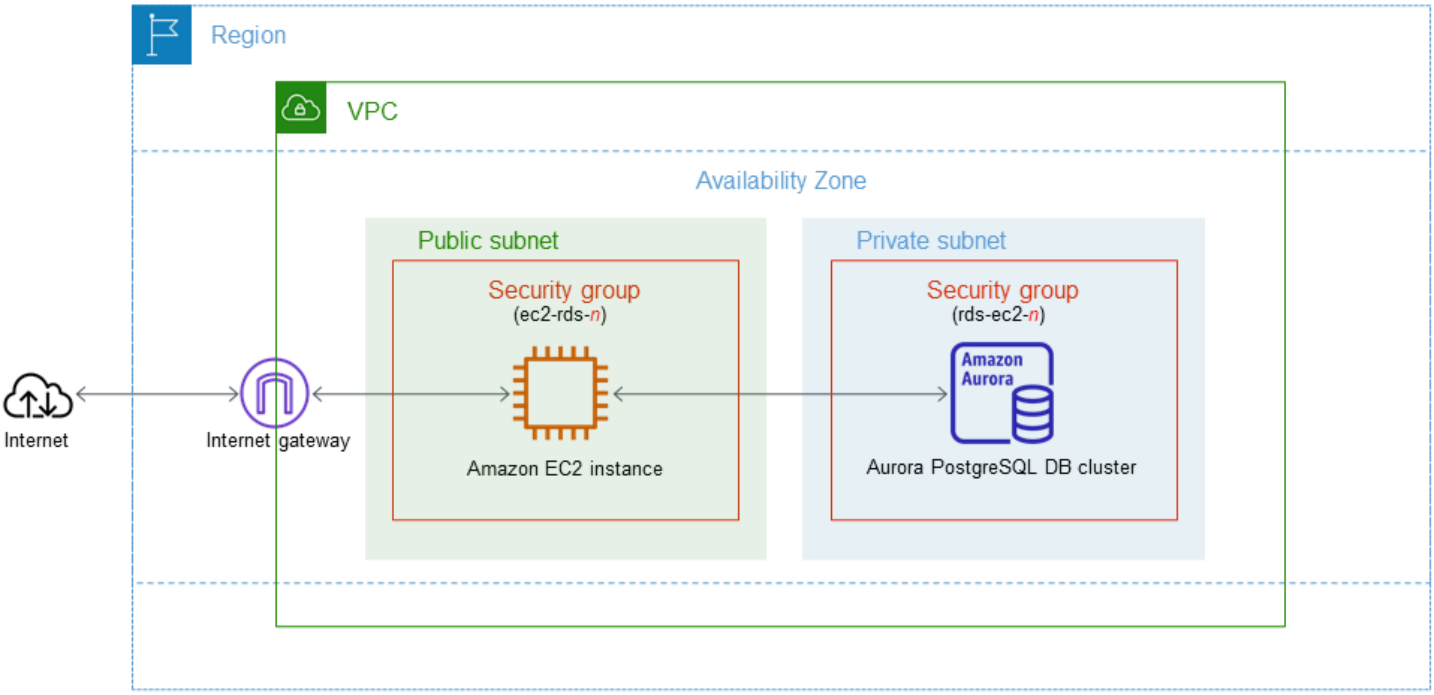
Membuat dan terhubung ke klaster DB Aurora PostgreSQL

Tutorial ini membuat instans EC2 dan klaster DB Aurora PostgreSQL. Tutorial ini menunjukkan cara mengakses klaster DB dari instans EC2 menggunakan klien PostgreSQL standar. Sebagai praktik terbaik, tutorial ini membuat klaster DB privat dalam cloud privat virtual (VPC). Dalam kebanyakan kasus, sumber daya lain dalam VPC yang sama, seperti instans EC2, dapat mengakses klaster DB, tetapi sumber daya di luar VPC tidak dapat mengaksesnya.

Setelah Anda menyelesaikan tutorial, ada subnet publik dan privat di setiap Zona Ketersediaan di VPC Anda. Dalam satu Zona Ketersediaan, instans EC2 berada di subnet publik, dan instans DB berada di subnet privat.

> **Important**
>
> Tidak ada biaya untuk membuat AWS akun. Namun, dengan menyelesaikan tutorial ini, Anda mungkin dikenakan biaya untuk AWS sumber daya yang Anda gunakan. Anda dapat menghapus sumber daya ini setelah menyelesaikan tutorial jika tidak diperlukan lagi.

Diagram berikut menunjukkan konfigurasi setelah tutorial selesai.



Tutorial ini memungkinkan Anda untuk membuat sumber daya Anda dengan menggunakan salah satu metode berikut:

1. Gunakan AWS Management Console - Langkah 1: Buat instans EC2 dan Langkaah 2: Membuat klaster DB Aurora PostgreSQL
2. Gunakan AWS CloudFormation untuk membuat instance database dan instans EC2 - (Opsional) Buat VPC, instans EC2, dan cluster Aurora PostgreSQL menggunakan AWS CloudFormation

Metode pertama menggunakan Easy create untuk membuat cluster Aurora PostgreSQL DB pribadi dengan. AWS Management Console Di sini, Anda hanya menentukan jenis mesin DB, ukuran instans DB, dan pengidentifikasi cluster DB. Pembuatan Mudah menggunakan pengaturan default untuk opsi konfigurasi lainnya.

Saat Anda menggunakan Standard create sebagai gantinya, Anda dapat menentukan lebih banyak opsi konfigurasi saat membuat cluster DB. Opsi ini mencakup pengaturan untuk ketersediaan, keamanan, cadangan, dan pemeliharaan. Untuk membuat klaster DB publik, Anda harus menggunakan Pembuatan standar. Untuk informasi, lihat the section called “Membuat klaster DB”.

Topik

- Prasyarat
- Langkah 1: Buat instans EC2

- Langkaah 2: Membuat klaster DB Aurora PostgreSQL
- (Opsional) Buat VPC, instans EC2, dan cluster Aurora PostgreSQL menggunakan AWS CloudFormation
- Langkah 3: Membuat klaster DB Aurora PostgreSQL
- Langkah 4: Hapus instans EC2 dan klaster DB
- (Opsional) Hapus instans EC2 dan cluster DB yang dibuat dengan CloudFormation
- (Opsional) Menghubungkan klaster DB Anda ke fungsi Lambda

Prasyarat

Sebelum memulai, selesaikan langkah-langkah di bagian berikut:

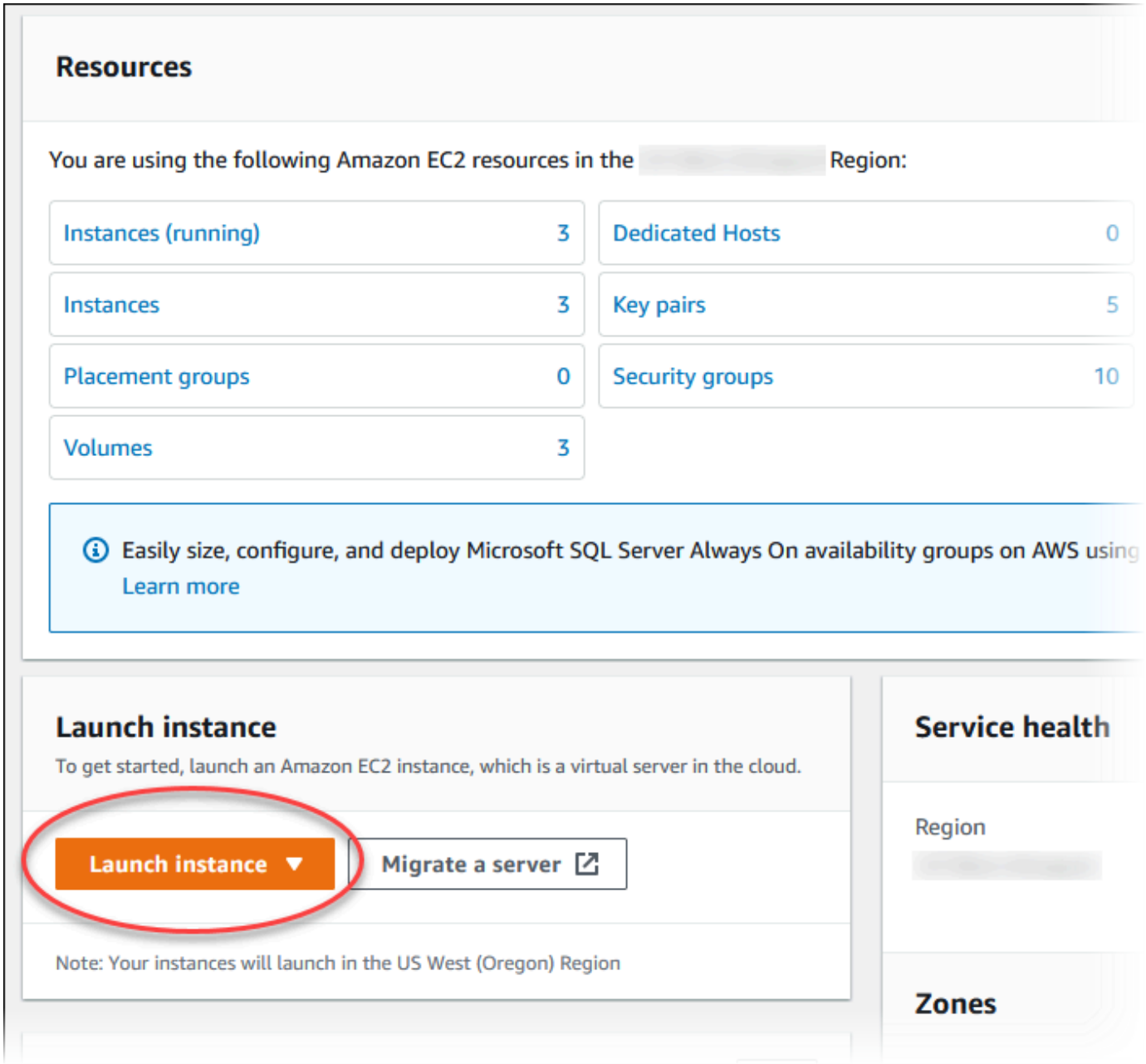
- Mendaftar Akun AWS
- Membuat pengguna administratif

Langkah 1: Buat instans EC2

Buat instans Amazon EC2 yang akan Anda gunakan untuk menghubungkan ke basis data Anda.

Untuk membuat instans EC2

1. Masuk ke AWS Management Console dan buka konsol Amazon EC2 di https://console.aws.amazon.com/ec2/.
2. Di sudut kanan atas AWS Management Console, pilih Wilayah AWS di mana Anda ingin membuat instans EC2.
3. Pilih Dasbor EC2, lalu pilih Luncurkan instans seperti yang ditampilkan dalam gambar berikut.

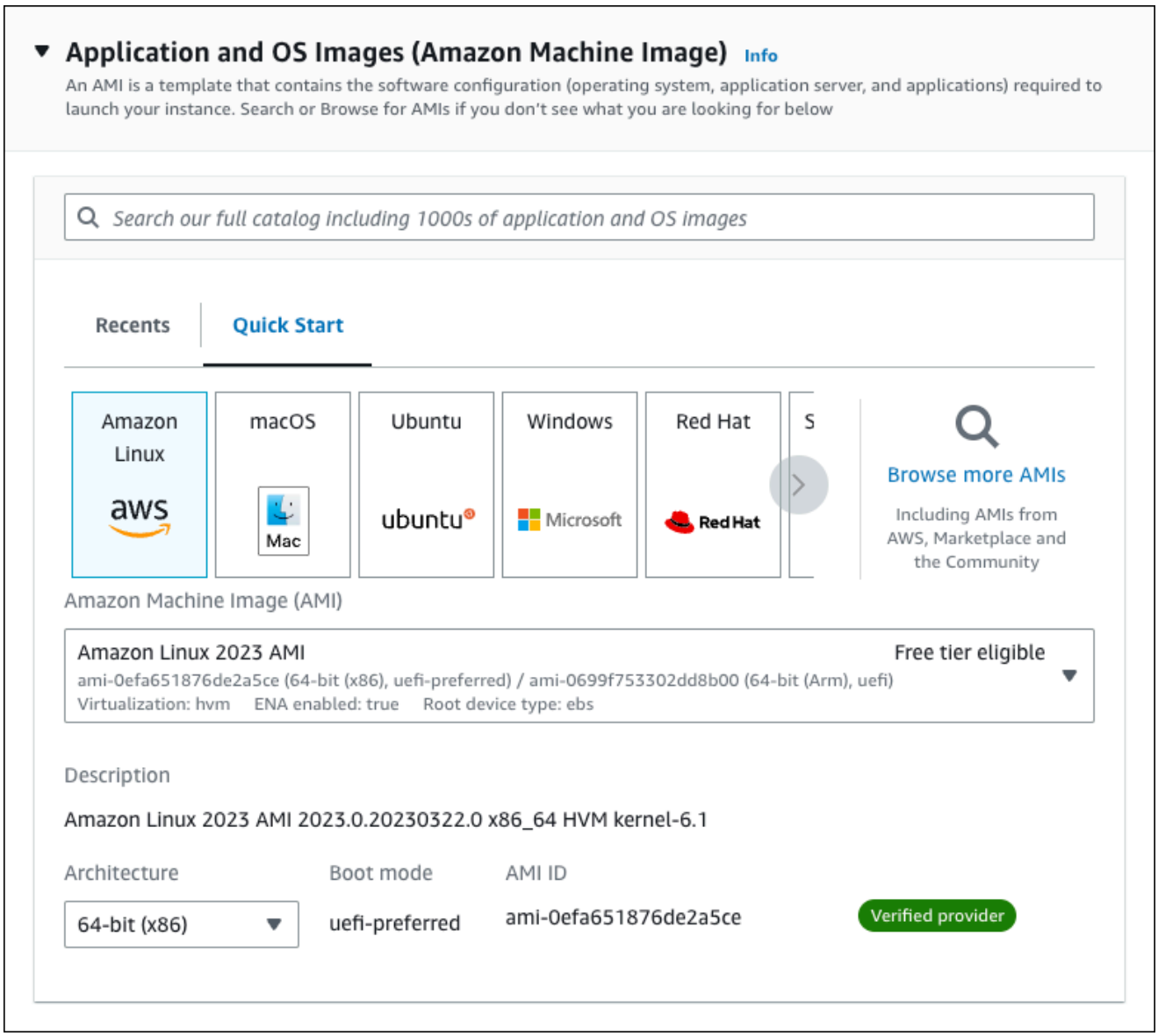


Halaman Meluncurkan instans akan terbuka.

4. Pilih pengaturan berikut di halaman Meluncurkan instans.

   a. Di bagian Nama dan tag, untuk Nama, masukkan **`ec2-database-connect`**.

   b. Di bagian Gambar Aplikasi dan OS (Amazon Machine Image), pilih Amazon Linux, lalu pilih AMI Amazon Linux 2023. Biarkan default untuk pilihan lainnya.



c. Di bagian Jenis instans, pilih t2.micro.

d. Di bagian Pasangan kunci (login), pilih nama Pasangan kunci untuk menggunakan pasangan kunci yang ada. Untuk membuat pasangan kunci baru untuk instans Amazon EC2, pilih Buat Pasangan kunci baru lalu gunakan jendela Buat pasangan kunci untuk membuatnya.

   Untuk informasi selengkapnya tentang membuat pasangan kunci baru, lihat Membuat pasangan kunci di Panduan Pengguna Amazon EC2 untuk Instans Linux.

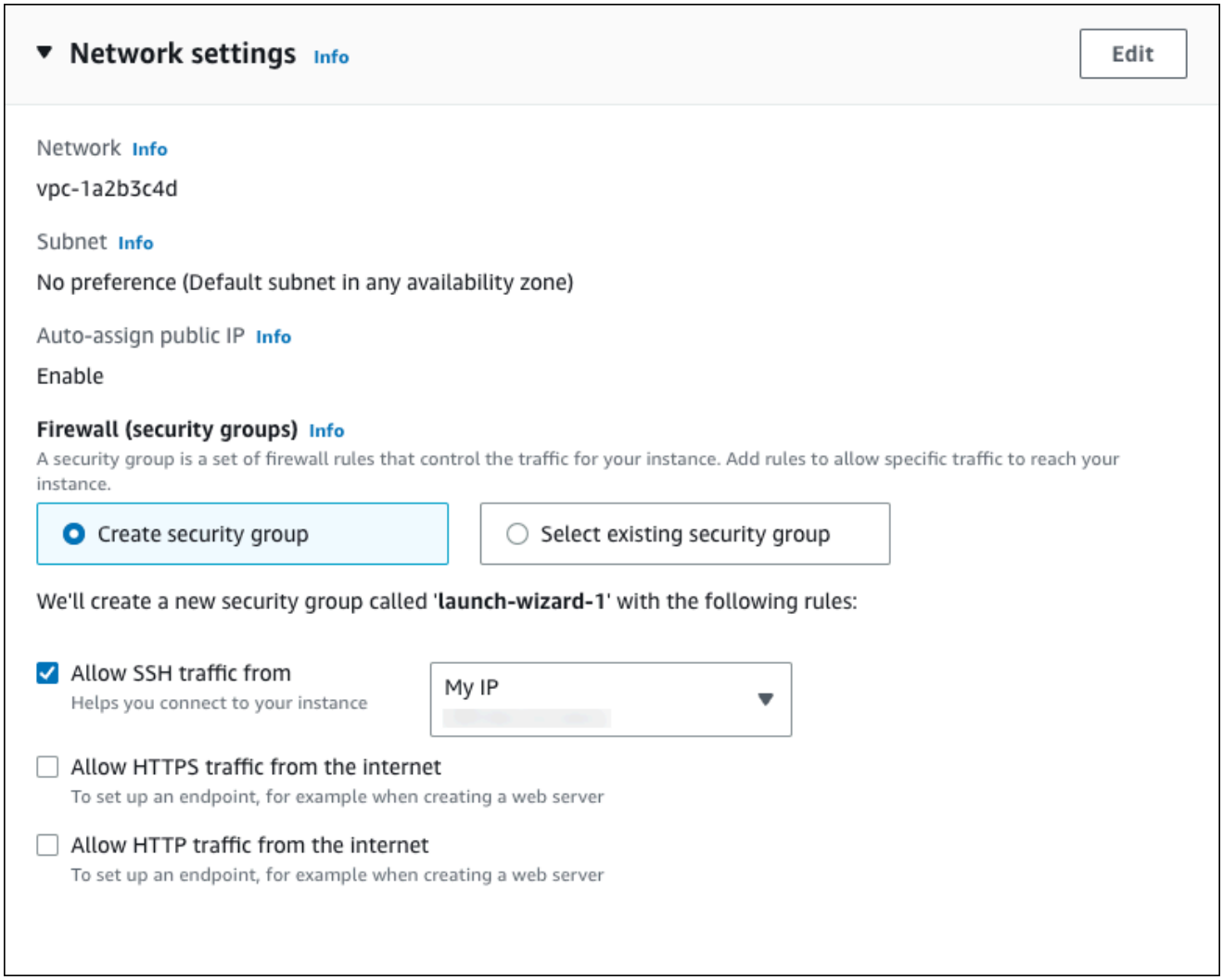
e. Untuk Izinkan lalu lintas SSH di Pengaturan jaringan, pilih sumber koneksi SSH ke instans EC2.

Anda dapat memilih IP Saya jika alamat IP yang ditampilkan benar untuk koneksi SSH. Jika tidak, Anda dapat menentukan alamat IP yang akan digunakan untuk menghubungkan ke instans EC2 di VPC Anda menggunakan Secure Shell (SSH). Untuk menentukan alamat IP publik Anda, Anda dapat membuka layanan di https://checkip.amazonaws.com di jendela atau tab browser lain. Contoh alamat IP adalah 192.0.2.1/32.

Dalam banyak kasus, Anda dapat menghubungkan melalui penyedia layanan Internet (ISP) atau dari belakang firewall Anda tanpa alamat IP statis. Jika demikian, tentukan rentang alamat IP yang digunakan oleh komputer klien.

> **Warning**
>
> Jika menggunakan `0.0.0.0/0` untuk akses SSH, Anda memungkinkan semua alamat IP untuk mengakses instans publik EC2 Anda menggunakan SSH. Hal ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, Anda hanya dapat memberikan otorisasi pada alamat IP atau rentang alamat tertentu saja untuk mengakses instans EC2 Anda menggunakan SSH.

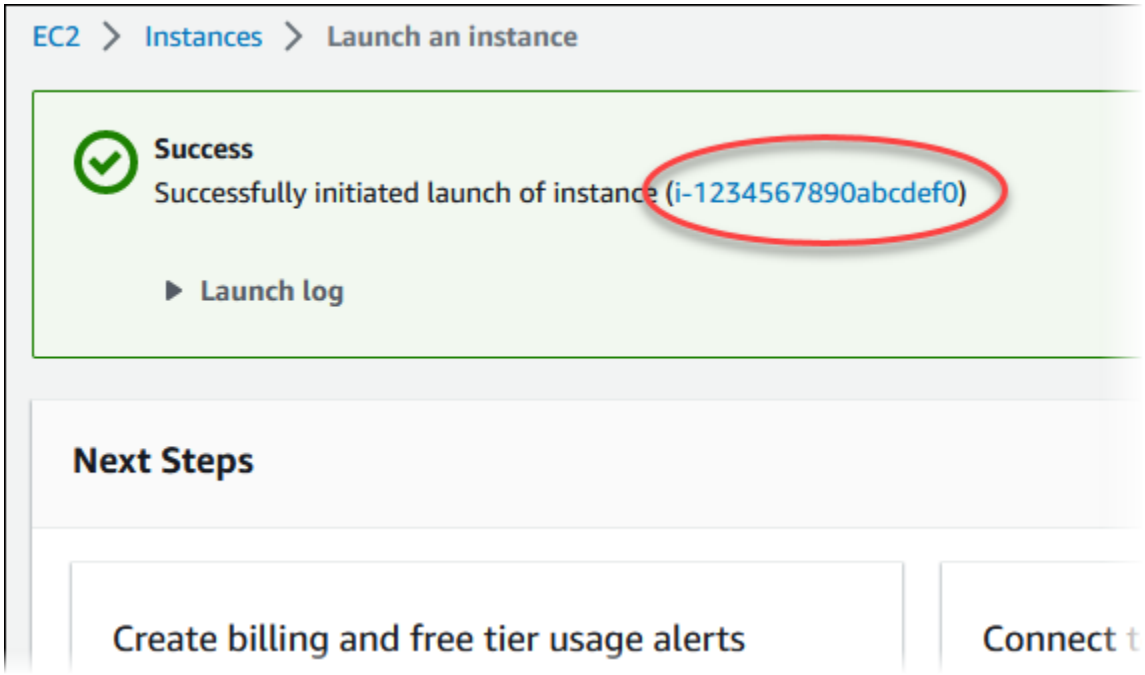
Gambar berikut menunjukkan contoh bagian Pengaturan jaringan.



f. Biarkan nilai default untuk bagian yang lainnya.

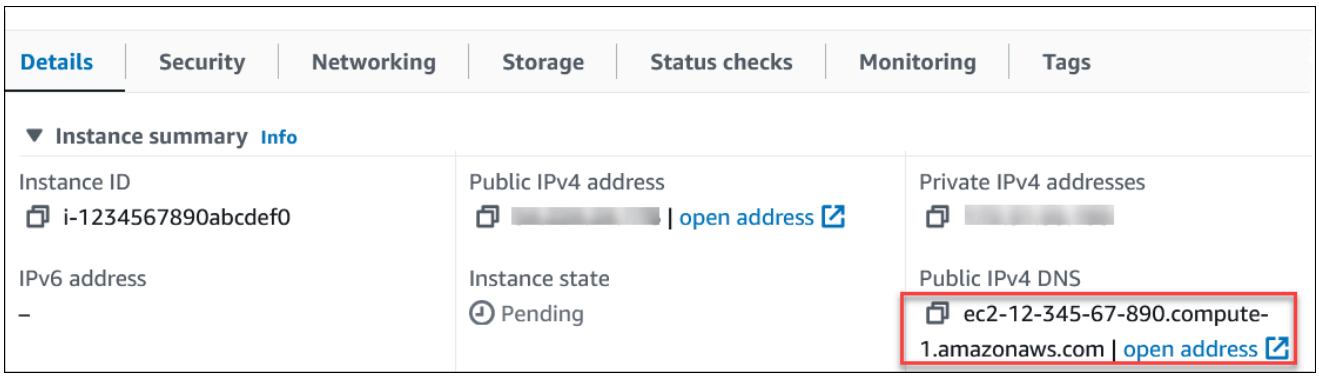
g. Tinjau ringkasan konfigurasi instans EC2 Anda di panel Ringkasan, dan setelah Anda siap, pilih Luncurkan instans.

5. Di halaman Status Peluncuran, catat pengidentifikasi untuk instans EC2 baru Anda, misalnya: `i-1234567890abcdef0`.

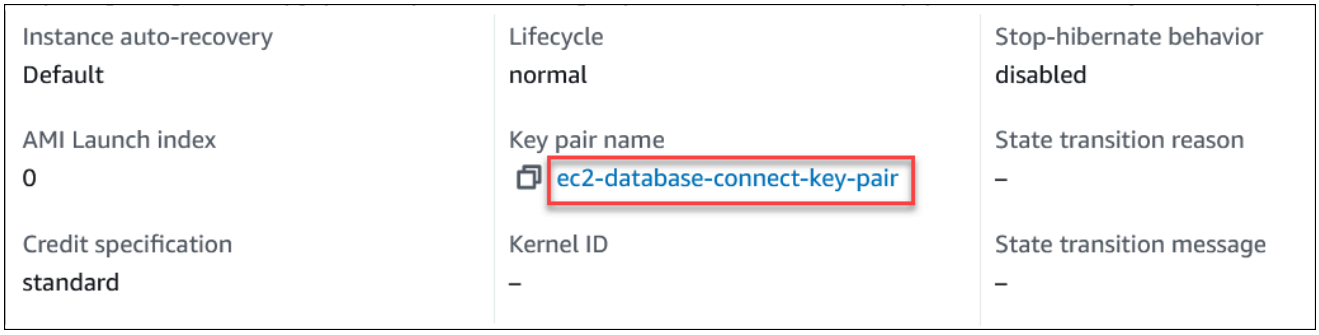


6. Pilih pengidentifikasi instans EC2 untuk membuka daftar instans EC2, lalu pilih instans EC2 Anda.
7. Di tab Detail, catat nilai-nilai berikut, yang akan Anda butuhkan saat menghubungkan menggunakan SSH:

   a. Di Ringkasan instans, catat nilai untuk DNS IPv4 Publik.



   b. Di Detail instans, catat nilai untuk Nama pasangan kunci.



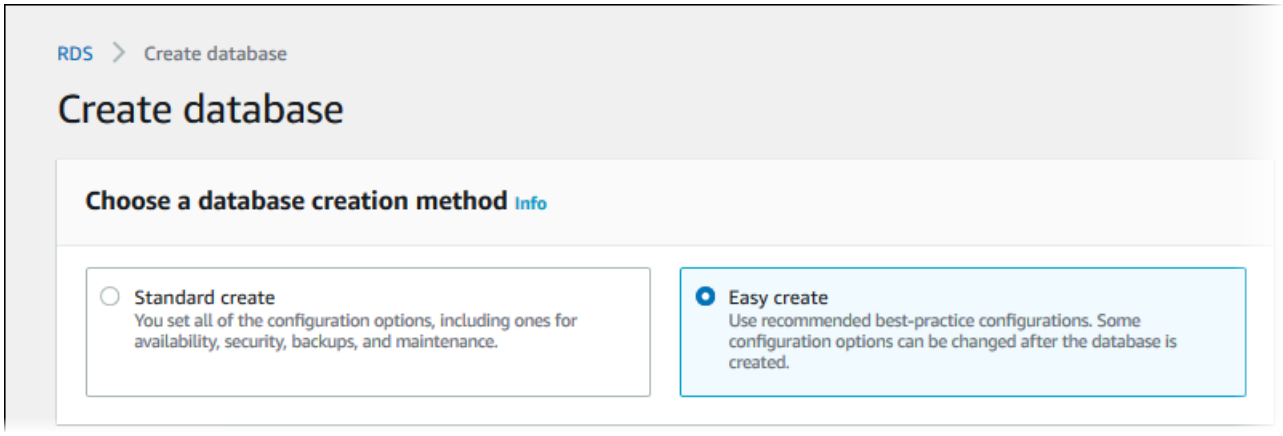
8. Tunggu hingga Status instans untuk instans EC2 Anda berstatus Berjalan sebelum melanjutkan.

Langkaah 2: Membuat klaster DB Aurora PostgreSQL

Dalam contoh ini, Anda menggunakan Pembuatan mudah untuk membuat klaster DB Aurora PostgreSQL dengan kelas instans DB db.t4g.large.

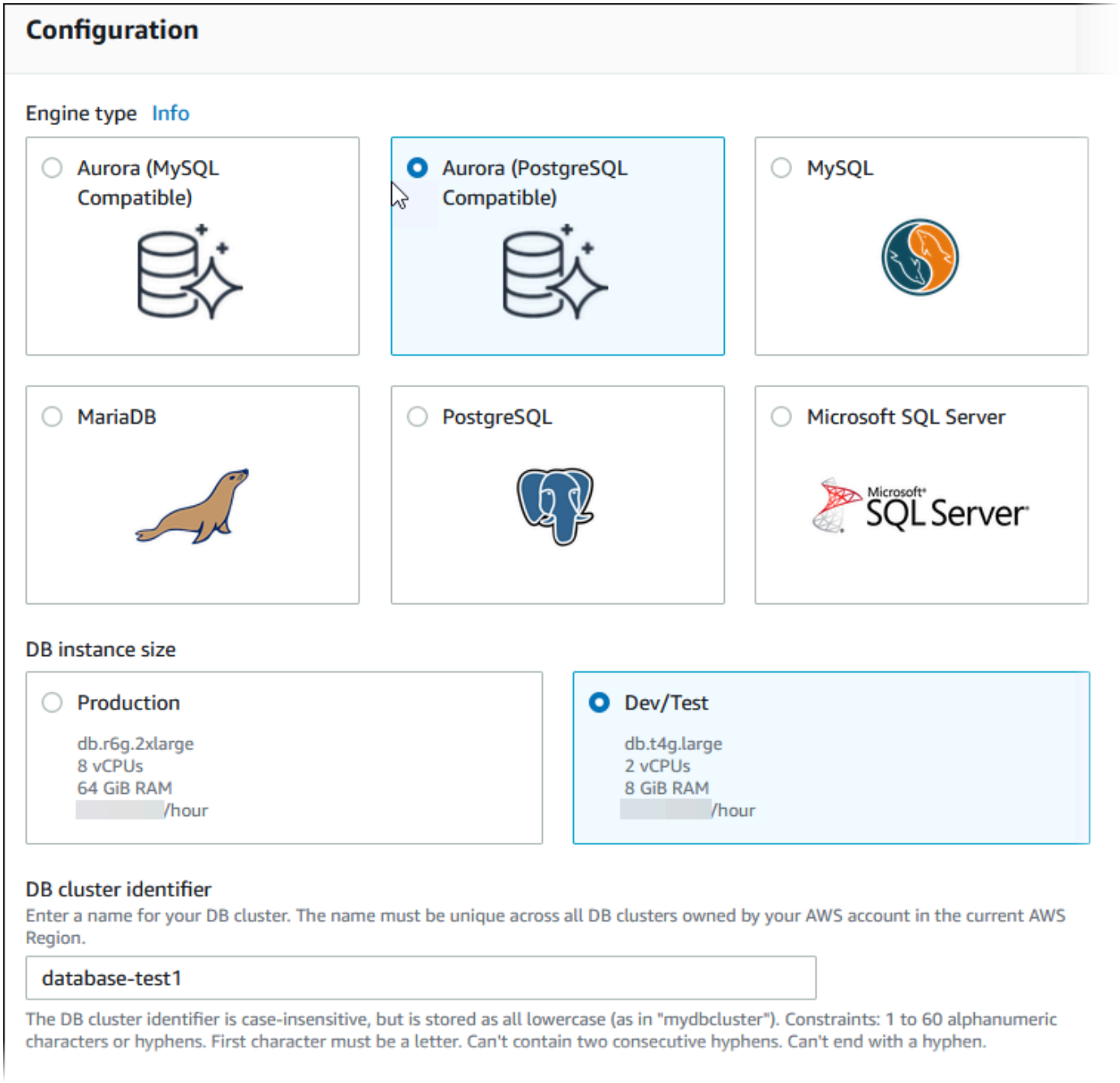
Untuk membuat klaster DB Aurora PostgreSQL dengan Pembuatan mudah

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas konsol Amazon RDS, pilih Wilayah AWS tempat Anda akan membuat klaster DB.
3. Di panel navigasi, pilih Basis data.
4. Pilih Buat basis data dan pastikan bahwa Pembuatan mudah dipilih.



5. Di Konfigurasi, pilih Aurora (Kompatibel dengan PostgreSQL) untuk Jenis mesin.
6. Untuk Ukuran instans DB, pilih Dev/Tes.
7. Untuk pengidentifikasi klaster DB, masukkan **`database-test1`**.

Tampilan halaman Membuat basis data seperti gambar berikut.

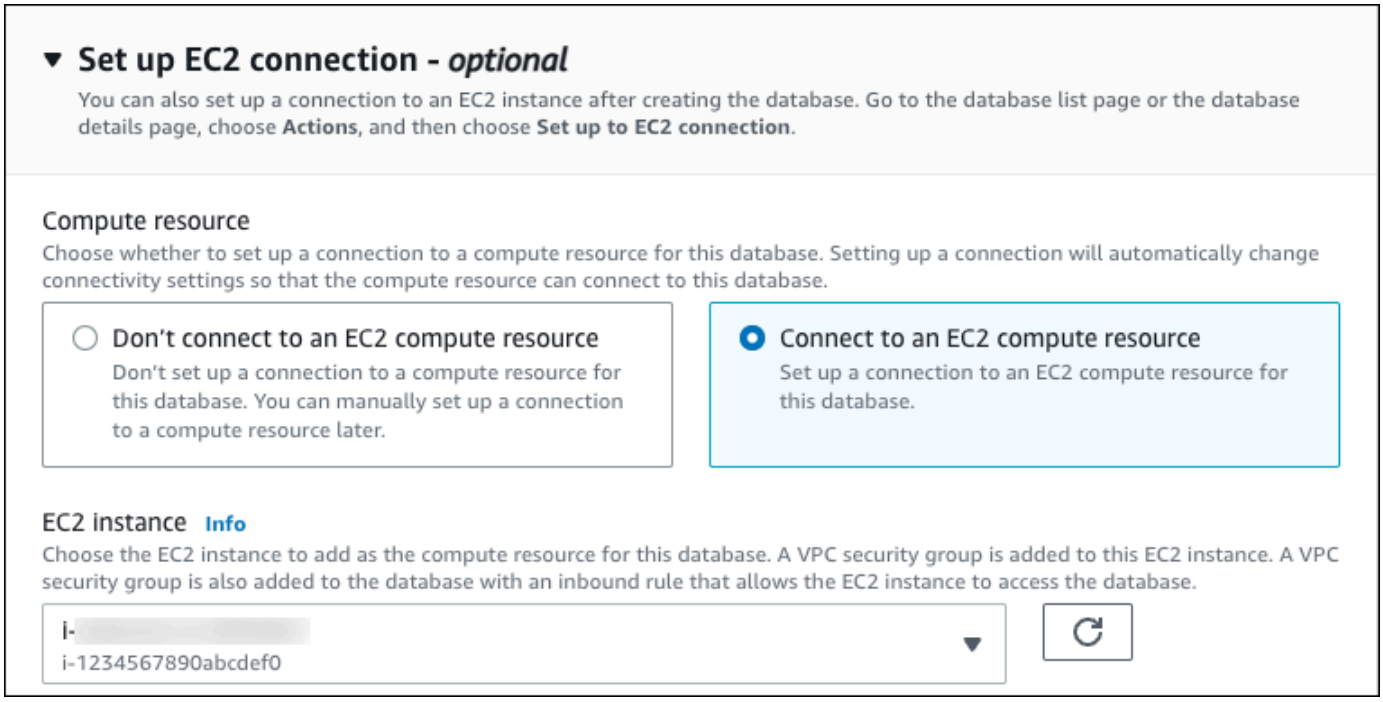


8. Untuk Nama pengguna utama, masukkan nama untuk pengguna, atau tetap gunakan nama default (**`postgres`**).
9. Untuk menggunakan kata sandi utama yang dibuat secara otomatis untuk klaster DB, pilih Buat kata sandi secara otomatis.

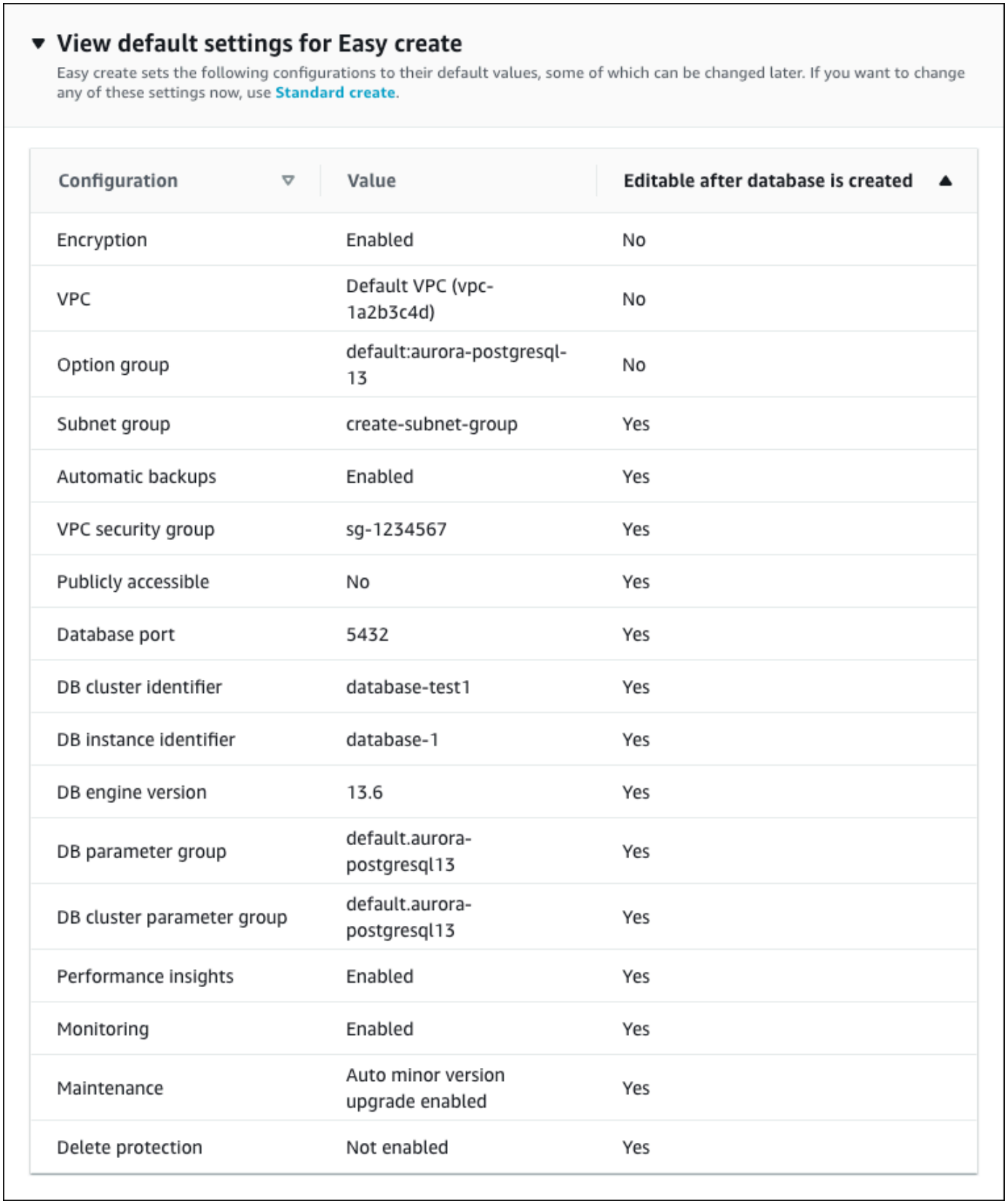
Untuk memasukkan kata sandi utama Anda, hapus centang pada Buat kata sandi secara otomatis, lalu masukkan kata sandi yang sama dalam Kata sandi utama dan Konfirmasi kata sandi.

10. Untuk menyiapkan koneksi dengan instans EC2 yang Anda buat sebelumnya, buka Menyiapkan koneksi EC2 - opsional.

    Pilih Hubungkan ke sumber daya komputasi EC2. Pilih instans EC2 yang Anda buat sebelumnya.



11. Buka Lihat pengaturan default untuk Pembuatan Mudah.



View default settings for Easy create

Easy create sets the following configurations to their default values, some of which can be changed later. If you want to change any of these settings now, use **Standard create**.

Configuration	Value	Editable after database is created
Encryption	Enabled	No
VPC	Default VPC (vpc-1a2b3c4d)	No
Option group	default:aurora-postgresql-13	No
Subnet group	create-subnet-group	Yes
Automatic backups	Enabled	Yes
VPC security group	sg-1234567	Yes
Publicly accessible	No	Yes
Database port	5432	Yes
DB cluster identifier	database-test1	Yes
DB instance identifier	database-1	Yes
DB engine version	13.6	Yes
DB parameter group	default.aurora-postgresql13	Yes
DB cluster parameter group	default.aurora-postgresql13	Yes
Performance insights	Enabled	Yes
Monitoring	Enabled	Yes
Maintenance	Auto minor version upgrade enabled	Yes
Delete protection	Not enabled	Yes

Anda dapat memeriksa pengaturan default yang digunakan dengan Pembuatan mudah. Kolom Dapat diedit setelah basis data dibuat menunjukkan opsi yang dapat Anda ubah setelah membuat basis data.

- Jika pengaturan memiliki Tidak di kolom tersebut, dan Anda menginginkan pengaturan yang berbeda, Anda dapat menggunakan Pembuatan standar untuk membuat klaster DB.
- Jika pengaturan memiliki Ya di kolom tersebut, dan Anda menginginkan pengaturan yang berbeda, Anda dapat menggunakan Pembuatan standar untuk membuat klaster DB, atau mengubah klaster DB setelah Anda membuatnya untuk mengubah pengaturan.

12. Pilih Buat basis data.

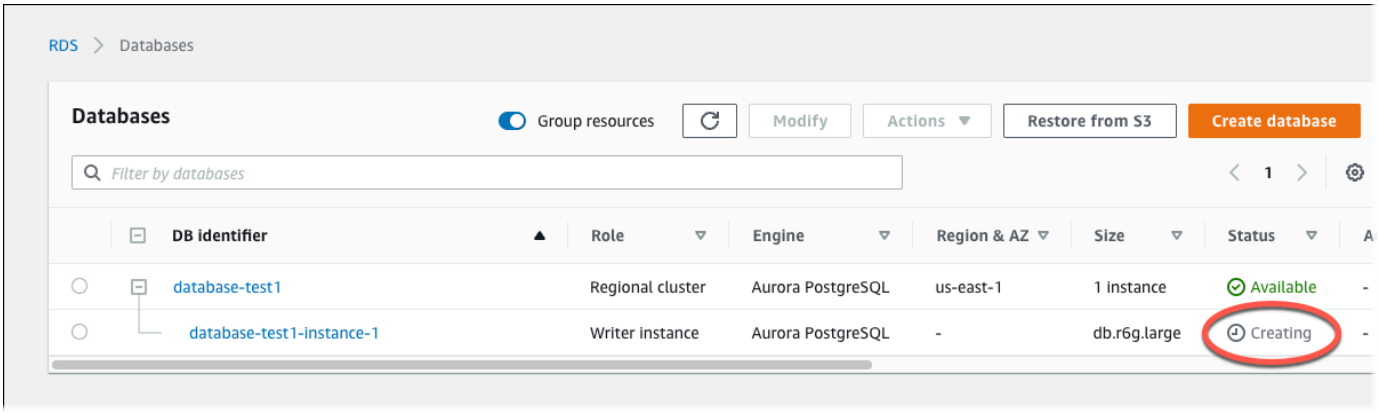
    Untuk melihat nama pengguna dan kata sandi utama untuk klaster DB, pilih Lihat detail kredensial.

    Anda dapat menggunakan nama pengguna dan kata sandi yang ditampilkan untuk terhubung ke klaster DB sebagai pengguna utama.

    > **Important**
    >
    > Anda tidak dapat melihat kata sandi pengguna utama lagi. Jika tidak mencatatnya, Anda mungkin harus mengubahnya.
    > Jika perlu mengubah kata sandi pengguna utama setelah klaster DB tersedia, Anda dapat mengubah klaster DB untuk melakukannya. Untuk informasi selengkapnya tentang cara memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

13. Dalam daftar Basis data, pilih nama klaster DB Aurora PostgreSQL baru untuk menampilkan detailnya.

    Instans penulis memiliki status Membuat hingga klaster DB siap digunakan.



Saat status instans penulis berubah menjadi Tersedia, Anda dapat terhubung ke klaster DB. Tergantung pada kelas instans DB dan jumlah penyimpanan, diperlukan waktu hingga 20 menit sebelum klaster DB baru tersedia.

(Opsional) Buat VPC, instans EC2, dan cluster Aurora PostgreSQL menggunakan AWS CloudFormation

Alih-alih menggunakan konsol untuk membuat VPC, instans EC2, dan klaster Aurora PostgreSQL DB, Anda dapat menggunakannya untuk menyediakan sumber daya dengan memperlakukan infrastruktur sebagai kode. AWS CloudFormation AWS Untuk membantu Anda mengatur AWS sumber daya Anda menjadi unit yang lebih kecil dan lebih mudah dikelola, Anda dapat menggunakan fungsionalitas tumpukan AWS CloudFormation bersarang. Untuk informasi selengkapnya, lihat Membuat tumpukan di AWS CloudFormation konsol dan Bekerja dengan tumpukan bersarang.

> **Important**
>
> AWS CloudFormation gratis, tetapi sumber daya yang CloudFormation menciptakan hidup. Anda dikenakan biaya penggunaan standar untuk sumber daya ini sampai Anda menghentikannya. Total biaya akan minimal. Untuk informasi tentang bagaimana Anda dapat meminimalkan biaya apa pun, buka Tingkat AWS Gratis.

Untuk membuat sumber daya Anda menggunakan AWS CloudFormation konsol, selesaikan langkah-langkah berikut:

- Langkah 1: Unduh CloudFormation template
- Langkah 2: Konfigurasikan sumber daya Anda menggunakan CloudFormation

Unduh CloudFormation templatnya

CloudFormation Template adalah file teks JSON atau YAMG yang berisi informasi konfigurasi tentang sumber daya yang ingin Anda buat di tumpukan. Template ini juga membuat VPC dan host benteng untuk Anda bersama dengan cluster Aurora.

Untuk men-download file template, buka link berikut, Aurora PostgreSQL template. CloudFormation

Di halaman Github, klik tombol Unduh file mentah untuk menyimpan file YAMAL template.

Konfigurasikan sumber daya Anda menggunakan CloudFormation

> **Note**
>
> Sebelum memulai proses ini, pastikan Anda memiliki pasangan Kunci untuk instans EC2 di. Akun AWS Untuk informasi selengkapnya, lihat Pasangan kunci Amazon EC2 dan instans Linux.

Ketika Anda menggunakan AWS CloudFormation template, Anda harus memilih parameter yang benar untuk memastikan sumber daya Anda dibuat dengan benar. Ikuti langkah-langkah di bawah ini:

1. Masuk ke AWS Management Console dan buka AWS CloudFormation konsol di https://console.aws.amazon.com/cloudformation.
2. Pilih Buat tumpukan.
3. Di bagian Tentukan templat, pilih Unggah file templat dari komputer Anda, lalu pilih Berikutnya.
4. Di halaman Tentukan detail tumpukan, atur parameter berikut:
   a. Tetapkan nama Stack ke AurPostgreSQL TestStack.
   b. Di bawah Parameter, atur Availability Zone dengan memilih dua zona ketersediaan.
   c. Di bawah konfigurasi Linux Bastion Host, untuk Key Name, pilih key pair untuk login ke instans EC2 Anda.
   d. Dalam pengaturan konfigurasi Linux Bastion Host, atur rentang IP yang Diizinkan ke alamat IP Anda. Untuk terhubung ke instans EC2 di VPC Anda menggunakan Secure Shell (SSH), tentukan alamat IP publik Anda menggunakan layanan di https://checkip.amazonaws.com. Contoh alamat IP adalah 192.0.2.1/32.

> **Warning**
>
> Jika menggunakan `0.0.0.0/0` untuk akses SSH, Anda memungkinkan semua alamat IP untuk mengakses instans publik EC2 Anda menggunakan SSH. Hal ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, Anda hanya dapat memberikan otorisasi pada alamat IP atau rentang alamat tertentu saja untuk mengakses instans EC2 Anda menggunakan SSH.

e. Di bawah konfigurasi Database General, atur kelas instance Database ke db.t4g.large.

f. Tetapkan nama Database ke**`database-test1`**.

g. Untuk nama pengguna master Database, masukkan nama untuk pengguna master.

h. Atur Kelola kata sandi pengguna master DB dengan Secrets Manager `false` untuk tutorial ini.

i. Untuk kata sandi Database, tetapkan kata sandi pilihan Anda. Ingat kata sandi ini untuk langkah lebih lanjut dalam tutorial.

j. Setel penerapan Multi-AZ ke. `false`

k. Biarkan semua pengaturan lainnya sebagai nilai default. Klik Berikutnya untuk melanjutkan.

5. Di halaman Configure stack options, tinggalkan semua opsi default. Klik Berikutnya untuk melanjutkan.
6. Di halaman tumpukan Tinjauan, pilih Kirim setelah memeriksa database dan opsi host bastion Linux.

Setelah proses pembuatan tumpukan selesai, lihat tumpukan dengan nama BastionStackdan APGNS untuk mencatat informasi yang Anda butuhkan untuk terhubung ke database. Untuk informasi selengkapnya, lihat Melihat data AWS CloudFormation tumpukan dan sumber daya di AWS Management Console.

Langkah 3: Membuat klaster DB Aurora PostgreSQL

Anda dapat menggunakan aplikasi klien PostgreSQL standar untuk menghubungkan ke klaster DB. Dalam contoh ini, Anda terhubung ke klaster DB Aurora PostgreSQL menggunakan klien baris perintah psql.

Untuk menghubungkan ke klaster DB Aurora PostgreSQL

1. Temukan titik akhir (nama DNS) dan nomor port untuk instans penulis klaster DB Anda.

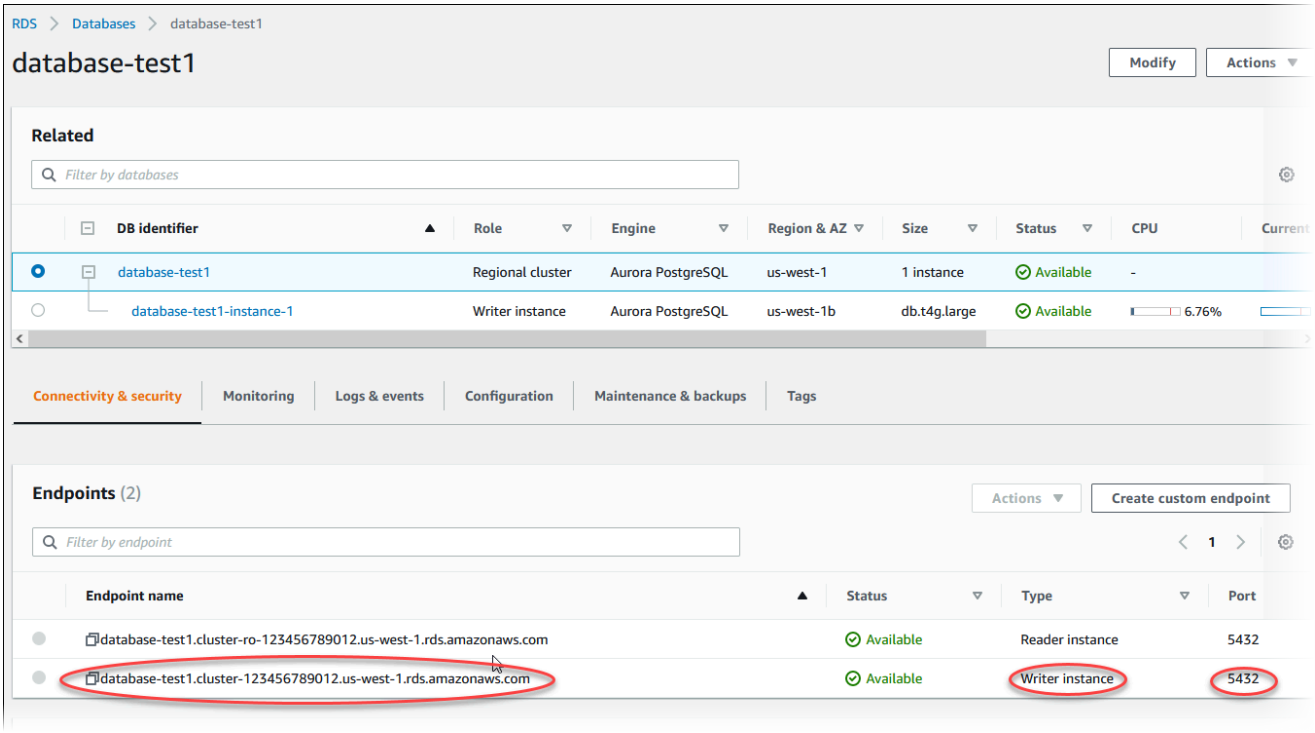
   a. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

   b. Di sudut kanan atas konsol Amazon RDS, pilih Wilayah AWS untuk cluster DB.

   c. Di panel navigasi, pilih Basis Data.

   d. Pilih nama klaster DB Aurora PostgreSQL untuk menampilkan detailnya.

   e. Di tab Konektivitas & keamanan, salin titik akhir instans penulis. Perhatikan juga nomor port. Anda memerlukan titik akhir dan nomor port untuk terhubung ke klaster DB.



2. Hubungkan ke instans EC2 yang Anda buat sebelumnya dengan mengikuti langkah-langkah di Menghubungkan ke instans Linux di Panduan Pengguna Amazon EC2 untuk Instans Linux.

   Sebaiknya Anda menghubungkan ke instans EC2 menggunakan SSH. Jika utilitas klien SSH diinstal di Windows, Linux, atau Mac, Anda dapat menghubungkan ke instans menggunakan format perintah berikut:

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

Misalnya, asumsikan bahwa `ec2-database-connect-key-pair.pem` disimpan di `/dir1` di Linux, dan DNS IPv4 publik untuk instans EC2 Anda adalah `ec2-12-345-678-90.compute-1.amazonaws.com`. Perintah SSH Anda akan tampak seperti berikut:

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

3. Dapatkan pembaruan keamanan dan perbaikan bug terbaru dengan memperbarui perangkat lunak di instans EC2 Anda. Untuk melakukannya, gunakan perintah berikut.

> **Note**
>
> Opsi `-y` menginstal pembaruan tanpa meminta konfirmasi. Hilangkan opsi ini untuk memeriksa pembaruan sebelum menginstal.

```
sudo dnf update -y
```

4. Instal klien baris perintah psql dari PostgreSQL di Amazon Linux 2023, menggunakan perintah berikut:

```
sudo dnf install postgresql15
```

5. Hubungkan ke klaster DB Aurora PostgreSQL. Misalnya, masukkan perintah berikut. Tindakan ini memungkinkan Anda terhubung ke klaster DB Aurora PostgreSQL menggunakan klien psql.

   Ganti titik akhir instans penulis untuk *`endpoint`*, ganti nama basis data `--dbname` yang ingin Anda hubungkan untuk *`postgres`*, dan ganti nama pengguna utama yang Anda gunakan untuk *`postgres`*. Masukkan kata sandi utama yang Anda gunakan saat dimintai kata sandi.

```
psql --host=endpoint --port=5432 --dbname=postgres --username=postgres
```

   Setelah Anda memasukkan kata sandi untuk pengguna, Anda akan melihat output yang serupa dengan yang berikut ini.

```
psql (14.3, server 14.6)
```

```
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.

postgres=>
```

Untuk informasi lebih lanjut tentang menghubungkan ke klaster DB Aurora PostgreSQL, lihat Menghubungkan ke klaster DB Amazon Aurora PostgreSQL.. Jika Anda tidak dapat terhubung ke klaster DB Anda, lihat Tidak dapat terhubung ke instans DB Amazon RDS.

Untuk keamanan, ini adalah praktik terbaik untuk menggunakan koneksi terenkripsi. Hanya gunakan koneksi PostgreSQL yang tidak terenkripsi saat klien dan server berada di VPC yang sama dan jaringan tepercaya. Untuk informasi tentang cara menggunakan koneksi terenkripsi, lihat Mengamankan data Aurora PostgreSQL dengan SSL/TLS.

6. Jalankan perintah SQL.

   Misalnya, perintah SQL berikut menunjukkan tanggal dan waktu saat ini:

```
SELECT CURRENT_TIMESTAMP;
```

Langkah 4: Hapus instans EC2 dan klaster DB

Setelah Anda terhubung ke dan menjelajahi instans EC2 dan klaster DB sampel yang Anda buat, hapus instans tersebut sehingga Anda tidak lagi dikenakan biaya untuk instans DB tersebut.

Jika Anda biasa AWS CloudFormation membuat sumber daya, lewati langkah ini dan lanjutkan ke langkah berikutnya.

Untuk menghapus instans EC2

1. Masuk ke AWS Management Console dan buka konsol Amazon EC2 di https://console.aws.amazon.com/ec2/.
2. Di panel navigasi, pilih Instans.
3. Pilih instans EC2, dan pilih Status instans, Akhiri instans.
4. Pilih Akhiri saat diminta untuk konfirmasi.

Untuk informasi selengkapnya tentang menghapus instans EC2, lihat Mengakhiri Instans Anda dalam Panduan Pengguna Amazon EC2 untuk Instans Linux.

Untuk menghapus klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data lalu pilih instans DB yang terkait dengan klaster DB.
3. Untuk Tindakan, pilih Hapus.
4. Pilih Hapus.

   Setelah semua instans DB yang terkait dengan klaster DB dihapus, klaster DB akan dihapus secara otomatis.

(Opsional) Hapus instans EC2 dan cluster DB yang dibuat dengan CloudFormation

Jika Anda biasa AWS CloudFormation membuat sumber daya, hapus CloudFormation tumpukan setelah Anda terhubung dan jelajahi contoh instans EC2 dan kluster DB, sehingga Anda tidak lagi dikenakan biaya untuk itu.

Untuk menghapus sumber CloudFormation daya

1. Buka AWS CloudFormation konsol.
2. Pada halaman Stacks di CloudFormation konsol, pilih tumpukan root (tumpukan tanpa nama VPCStack, BastionStack atau APGNS).
3. Pilih Hapus.
4. Pilih Hapus tumpukan saat diminta konfirmasi.

Untuk informasi selengkapnya tentang menghapus tumpukan CloudFormation, lihat Menghapus tumpukan di AWS CloudFormation konsol di AWS CloudFormation Panduan Pengguna.

(Opsional) Menghubungkan klaster DB Anda ke fungsi Lambda

Anda juga dapat menghubungkan cluster DB PostgreSQL Aurora Anda ke sumber daya komputasi tanpa server Lambda. Fungsi Lambda memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola infrastruktur. Fungsi Lambda juga memungkinkan Anda untuk otomatis merespons

permintaan eksekusi kode pada skala apa pun, mulai dari selusin peristiwa dalam sehari hingga ratusan per detik. Lihat informasi yang lebih lengkap di Menghubungkan secara otomatis fungsi Lambda dan klaster DB Aurora.

Tutorial: Membuat server web dan klaster DB Amazon Aurora

Tutorial ini menunjukkan cara menginstal server web Apache dengan PHP dan membuat basis data MariaDB, MySQL, atau PostgreSQL. Server web berjalan di instans Amazon EC2 menggunakan Amazon Linux 2023, dan Anda dapat memilih antara klaster DB Aurora MySQL atau Aurora PostgreSQL. Baik instans Amazon EC2 maupun klaster DB berjalan di cloud privat virtual (VPC) berdasarkan layanan Amazon VPC.

Important

Tidak ada biaya untuk membuat akun AWS. Namun, dengan menyelesaikan tutorial ini, Anda mungkin akan dikenaki biaya untuk sumber daya AWS yang digunakan. Anda dapat menghapus sumber daya ini setelah menyelesaikan tutorial jika tidak diperlukan lagi.

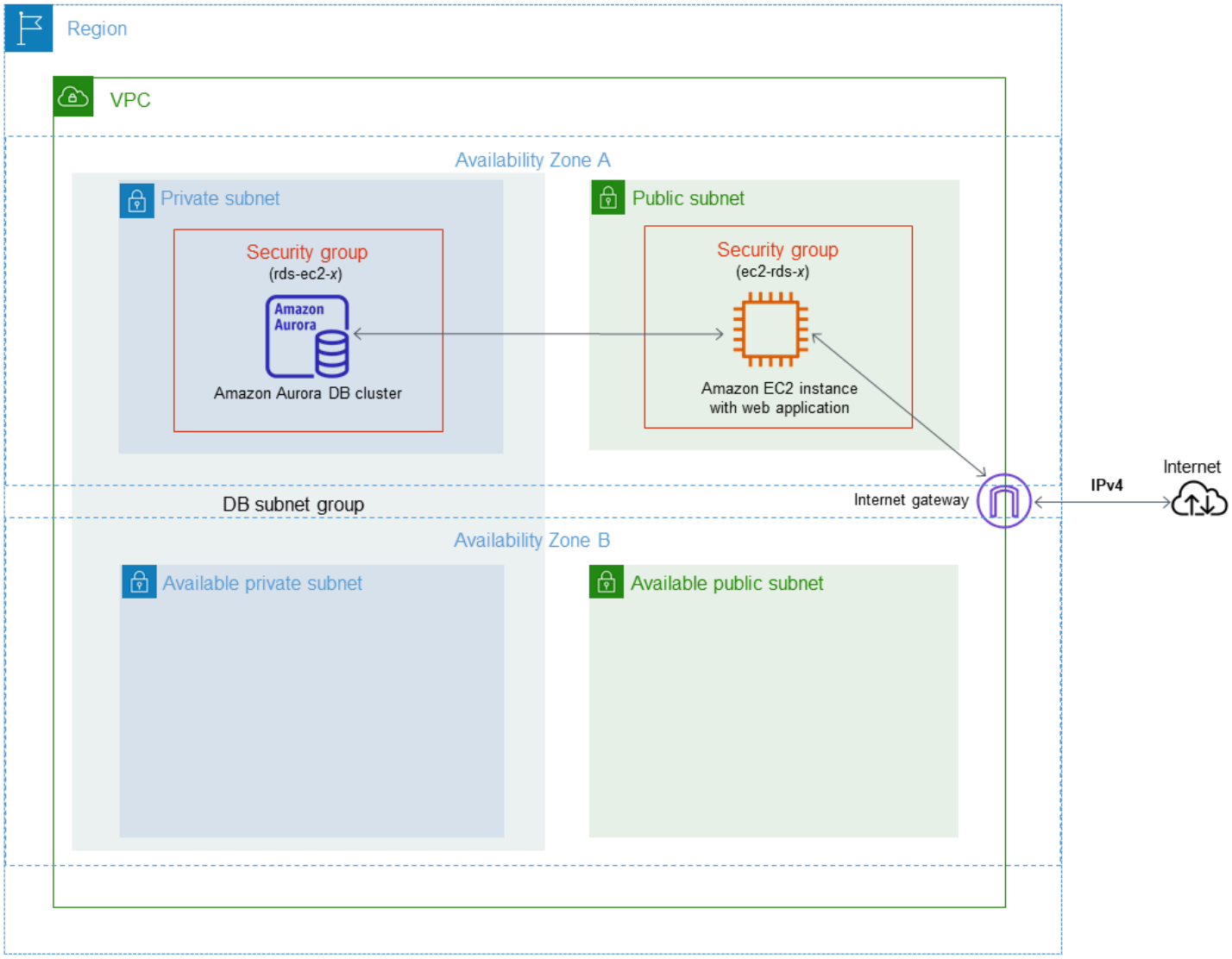
Note

Tutorial ini berfungsi dengan Amazon Linux 2023 dan mungkin tidak berfungsi untuk versi Linux lainnya.

Dalam tutorial berikut, Anda membuat instans EC2 yang menggunakan VPC, subnet, dan grup keamanan default untuk Akun AWS Anda. Tutorial ini menunjukkan cara membuat klaster DB dan secara otomatis menyiapkan konektivitas dengan instans EC2 yang Anda buat. Tutorial kemudian menunjukkan cara menginstal server web pada instans EC2. Anda menghubungkan server web ke klaster DB di VPC menggunakan titik akhir penulis klaster DB.

1. Meluncurkan instans EC2
2. Membuat klaster DB Amazon Aurora
3. Menginstal server web di instans EC2 Anda

Diagram berikut menunjukkan konfigurasi setelah tutorial selesai.



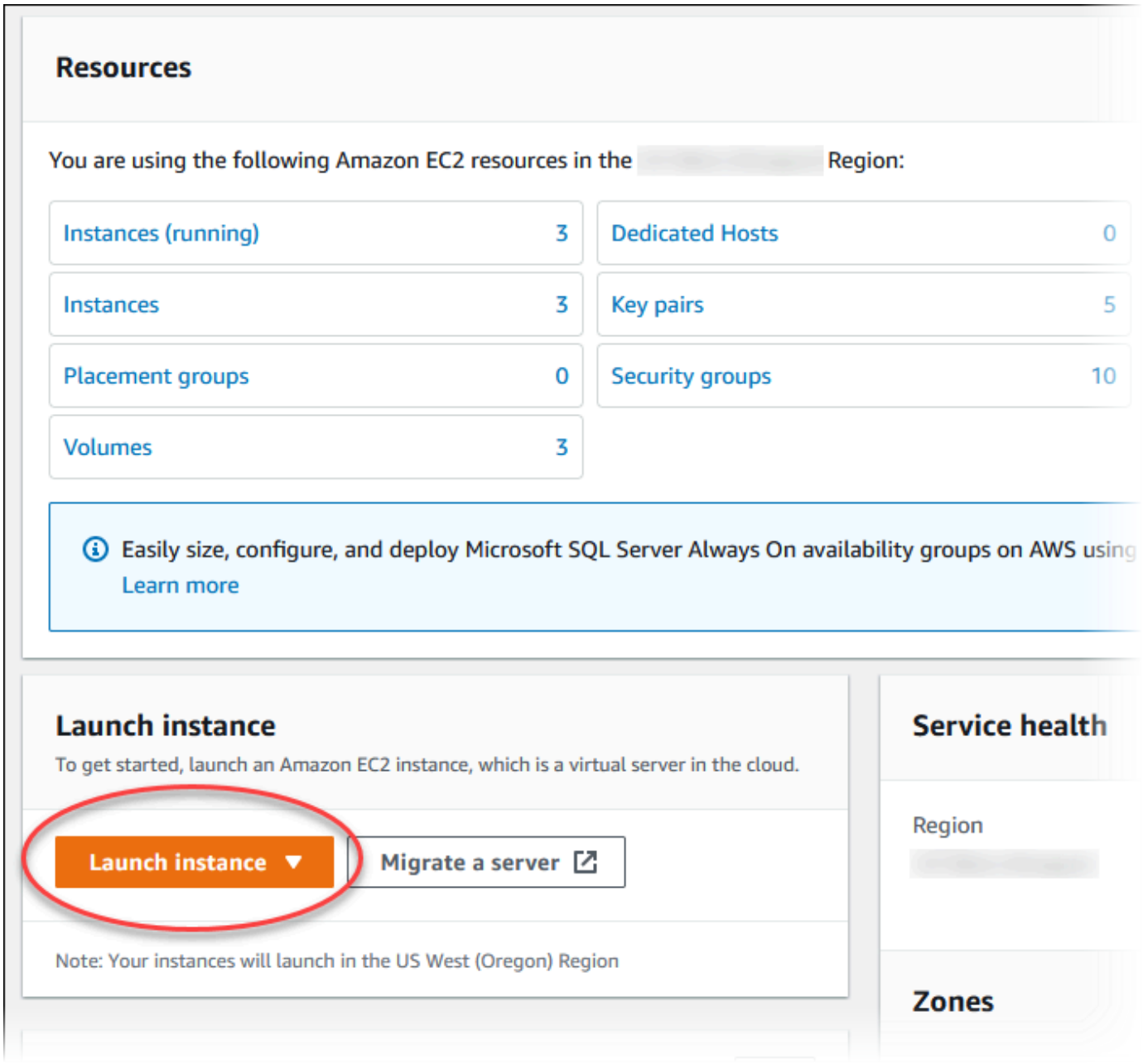
> **Note**
>
> Setelah Anda menyelesaikan tutorial, ada subnet publik dan privat di setiap Zona Ketersediaan di VPC Anda. Tutorial ini menggunakan VPC default untuk Akun AWS Anda dan secara otomatis menyiapkan konektivitas antara instans EC2 dan klaster DB. Jika Anda lebih suka mengonfigurasi VPC baru untuk skenario ini, selesaikan tugas di Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4).

Meluncurkan instans EC2

Buat instans Amazon EC2 di subnet publik VPC Anda.

Meluncurkan instans EC2

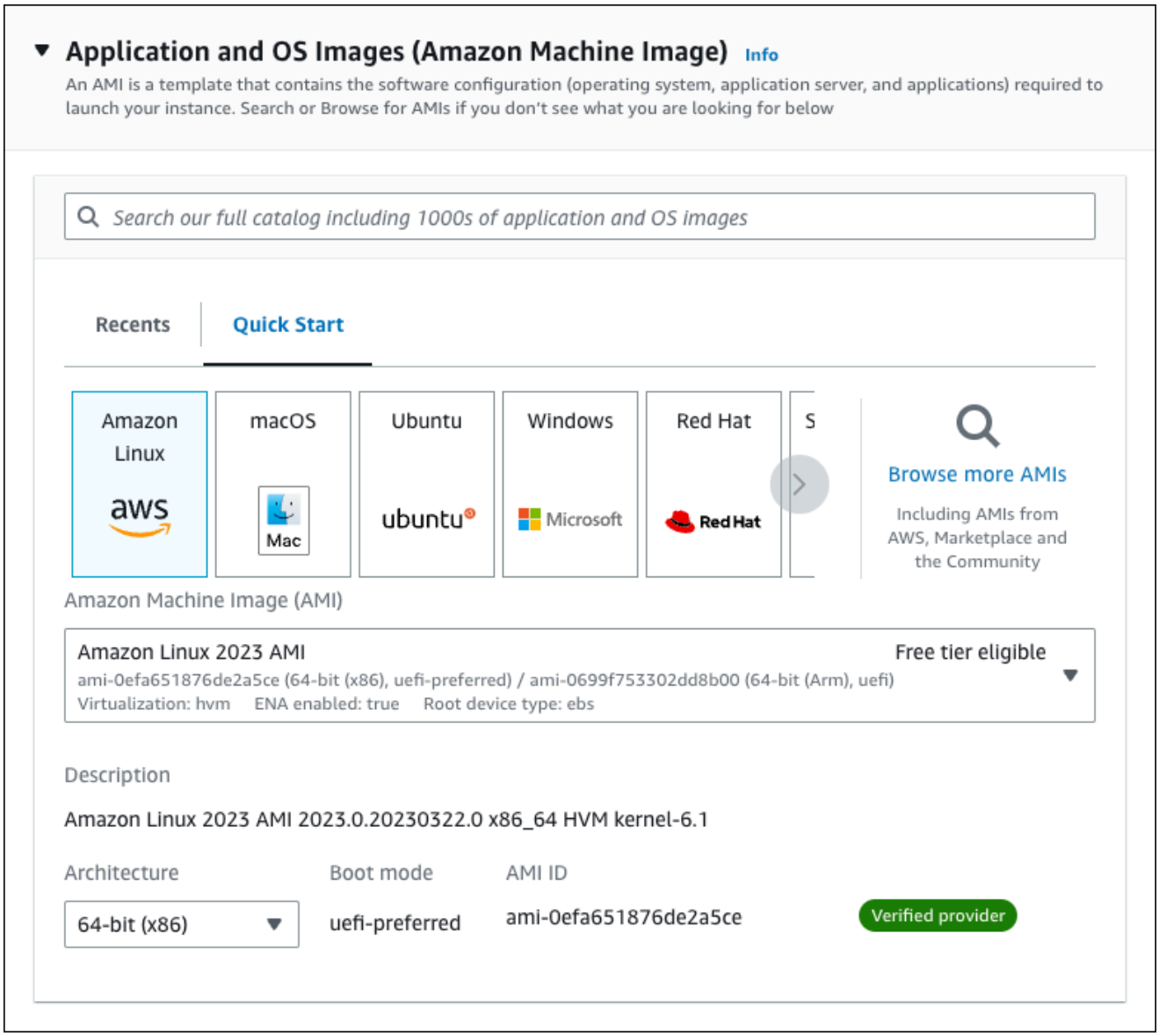
1. Masuk ke AWS Management Console dan buka konsol Amazon EC2 di https://console.aws.amazon.com/ec2/.
2. Di pojok kanan atas AWS Management Console, pilih Wilayah AWS tempat Anda akan membuat instans EC2.
3. Pilih Dasbor EC2, lalu pilih Luncurkan instans seperti yang ditampilkan berikut.



4. Pilih pengaturan berikut di halaman Meluncurkan instans.

a. Pada Nama dan tag, untuk Nama, masukkan **`tutorial-ec2-instance-web-server`**.

b. Pada Gambar Aplikasi dan OS (Amazon Machine Image), pilih Amazon Linux, lalu pilih AMI Amazon Linux 2023. Biarkan default untuk pilihan lain.



c. Di bagian Jenis instans, pilih t2.micro.

d. Di bagian Pasangan kunci (login), pilih Nama pasangan kunci untuk menggunakan pasangan kunci yang ada. Untuk membuat pasangan kunci baru untuk instans Amazon EC2, pilih Buat Pasangan kunci baru lalu gunakan jendela Buat pasangan kunci untuk membuatnya.

   Untuk informasi selengkapnya tentang membuat pasangan kunci baru, lihat Membuat pasangan kunci di Panduan Pengguna Amazon EC2 untuk Instans Linux.

e. Di bagian Pengaturan jaringan, atur nilai-nilai ini dan biarkan nilai-nilai lainnya sebagai default:

- Untuk Izinkan lalu lintas SSH dari, pilih sumber koneksi SSH ke instans EC2.

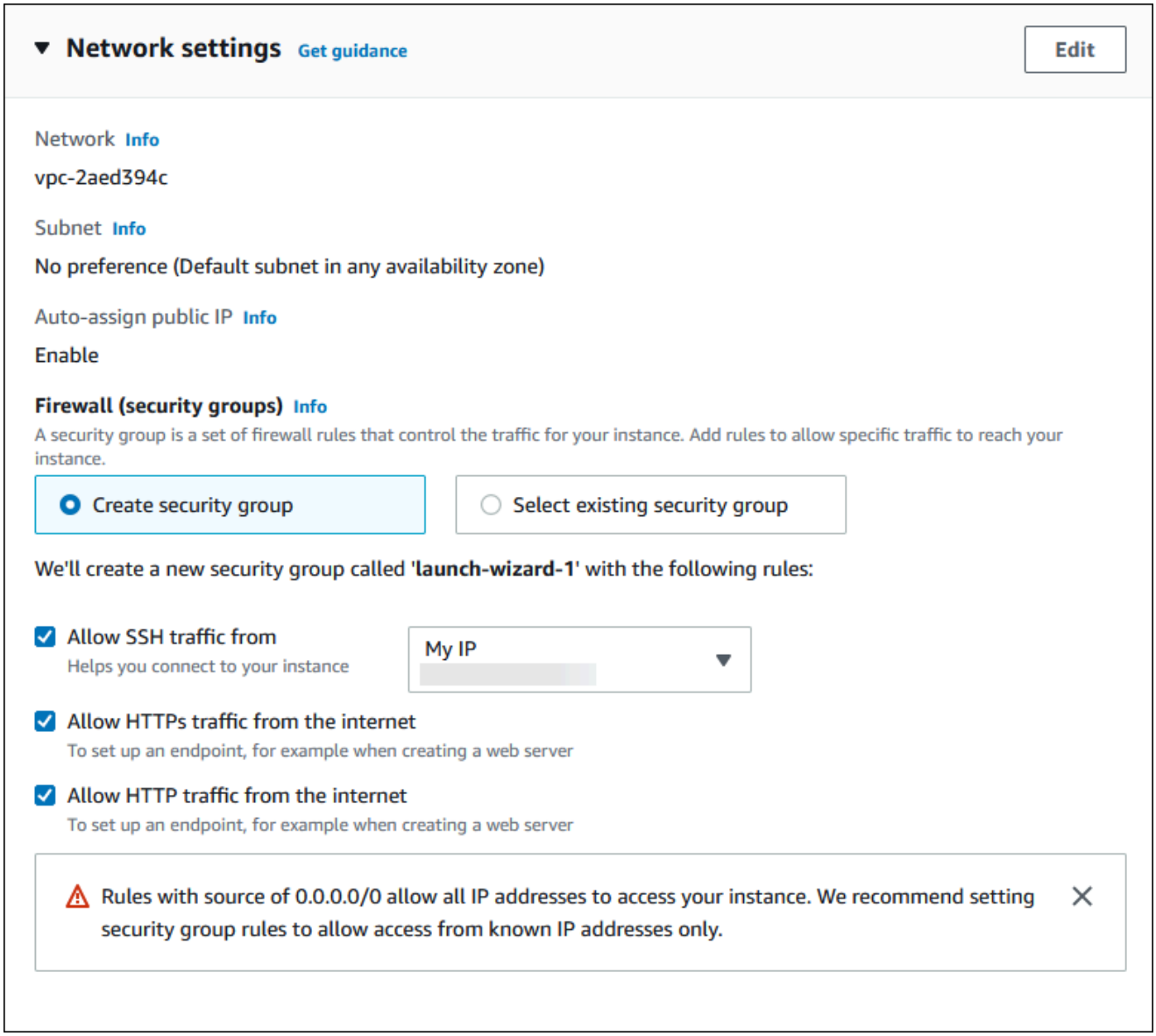
  Anda dapat memilih IP Saya jika alamat IP yang ditampilkan benar untuk koneksi SSH.

  Jika tidak, Anda dapat menentukan alamat IP yang akan digunakan untuk menghubungkan ke instans EC2 di VPC Anda menggunakan Secure Shell (SSH). Untuk menentukan alamat IP publik Anda, Anda dapat menggunakan layanan di https://checkip.amazonaws.com di jendela atau tab browser lain. Contoh alamat IP adalah `203.0.113.25/32`.

  Dalam banyak kasus, Anda dapat menghubungkan melalui penyedia layanan Internet (ISP) atau dari belakang firewall Anda tanpa alamat IP statis. Jika demikian, tentukan rentang alamat IP yang digunakan oleh komputer klien.

  > **Warning**
  >
  > Jika menggunakan `0.0.0.0/0` untuk akses SSH, Anda memungkinkan semua alamat IP untuk mengakses instans publik Anda menggunakan SSH. Pendekatan ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, Anda hanya dapat memberikan otorisasi pada alamat IP atau rentang alamat tertentu saja untuk mengakses instans Anda menggunakan SSH.

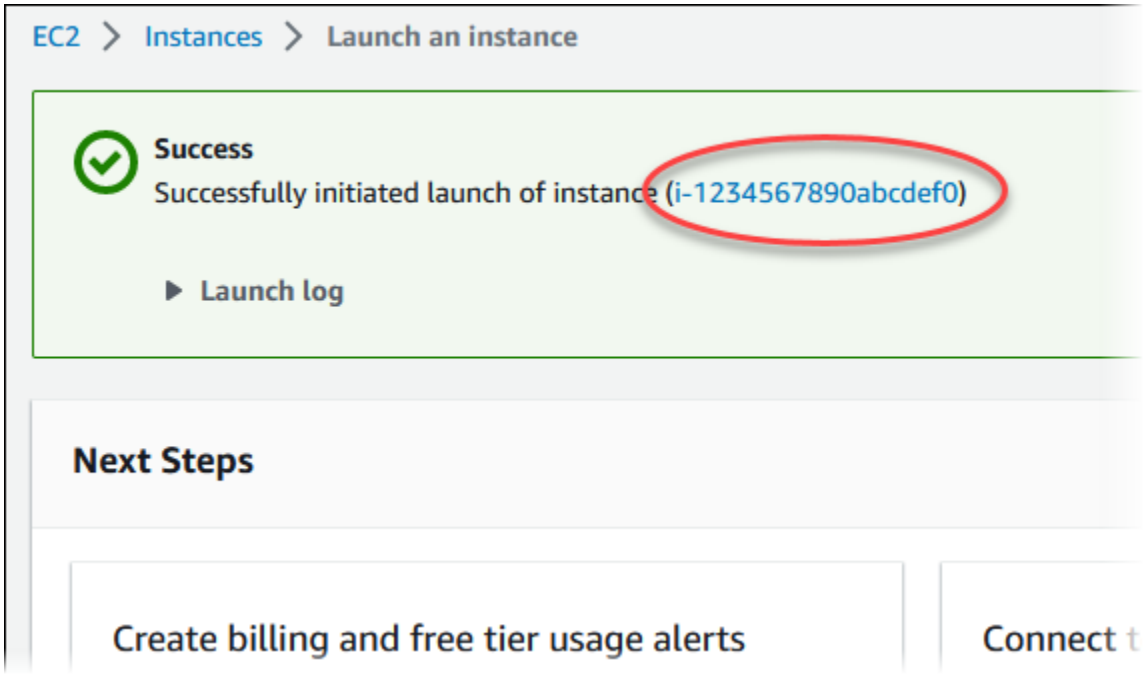
- Aktifkan Izinkan lalu lintas HTTPS dari internet.
- Aktifkan Izinkan lalu lintas HTTP dari internet.



f. Biarkan nilai default untuk bagian yang lainnya.

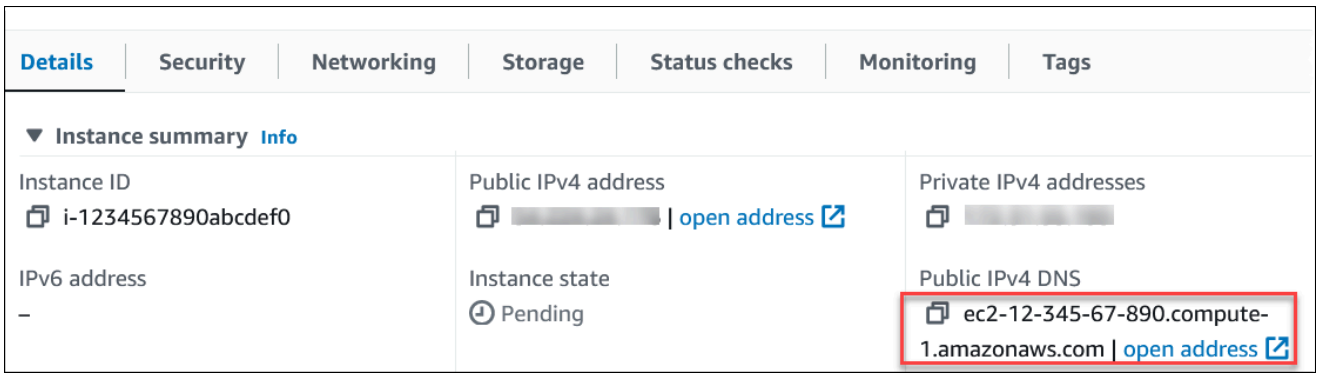
g. Tinjau ringkasan konfigurasi instans di panel Ringkasan, dan ketika Anda siap, pilih Luncurkan instans.

5. Di halaman Status Peluncuran, catat pengidentifikasi untuk instans EC2 baru Anda, misalnya: `i-1234567890abcdef0`.

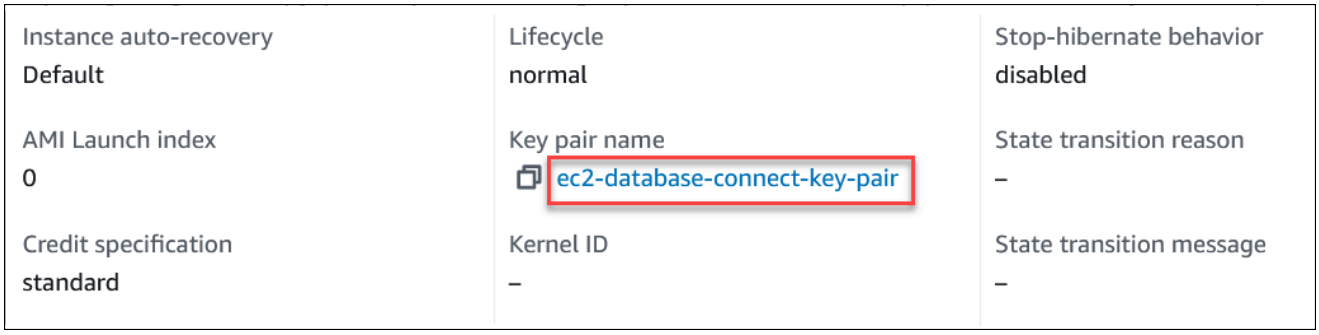


6. Pilih pengidentifikasi instans EC2 untuk membuka daftar instans EC2, lalu pilih instans EC2 Anda.
7. Di tab Detail, catat nilai-nilai berikut, yang akan Anda butuhkan saat menghubungkan menggunakan SSH:

   a. Di Ringkasan instans, catat nilai untuk DNS IPv4 Publik.



   b. Di Detail instans, catat nilai untuk Nama pasangan kunci.



8. Tunggu hingga status instans untuk instans Anda Berjalan sebelum melanjutkan.
9. Selesaikan Membuat klaster DB Amazon Aurora.

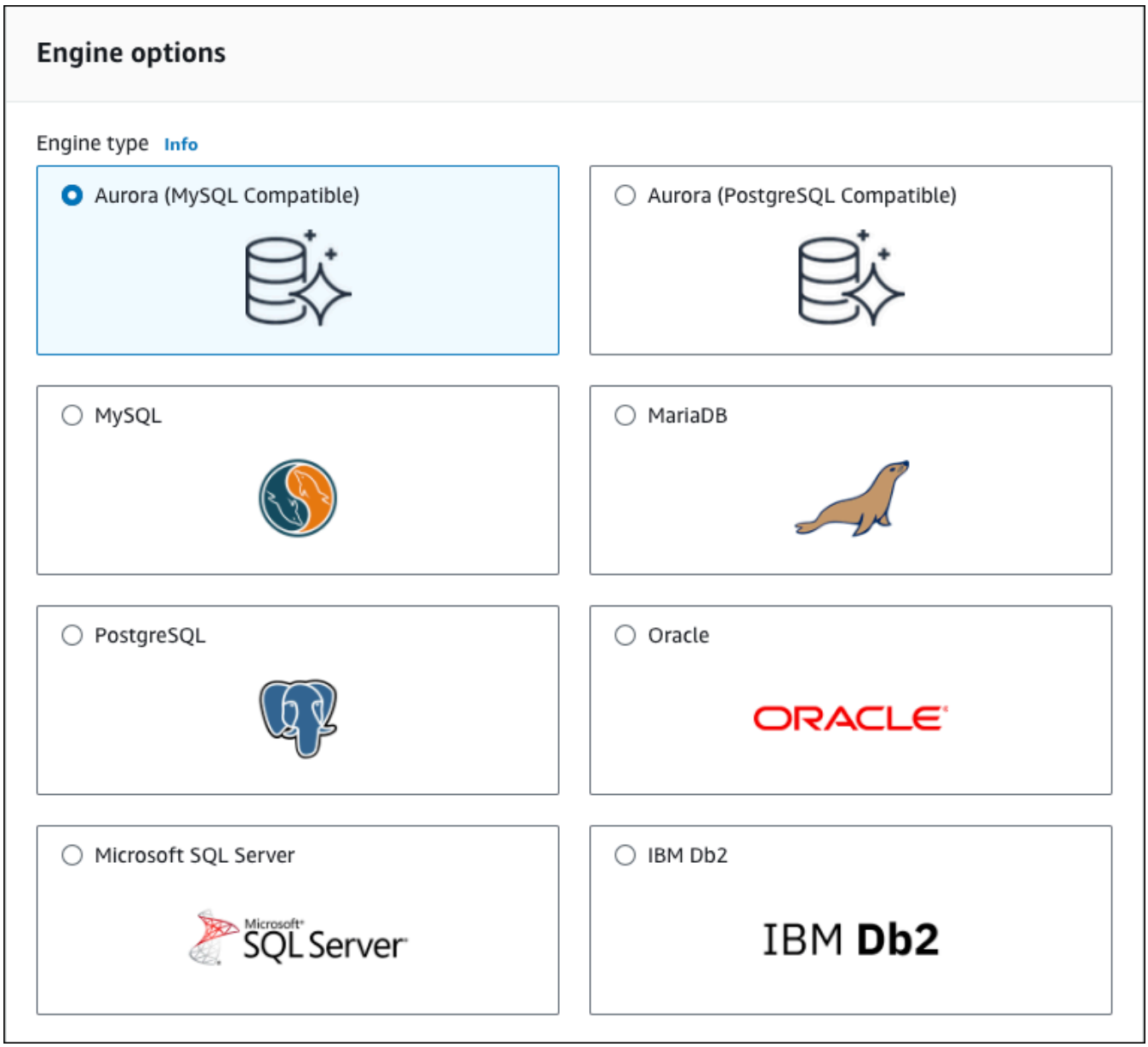
Membuat klaster DB Amazon Aurora

Buat klaster DB Amazon Aurora MySQL atau Aurora PostgreSQL DB yang mempertahankan data yang digunakan oleh aplikasi web.

Aurora MySQL

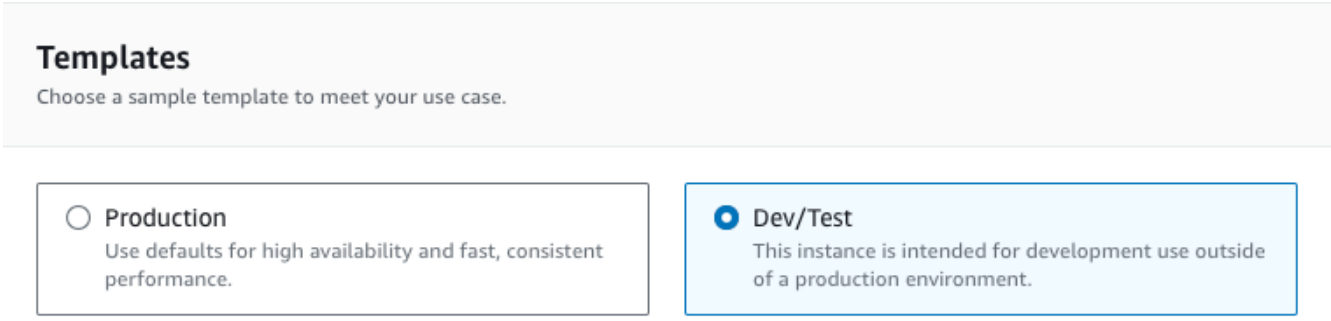
Untuk membuat klaster DB Aurora MySQL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas AWS Management Console, pastikan Wilayah AWS sama dengan tempat Anda membuat instans EC2.
3. Di panel navigasi, pilih Basis Data.
4. Pilih Buat basis data.
5. Di halaman Buat basis data, pilih Pembuatan standar.
6. Untuk Opsi mesin, pilih Aurora (Kompatibel dengan MySQL).



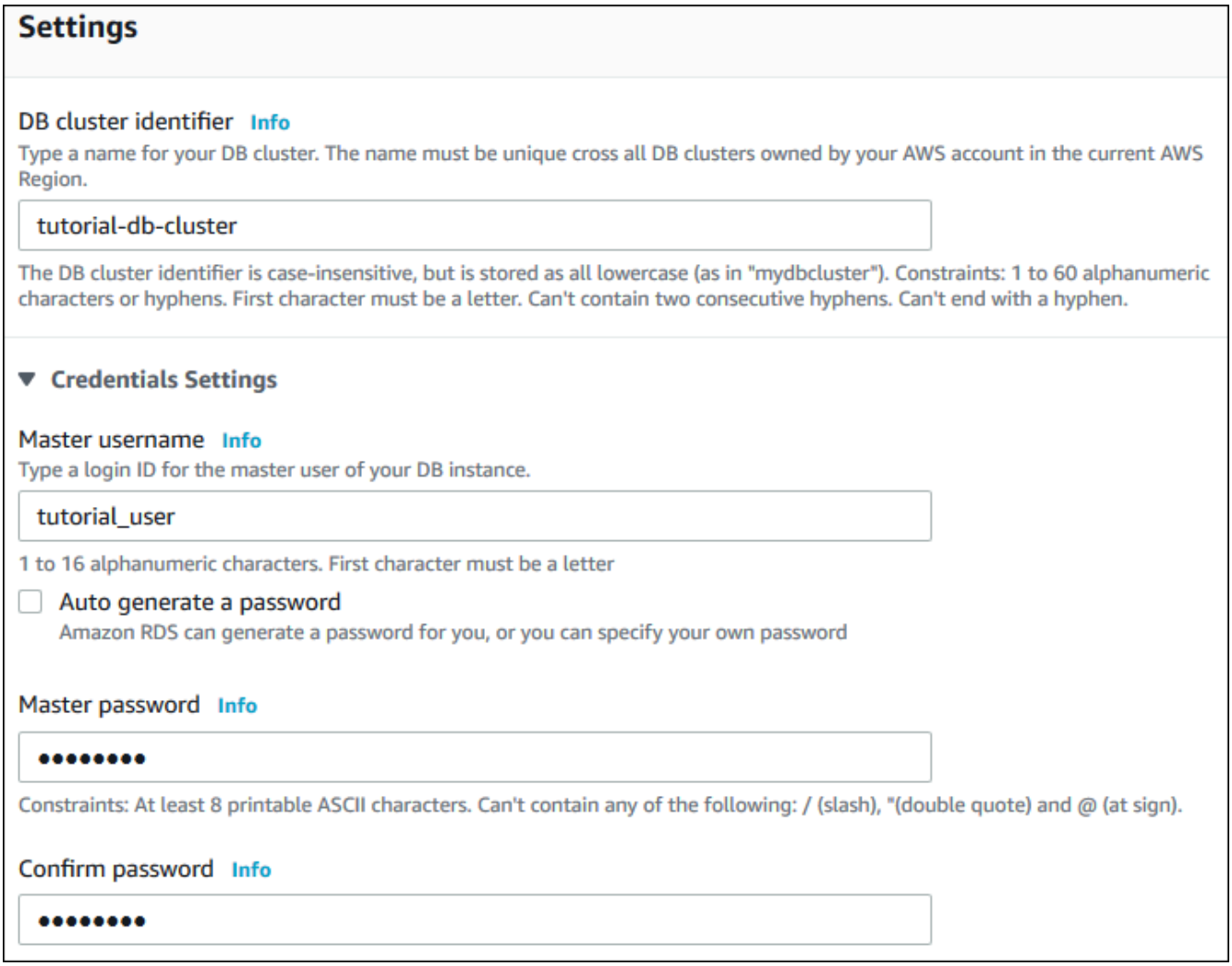
Pertahankan nilai default untuk Versi dan opsi mesin lain.

7. Di bagian Templat, pilih Dev/Test.



8. Di bagian Pengaturan, atur nilai-nilai ini:

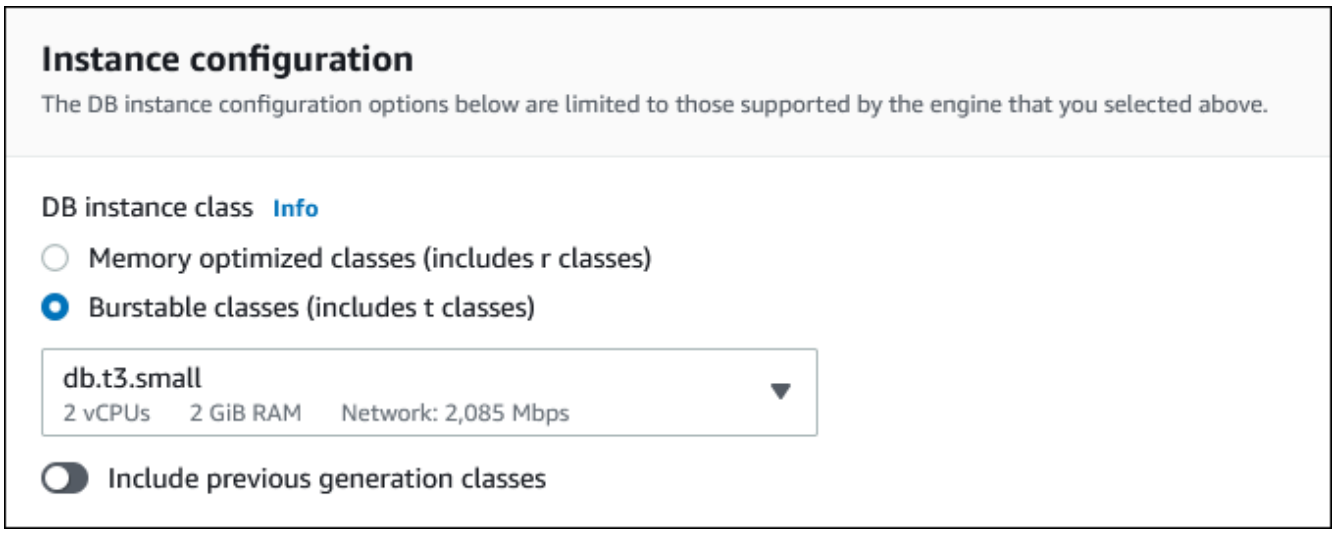
   - Pengidentifikasi klaster DB – Ketikkan **`tutorial-db-cluster`**.
   - Nama pengguna utama – Ketikkan **`tutorial_user`**.
   - Buat kata sandi secara otomatis – Biarkan opsi nonaktif.
   - Kata sandi utama – Ketikkan kata sandi.
   - Konfirmasi kata sandi – Ketik ulang kata sandi.



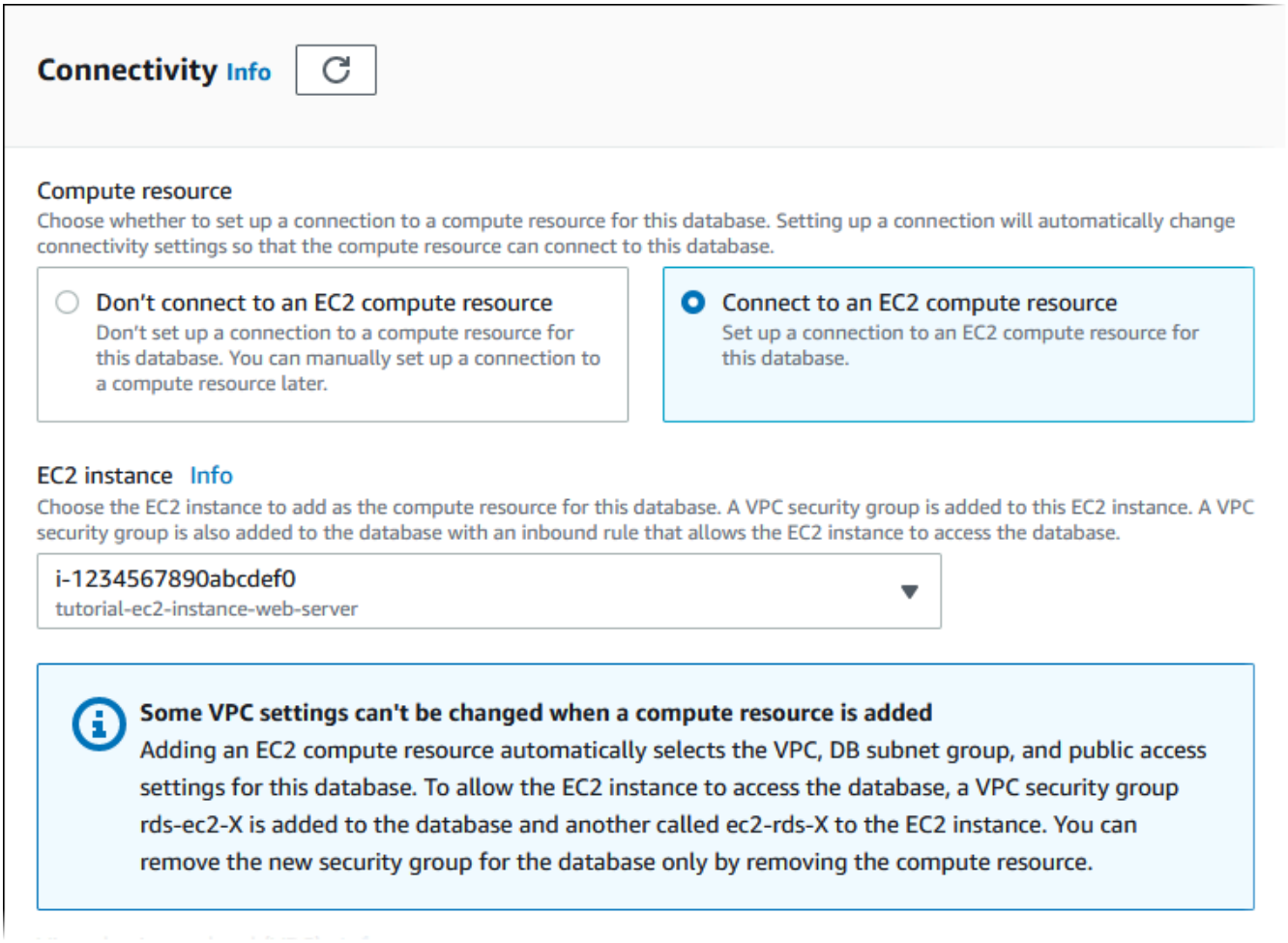
9. Di bagian Konfigurasi instans, atur nilai-nilai ini:

   - Kelas runtutan (termasuk kelas t)
   - db.t3.small atau db.t3.medium

> **Note**
>
> Sebaiknya hanya gunakan kelas instans DB T untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Jenis kelas instans DB.

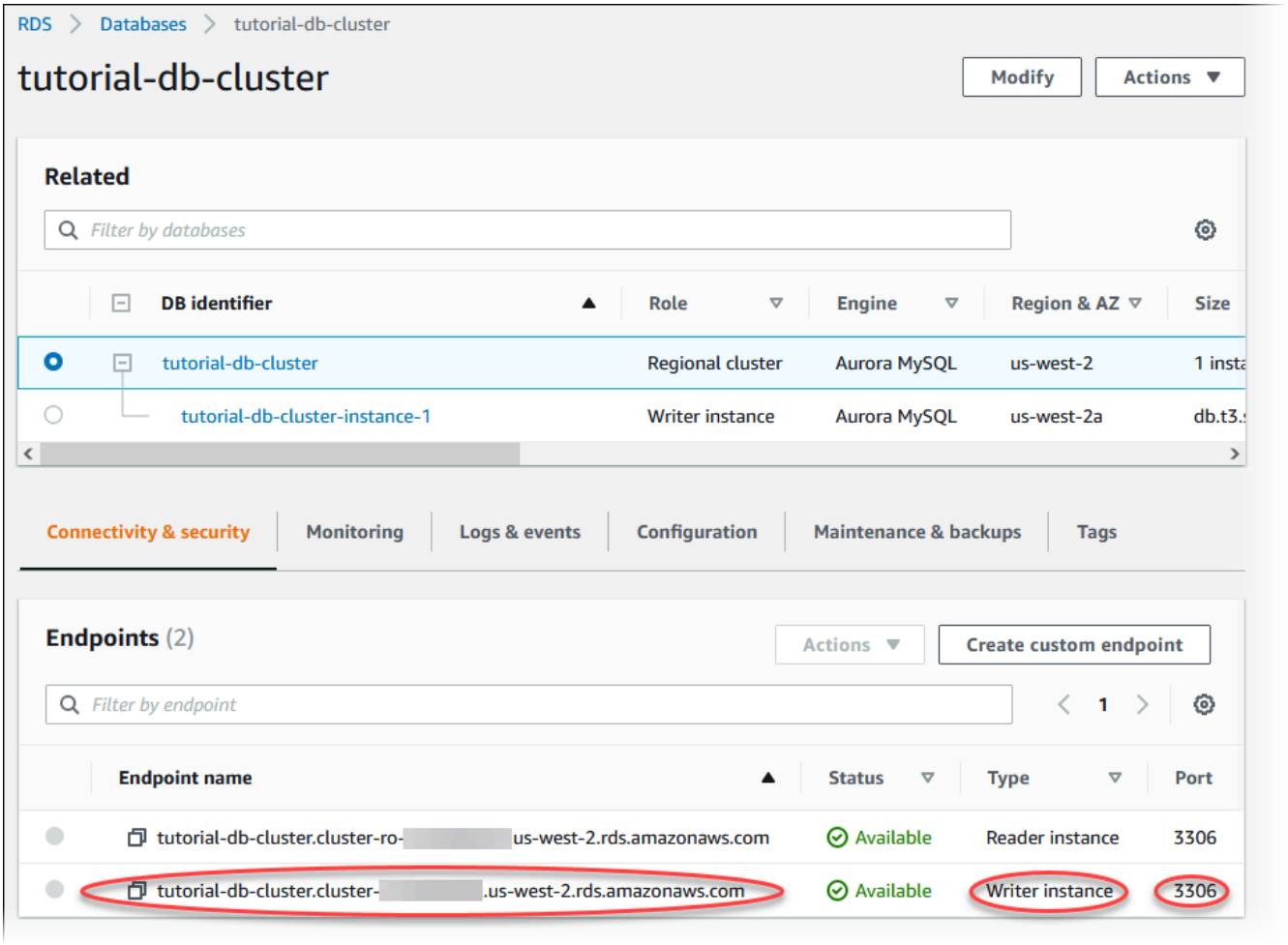


10. Di bagian Ketersediaan dan daya tahan, gunakan nilai default.
11. Di bagian Konektivitas, atur nilai-nilai ini dan biarkan nilai lainnya sebagai default:
    - Untuk sumber daya Komputasi, pilih Hubungkan ke sumber daya komputasi EC2.
    - Untuk contoh EC2, pilih instans EC2 yang Anda buat sebelumnya, seperti tutorial-ec2 -. instance-web-server



12. Buka bagian Konfigurasi tambahan, dan masukkan **`sample`** untuk Nama database awal. Biarkan opsi lainnya menggunakan pengaturan default.
13. Untuk membuat klaster DB Aurora MySQL, pilih Buat basis data.

    Klaster DB baru Anda muncul di daftar Basis Data dengan status Membuat.
14. Tunggu sampai Status klaster DB baru Anda menampilkan status Tersedia. Lalu pilih nama klaster DB untuk menampilkan detailnya.
15. Di bagian Konektivitas & keamanan, lihat Titik Akhir dan Port instans DB tulis.



Catat titik akhir dan port untuk instans DB tulis Anda. Gunakan informasi ini untuk menghubungkan server web Anda ke klaster DB Anda.

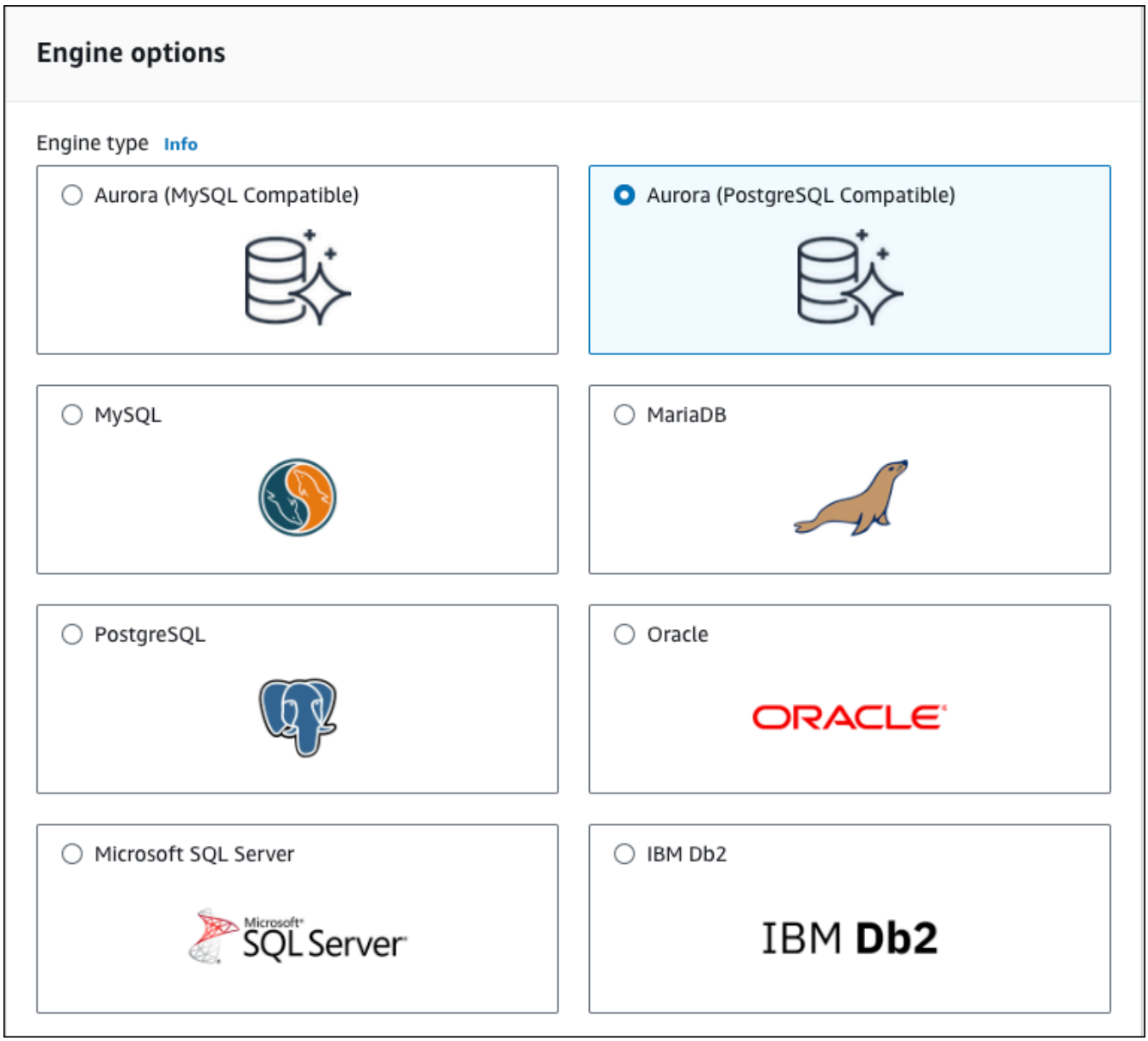
16. Selesaikan Menginstal server web di instans EC2 Anda.

Aurora PostgreSQL

Untuk membuat klaster DB Aurora PostgreSQL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas AWS Management Console, pastikan Wilayah AWS sama dengan tempat Anda membuat instans EC2.
3. Di panel navigasi, pilih Basis Data.
4. Pilih Buat basis data.

5. Di halaman Buat basis data, pilih Pembuatan standar.
6. Untuk Opsi mesin, pilih Aurora (Kompatibel dengan PostgreSQL).

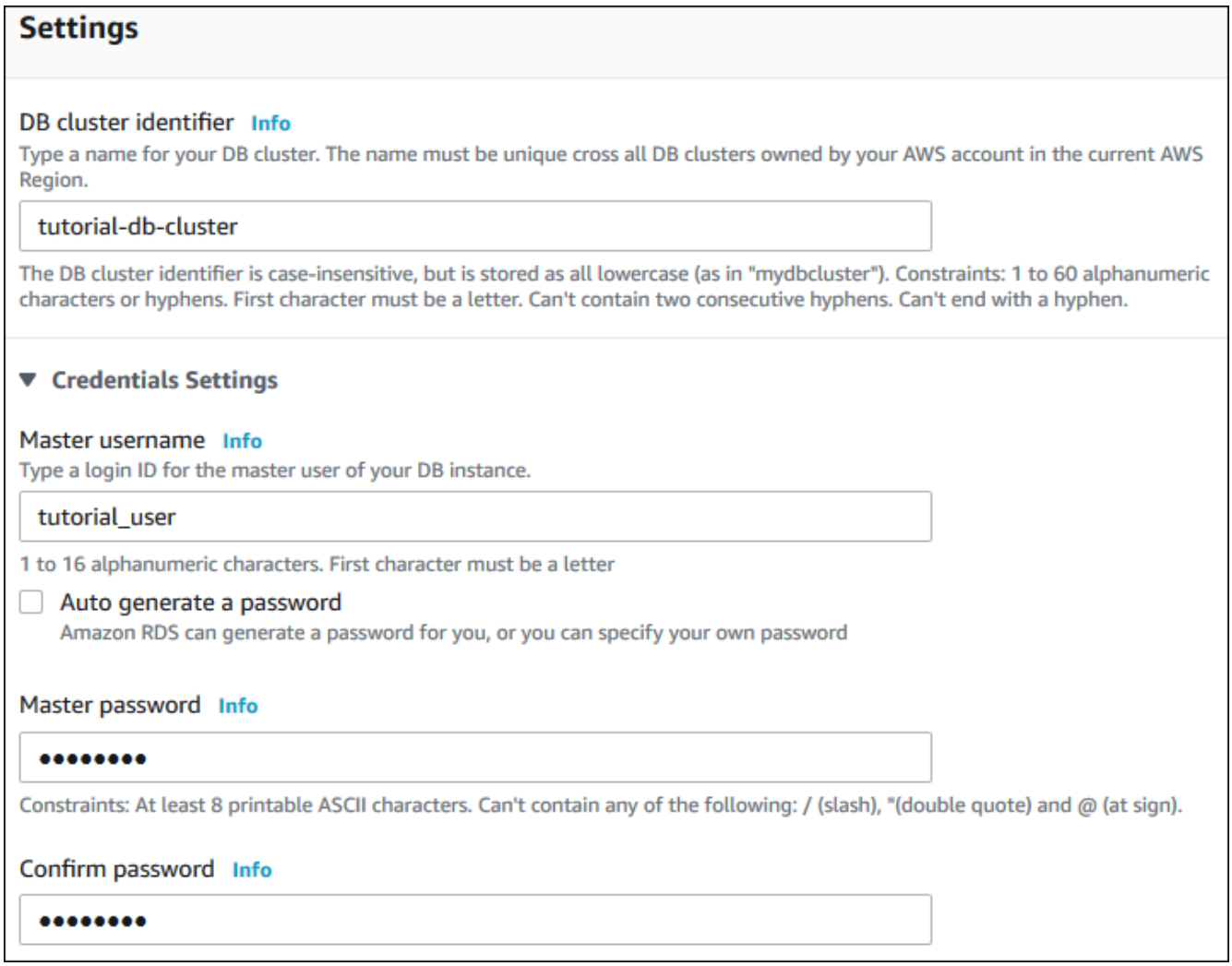


Pertahankan nilai default untuk Versi dan opsi mesin lain.

7. Di bagian Templat, pilih Dev/Test.

8. Di bagian Pengaturan, atur nilai-nilai ini:

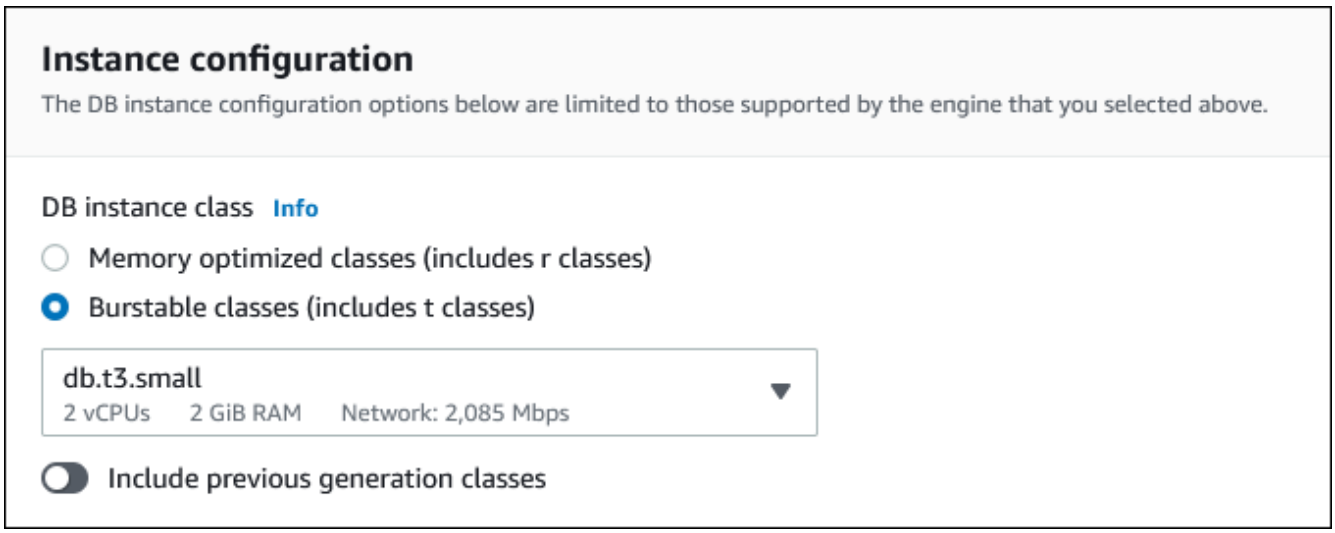
   - Pengidentifikasi klaster DB – Ketikkan **`tutorial-db-cluster`**.
   - Nama pengguna utama – Ketikkan **`tutorial_user`**.
   - Buat kata sandi secara otomatis – Biarkan opsi nonaktif.
   - Kata sandi utama – Ketikkan kata sandi.
   - Konfirmasi kata sandi – Ketik ulang kata sandi.



9. Di bagian Konfigurasi instans, atur nilai-nilai ini:

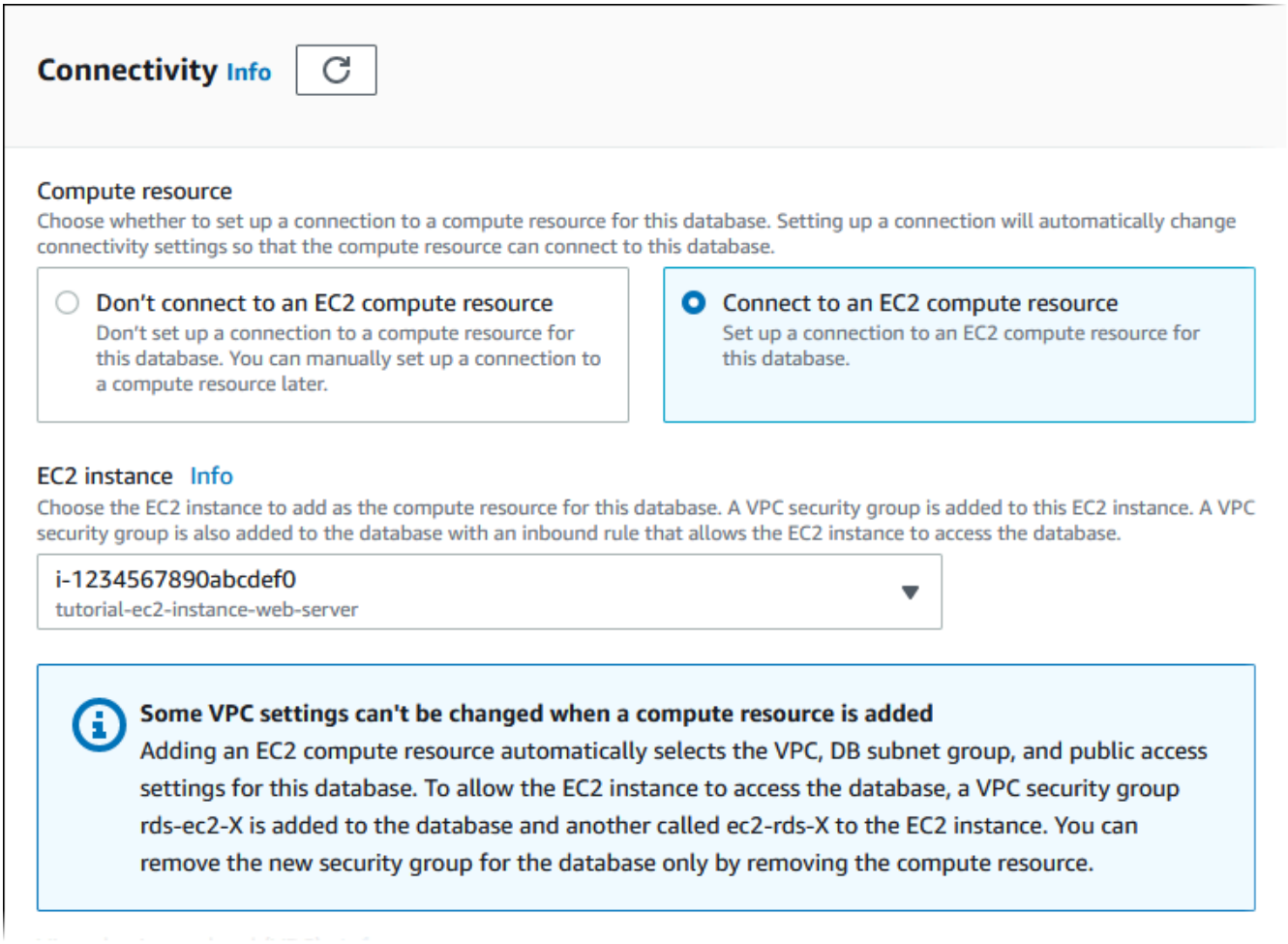
   - Kelas runtutan (termasuk kelas t)
   - db.t3.small atau db.t3.medium

> **Note**
>
> Sebaiknya hanya gunakan kelas instans DB T untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Jenis kelas instans DB.



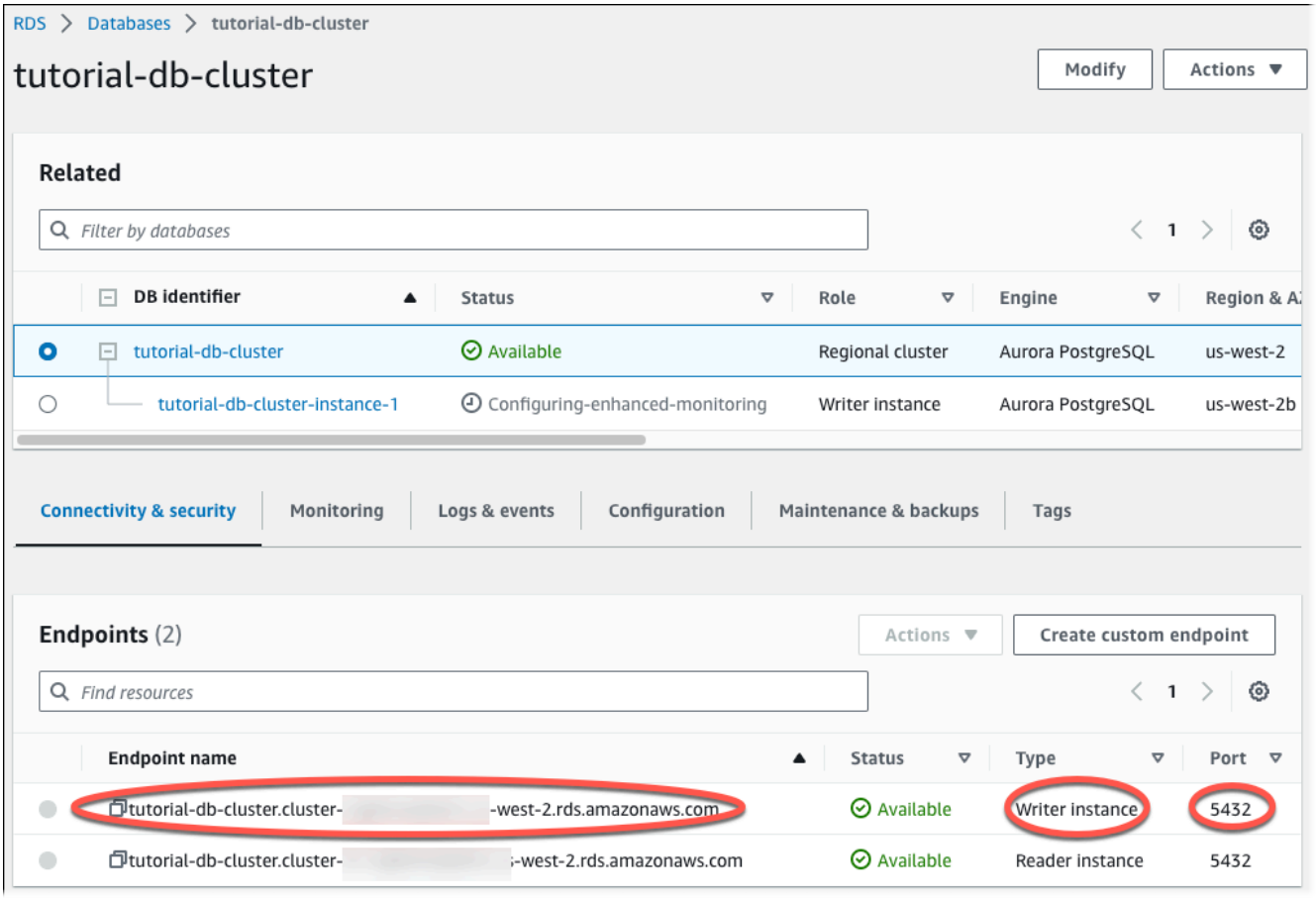
10. Di bagian Ketersediaan dan daya tahan, gunakan nilai default.
11. Di bagian Konektivitas, atur nilai-nilai ini dan biarkan nilai lainnya sebagai default:

    - Untuk sumber daya Komputasi, pilih Hubungkan ke sumber daya komputasi EC2.
    - Untuk contoh EC2, pilih instans EC2 yang Anda buat sebelumnya, seperti tutorial-ec2 -. instance-web-server



12. Buka bagian Konfigurasi tambahan, dan masukkan **`sample`** untuk Nama database awal. Biarkan opsi lainnya menggunakan pengaturan default.
13. Untuk membuat klaster DB Aurora PostgreSQL, pilih Buat basis data.

    Klaster DB baru Anda muncul di daftar Basis Data dengan status Membuat.
14. Tunggu sampai Status klaster DB baru Anda menampilkan status Tersedia. Lalu pilih nama klaster DB untuk menampilkan detailnya.
15. Di bagian Konektivitas & keamanan, lihat Titik Akhir dan Port instans DB tulis.



Catat titik akhir dan port untuk instans DB tulis Anda. Gunakan informasi ini untuk menghubungkan server web Anda ke klaster DB Anda.

16. Selesaikan Menginstal server web di instans EC2 Anda.

Menginstal server web di instans EC2 Anda

Instal server web pada instans EC2 yang Anda buat di Meluncurkan instans EC2. Server web terhubung ke klaster DB Amazon Aurora yang Anda buat di Membuat klaster DB Amazon Aurora.

Menginstal server web Apache dengan PHP dan MariaDB

Hubungkan ke instans EC2 Anda dan instal server web.

Menghubungkan ke instans EC2 dan menginstal server web Apache dengan PHP

1. Hubungkan ke instans EC2 yang Anda buat sebelumnya dengan mengikuti langkah-langkah di Menghubungkan ke instans Linux di Panduan Pengguna Amazon EC2 untuk Instans Linux.

Kami merekomendasikan agar Anda menghubungkan ke instans EC2 menggunakan SSH. Jika utilitas klien SSH diinstal di Windows, Linux, atau Mac, Anda dapat menghubungkan ke instans menggunakan format perintah berikut:

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

Misalnya, asumsikan bahwa `ec2-database-connect-key-pair.pem` disimpan di `/dir1` di Linux, dan DNS IPv4 publik untuk instans EC2 Anda adalah `ec2-12-345-678-90.compute-1.amazonaws.com`. Perintah SSH Anda akan tampak seperti berikut:

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

2. Dapatkan perbaikan bug terbaru dan pembaruan keamanan dengan memperbarui perangkat lunak di instans EC2 Anda. Untuk melakukannya, gunakan perintah berikut.

> **Note**
>
> Opsi `-y` menginstal pembaruan tanpa meminta konfirmasi. Hilangkan opsi ini untuk memeriksa pembaruan sebelum menginstal.

```
sudo dnf update -y
```

3. Setelah pembaruan selesai, instal server web Apache, PHP, dan MariaDB atau perangkat lunak PostgreSQL menggunakan perintah berikut. Perintah ini menginstal beberapa paket perangkat lunak dan dependensi terkait secara bersamaan.

   MariaDB & MySQL

   ```
   sudo dnf install -y httpd php php-mysqli mariadb105
   ```

   PostgreSQL

   ```
   sudo dnf install -y httpd php php-pgsql postgresql15
   ```

Jika Anda mengalami kesalahan, instans Anda mungkin tidak diluncurkan dengan AMI Amazon Linux 2023. Sebaiknya gunakan AMI Amazon Linux 2 AMI. Anda dapat melihat versi Amazon Linux Anda menggunakan perintah berikut.

```
cat /etc/system-release
```

Untuk informasi selengkapnya, lihat Memperbarui perangkat lunak instans.

4. Mulai server web dengan perintah yang ditampilkan berikut.

```
sudo systemctl start httpd
```

Anda dapat menguji apakah server web Anda terinstal dan berjalan dengan benar. Untuk melakukannya, masukkan nama Sistem Nama Domain (DNS) publik dari instans EC2 Anda di bilah alamat browser web, misalnya: `http://ec2-42-8-168-21.us-west-1.compute.amazonaws.com`. Jika server web Anda berjalan, Anda akan melihat halaman uji Apache.

Jika Anda tidak melihat halaman uji Apache, periksa aturan masuk Anda untuk grup keamanan VPC yang Anda buat di Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4). Pastikan aturan masuk Anda menyertakan aturan yang mengizinkan akses HTTP (port 80) untuk alamat IP agar terhubung ke server web.

> **Note**
>
> Halaman uji Apache hanya muncul jika tidak ada konten di direktori root dokumen, `/var/www/html`. Setelah konten ditambahkan ke direktori root dokumen, konten tersebut akan muncul di alamat DNS publik dari instans EC2 Anda. Sebelumnya, konten tersebut muncul di halaman uji Apache.

5. Konfigurasikan server web untuk memulai setiap boot sistem menggunakan perintah `systemctl`.

```
sudo systemctl enable httpd
```

Untuk mengizinkan `ec2-user` mengelola file di direktori root default untuk server web Apache Anda, ubah kepemilikan dan izin direktori `/var/www`. Ada banyak cara untuk menyelesaikan tugas ini. Dalam tutorial ini, Anda menambahkan `ec2-user` ke grup `apache`, untuk memberikan kepemilikan grup `apache` atas direktori `/var/www` dan menetapkan izin tulis ke grup.

Mengatur izin file untuk server web Apache

1. Tambahkan pengguna `ec2-user` ke grup `apache`.

   ```
   sudo usermod -a -G apache ec2-user
   ```

2. Keluar untuk menyegarkan izin Anda dan masukkan grup `apache` baru.

   ```
   exit
   ```

3. Masuk kembali dan pastikan grup `apache` ada dengan perintah `groups`.

   ```
   groups
   ```

   Output Anda akan terlihat seperti berikut:

   ```
   ec2-user adm wheel apache systemd-journal
   ```

4. Ubah kepemilikan grup atas direktori `/var/www` dan kontennya ke grup `apache`.

   ```
   sudo chown -R ec2-user:apache /var/www
   ```

5. Ubah izin direktori atas `/var/www` dan subdirektorinya untuk menambahkan izin tulis grup dan atur ID grup pada subdirektori yang dibuat di masa mendatang.

   ```
   sudo chmod 2775 /var/www
   find /var/www -type d -exec sudo chmod 2775 {} \;
   ```

6. Ubah izin file secara berulang di direktori `/var/www` dan subdirektorinya untuk menambahkan izin tulis grup.

   ```
   find /var/www -type f -exec sudo chmod 0664 {} \;
   ```

Sekarang, `ec2-user` (dan setiap anggota grup `apache` mendatang) dapat menambahkan, menghapus, dan mengedit file pada root dokumen Apache. Ini memungkinkan Anda untuk menambahkan konten, seperti situs web statis atau aplikasi PHP.

> **Note**
>
> Server web yang menjalankan protokol HTTP tidak memberikan keamanan transportasi untuk data yang dikirim atau diterimanya. Saat Anda menghubungkan ke server HTTP menggunakan browser web, banyak informasi yang terlihat oleh penyadap di mana saja di sepanjang jalur jaringan. Informasi ini mencakup URL yang Anda kunjungi, konten halaman web yang Anda terima, dan konten (termasuk kata sandi) dari setiap formulir HTML.
> Praktik terbaik untuk mengamankan server web Anda adalah dengan menginstal dukungan untuk HTTPS (HTTP Secure). Protokol ini melindungi data Anda dengan enkripsi SSL/TLS. Untuk informasi selengkapnya, lihat Tutorial: Mengonfigurasi SSL/TLS dengan Amazon Linux AMI di Panduan Pengguna Amazon EC2.

Menghubungkan server web Apache ke klaster DB

Selanjutnya, Anda akan menambahkan konten ke server web Apache yang terhubung ke klaster DB Amazon Aurora.

Menambahkan konten ke server web Apache yang terhubung ke klaster DB Anda

1. Saat Anda masih terhubung ke instans EC2, ubah direktori ke `/var/www` dan buat subdirektori baru yang diberi nama `inc`.

   ```
   cd /var/www
   mkdir inc
   cd inc
   ```

2. Buat file baru dalam direktori `inc` yang diberi nama `dbinfo.inc`, lalu edit file tersebut dengan menggunakan nano (atau editor pilihan Anda).

   ```
   >dbinfo.inc
   nano dbinfo.inc
   ```

3. Tambahkan konten berikut ini ke file `dbinfo.inc`. Di sini, *`db_instance_endpoint`* adalah titik akhir tulis klaster DB, tanpa port, untuk klaster DB Anda.

> **Note**
>
> Kami merekomendasikan agar Anda menempatkan nama pengguna dan informasi kata sandi dalam folder yang bukan bagian dari root dokumen untuk server web Anda. Hal ini mengurangi kemungkinan terungkapnya informasi keamanan Anda.
> Pastikan untuk mengubah `master password` ke kata sandi yang sesuai di aplikasi Anda.

```
<?php

define('DB_SERVER', 'db_cluster_writer_endpoint');
define('DB_USERNAME', 'tutorial_user');
define('DB_PASSWORD', 'master password');
define('DB_DATABASE', 'sample');
?>
```

4. Simpan dan tutup file `dbinfo.inc`. Jika Anda menggunakan nano, simpan dan tutup file dengan menggunakan Ctrl+S dan Ctrl+X.
5. Ubah direktori ke `/var/www/html`.

```
cd /var/www/html
```

6. Buat file baru dalam direktori `html` yang diberi nama `SamplePage.php`, lalu edit file tersebut dengan menggunakan nano (atau editor pilihan Anda).

```
>SamplePage.php
nano SamplePage.php
```

7. Tambahkan konten berikut ke file `SamplePage.php`:

   MariaDB & MySQL

```
<?php include "../inc/dbinfo.inc"; ?>
<html>
<body>
<h1>Sample page</h1>
<?php
```

```
  /* Connect to MySQL and select the database. */
  $connection = mysqli_connect(DB_SERVER, DB_USERNAME, DB_PASSWORD);

  if (mysqli_connect_errno()) echo "Failed to connect to MySQL: " .
 mysqli_connect_error();

  $database = mysqli_select_db($connection, DB_DATABASE);

  /* Ensure that the EMPLOYEES table exists. */
  VerifyEmployeesTable($connection, DB_DATABASE);

  /* If input fields are populated, add a row to the EMPLOYEES table. */
  $employee_name = htmlentities($_POST['NAME']);
  $employee_address = htmlentities($_POST['ADDRESS']);

  if (strlen($employee_name) || strlen($employee_address)) {
    AddEmployee($connection, $employee_name, $employee_address);
  }
?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
        <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
        <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
        <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>

<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
  <tr>
```

```
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = mysqli_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = mysqli_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>

</table>

<!-- Clean up. -->
<?php

  mysqli_free_result($result);
  mysqli_close($connection);

?>

</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = mysqli_real_escape_string($connection, $name);
   $a = mysqli_real_escape_string($connection, $address);

   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!mysqli_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}
```

```
/* Check whether the table exists and, if not, create it. */
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID int(11) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!mysqli_query($connection, $query)) echo("<p>Error creating table.</
p>");
  }
}

/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = mysqli_real_escape_string($connection, $tableName);
  $d = mysqli_real_escape_string($connection, $dbName);

  $checktable = mysqli_query($connection,
      "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME = '$t'
 AND TABLE_SCHEMA = '$d'");

  if(mysqli_num_rows($checktable) > 0) return true;

  return false;
}
?>
```

PostgreSQL

```
<?php include "../inc/dbinfo.inc"; ?>

<html>
<body>
<h1>Sample page</h1>
<?php

/* Connect to PostgreSQL and select the database. */
```

```
$constring = "host=" . DB_SERVER . " dbname=" . DB_DATABASE . " user=" .
 DB_USERNAME . " password=" . DB_PASSWORD ;
$connection = pg_connect($constring);

if (!$connection){
 echo "Failed to connect to PostgreSQL";
 exit;
}

/* Ensure that the EMPLOYEES table exists. */
VerifyEmployeesTable($connection, DB_DATABASE);

/* If input fields are populated, add a row to the EMPLOYEES table. */
$employee_name = htmlentities($_POST['NAME']);
$employee_address = htmlentities($_POST['ADDRESS']);

if (strlen($employee_name) || strlen($employee_address)) {
  AddEmployee($connection, $employee_name, $employee_address);
}

?>

<!-- Input form -->
<form action="<?PHP echo $_SERVER['SCRIPT_NAME'] ?>" method="POST">
  <table border="0">
    <tr>
      <td>NAME</td>
      <td>ADDRESS</td>
    </tr>
    <tr>
      <td>
    <input type="text" name="NAME" maxlength="45" size="30" />
      </td>
      <td>
    <input type="text" name="ADDRESS" maxlength="90" size="60" />
      </td>
      <td>
    <input type="submit" value="Add Data" />
      </td>
    </tr>
  </table>
</form>
<!-- Display table data. -->
<table border="1" cellpadding="2" cellspacing="2">
```

```
  <tr>
    <td>ID</td>
    <td>NAME</td>
    <td>ADDRESS</td>
  </tr>

<?php

$result = pg_query($connection, "SELECT * FROM EMPLOYEES");

while($query_data = pg_fetch_row($result)) {
  echo "<tr>";
  echo "<td>",$query_data[0], "</td>",
       "<td>",$query_data[1], "</td>",
       "<td>",$query_data[2], "</td>";
  echo "</tr>";
}
?>
</table>

<!-- Clean up. -->
<?php

  pg_free_result($result);
  pg_close($connection);
?>
</body>
</html>


<?php

/* Add an employee to the table. */
function AddEmployee($connection, $name, $address) {
   $n = pg_escape_string($name);
   $a = pg_escape_string($address);
   echo "Forming Query";
   $query = "INSERT INTO EMPLOYEES (NAME, ADDRESS) VALUES ('$n', '$a');";

   if(!pg_query($connection, $query)) echo("<p>Error adding employee data.</
p>");
}

/* Check whether the table exists and, if not, create it. */
```

```
function VerifyEmployeesTable($connection, $dbName) {
  if(!TableExists("EMPLOYEES", $connection, $dbName))
  {
     $query = "CREATE TABLE EMPLOYEES (
         ID serial PRIMARY KEY,
         NAME VARCHAR(45),
         ADDRESS VARCHAR(90)
       )";

     if(!pg_query($connection, $query)) echo("<p>Error creating table.</p>");
  }
}
/* Check for the existence of a table. */
function TableExists($tableName, $connection, $dbName) {
  $t = strtolower(pg_escape_string($tableName)); //table name is case sensitive
  $d = pg_escape_string($dbName); //schema is 'public' instead of 'sample' db
 name so not using that

  $query = "SELECT TABLE_NAME FROM information_schema.TABLES WHERE TABLE_NAME =
 '$t';";
  $checktable = pg_query($connection, $query);

  if (pg_num_rows($checktable) >0) return true;
  return false;

}
?>
```

8. Simpan dan tutup file `SamplePage.php`.
9. Pastikan server web Anda berhasil terhubung ke klaster DB Anda dengan membuka browser web dan menelusuri ke `http://`*`EC2 instance endpoint`*`/SamplePage.php`, misalnya: `http://ec2-12-345-67-890.us-west-2.compute.amazonaws.com/SamplePage.php`.

Anda dapat menggunakan `SamplePage.php` untuk menambahkan data ke klaster DB Anda. Data yang Anda tambahkan akan ditampilkan di halaman tersebut. Untuk memverifikasi apakah data dimasukkan ke dalam tabel, instal klien MySQL pada instans Amazon EC2. Kemudian, hubungkan ke klaster DB dan kueri tabelnya.

Untuk informasi tentang menghubungkan ke klaster DB, lihat Menghubungkan ke klaster DB Amazon Aurora.

Untuk memastikan klaster DB Anda seaman mungkin, pastikan sumber di luar VPC tidak dapat menghubungkan ke klaster DB Anda.

Setelah selesai menguji server web dan basis data, Anda harus menghapus klaster DB dan instans Amazon EC2 Anda.

- Untuk membuat klaster DB, ikuti petunjuk di Menghapus instans DB atau klaster DB Aurora. Anda tidak perlu membuat snapshot final.
- Untuk mengakhiri instans Amazon EC2, ikuti instruksi di Mengakhiri instans Anda dalam Panduan Pengguna Amazon EC2.

tutorial Amazon Aurora dan kode sampel

AWS Dokumentasi mencakup beberapa tutorial yang memandu Anda melalui kasus penggunaan Amazon Aurora yang umum. Banyak dari tutorial ini menunjukkan cara menggunakan Aurora dengan AWS layanan lain. Selain itu, Anda dapat mengakses kode sampel di GitHub.

> **Note**
>
> Tutorial lainnya dapat dilihat di Blog Basis Data AWS. Untuk informasi tentang pelatihan, lihat Pelatihan dan Sertifikasi AWS.

Topik

- Tutorial dalam panduan ini
- Tutorial dalam AWS panduan lain
- AWS lokakarya dan portal konten lab untuk Aurora PostgreSQL
- AWS lokakarya dan portal konten lab untuk
- Tutorial dan kode sampel di GitHub
- Menggunakan layanan ini dengan AWS SDK

Tutorial dalam panduan ini

Tutorial berikut dalam panduan ini menunjukkan cara melakukan tugas umum dengan Amazon Aurora:

- Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4)

  Pelajari cara menyertakan klaster DB dalam cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. Dalam hal ini, VPC membagikan data dengan server web yang dijalankan di instans Amazon EC2 dalam VPC yang sama.
- Tutorial: Membuat VPC untuk digunakan dengan klaster DB (mode tumpukan ganda)

  Pelajari cara menyertakan klaster DB dalam cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. Dalam hal ini, VPC membagikan data dengan instans Amazon EC2 dalam VPC

yang sama. Dalam tutorial ini, Anda akan membuat VPC untuk skenario ini yang berfungsi dengan basis data yang berjalan dalam mode tumpukan ganda.

- Tutorial: Membuat server web dan klaster DB Amazon Aurora

  Pelajari cara menginstal server web Apache dengan PHP dan membuat basis data MySQL. Server web yang berjalan di instans Amazon EC2 menggunakan Amazon Linux, dan basis data MySQL adalah klaster DB Aurora MySQL. Kedua instans Amazon EC2 dan klaster DB tersebut berjalan di Amazon VPC.

- Tutorial: Memulihkan klaster DB Amazon Aurora dari snapshot klaster DB

  Pelajari cara memulihkan klaster DB dari snapshot klaster DB.

- Tutorial: Menggunakan tag untuk menentukan klaster DB Aurora yang akan dihentikan.

  Pelajari cara menggunakan tag untuk menentukan klaster DB Aurora yang akan dihentikan.

- Tutorial: Mencatat perubahan status instans DB menggunakan Amazon EventBridge

  Pelajari cara mencatat perubahan status instans DB menggunakan Amazon EventBridge dan AWS Lambda.

Tutorial dalam AWS panduan lain

Tutorial berikut di AWS panduan lain menunjukkan kepada Anda cara melakukan tugas-tugas umum dengan Aurora:

> **Note**
>
> Beberapa tutorial menggunakan instans DB Amazon RDS, tetapi dapat disesuaikan untuk menggunakan klaster DB Aurora.

- Tutorial: Aurora Nirserver di Panduan Developer AWS AppSync

  Pelajari cara menggunakan sumber data AWS AppSync untuk menjalankan perintah SQL terhadap cluster Aurora Serverless DB dengan API Data diaktifkan. Anda dapat menggunakan resolver AWS AppSync untuk menjalankan pernyataan SQL terhadap Data API dengan kueri, mutasi, dan langganan GraphQL.

- Tutorial: Memutar Rahasia untuk AWS Database di AWS Secrets Manager Panduan Pengguna

Pelajari cara membuat rahasia untuk AWS database dan mengkonfigurasi rahasia untuk memutar pada jadwal. Anda memicu satu rotasi secara manual, kemudian mengonfirmasi bahwa versi baru rahasia terus memberikan akses.

- Tutorial dan sampel di Panduan Developer AWS Elastic Beanstalk

  Pelajari cara menerapkan aplikasi yang menggunakan database Amazon RDS. AWS Elastic Beanstalk

- Menggunakan Data dari Basis Data Amazon RDS untuk Membuat Sumber Data Amazon ML di Panduan Developer Amazon Machine Learning

  Pelajari cara membuat objek sumber data Amazon Machine Learning (Amazon ML) dari data yang disimpan dalam instans DB MySQL.

- Mengaktifkan Akses ke Instans Amazon RDS secara Manual di VPC di Panduan Pengguna Amazon QuickSight

  Pelajari cara mengaktifkan QuickSight akses Amazon ke instans Amazon RDS DB di VPC.

AWS lokakarya dan portal konten lab untuk Aurora PostgreSQL

Kumpulan lokakarya dan konten praktis berikut ini membantu Anda memperoleh pemahaman tentang fitur dan kapabilitas Amazon Aurora PostgreSQL:

- Membuat Klaster Aurora

  Pelajari cara membuat klaster Amazon Aurora PostgreSQL secara manual.

- Membuat lingkungan IDE berbasis Cloud Cloud9 untuk menghubungkan ke basis data

  Pelajari cara mengonfigurasi Cloud9 dan menginisialisasi basis data PostgreSQL.

- Kloning Cepat

  Pelajari cara membuat kloning cepat Aurora.

- Manajemen Rencana Kueri

  Pelajari cara mengontrol rencana eksekusi untuk serangkaian pernyataan menggunakan manajemen rencana kueri.

- Manajemen Cache Klaster

Pelajari fitur Manajemen Cache Klaster di Aurora PostgreSQL.

- Streaming Aktivitas Basis Data

  Pelajari cara memantau dan mengaudit aktivitas basis data Anda dengan fitur ini.

- Menggunakan Wawasan Performa

  Pelajari cara memantau dan menyetel instans DB Anda menggunakan Wawasan performa.

- Pemantauan Performa dengan Alat RDS

  Pelajari cara menggunakan AWS dan alat Postgres (Cloudwatch, Enhanced Monitoring, Slow Query Logs, Performance Insights, PostgreSQL Catalog Views) untuk memahami masalah kinerja dan mengidentifikasi cara untuk meningkatkan kinerja database Anda.

- Penskalaan Otomatis Replika Baca

  Pelajari cara kerja penskalaan otomatis replika baca Aurora dalam praktiknya menggunakan skrip pembuat beban.

- Menguji Toleransi Kesalahan

  Pelajari bagaimana klaster DB dapat menoleransi kegagalan.

- Basis data global Aurora

  Pelajari tentang Basis Data Global Aurora.

- Menggunakan Machine Learning

  Pelajari Aurora Machine Learning.

- Aurora Nirserver v2

  Pelajari tentang Aurora Nirserver v2.

- Ekstensi Bahasa Tepercaya untuk Aurora PostgreSQL

  Pelajari cara membuat ekstensi berperforma tinggi yang berjalan dengan aman di Aurora PostgreSQL.

AWS lokakarya dan portal konten lab untuk

Kumpulan lokakarya dan konten praktis berikut ini membantu Anda memperoleh pemahaman tentang fitur dan kapabilitas Amazon Aurora MySQL:

- Membuat Klaster Aurora

  Pelajari cara membuat klaster Amazon Aurora MySQL secara manual.

- Membuat lingkungan IDE berbasis Cloud Cloud9 untuk menghubungkan ke basis data

  Pelajari cara mengonfigurasi Cloud9 dan menginisialisasi basis data MySQL.

- Kloning Cepat

  Pelajari cara membuat kloning cepat Aurora.

- Melacak Mundur sebuah Klaster

  Pelajari cara melacak mundur klaster DB.

- Menggunakan Wawasan Performa

  Pelajari cara memantau dan menyetel instans DB Anda menggunakan Wawasan performa.

- Pemantauan Performa dengan Alat RDS

  Pelajari cara menggunakan AWS dan alat SQL untuk memahami masalah kinerja dan mengidentifikasi cara untuk meningkatkan kinerja database Anda.

- Menganalisis Performa Kueri

  Pelajari cara memecahkan masalah terkait performa SQL menggunakan alat lain.

- Penskalaan Otomatis Replika Baca

  Pelajari cara kerja penskalaan otomatis replika baca.

- Menguji Toleransi Kesalahan

  Pelajari tentang fitur toleransi kesalahan dan ketersediaan tinggi di Aurora MySQL.

- Basis data global Aurora

  Pelajari tentang Basis Data Global Aurora.

- Aurora Nirserver v2

  Pelajari tentang Aurora Nirserver v2.

- Menggunakan Machine Learning

  Pelajari Aurora Machine Learning.

Tutorial dan kode sampel di GitHub

Tutorial dan kode contoh berikut GitHub menunjukkan kepada Anda cara melakukan tugas-tugas umum dengan Aurora:

- Membuat perpustakaan peminjaman Aurora Serverless v2

  Pelajari cara membuat aplikasi perpustakaan peminjaman tempat pengunjung dapat meminjam dan mengembalikan buku. Contohnya menggunakan Aurora Serverless v2 dan AWS SDK for Python (Boto3).
- Membuat aplikasi pelacak item Amazon Aurora dengan API REST Spring yang mengueri data Aurora Serverless v2 menggunakan SDK for Java 2.x

  Pelajari cara membuat API REST Spring yang mengueri data Aurora Serverless v2. Ini untuk penggunaan oleh aplikasi React menggunakan SDK for Java 2.x.
- Membuat aplikasi pelacak item Amazon Aurora yang menanyakan data menggunakan Aurora Serverless v2AWS SDK for PHP

  Pelajari cara membuat aplikasi yang menggunakan `RdsDataClient` API Data dan Aurora Serverless v2 untuk melacak serta melaporkan item pekerjaan. Contoh menggunakan AWS SDK for PHP.
- Membuat aplikasi pelacak item Amazon Aurora yang mengueri data Aurora Serverless v2 menggunakan AWS SDK for Python (Boto3)

  Pelajari cara membuat aplikasi yang menggunakan `RdsDataClient` API Data dan Aurora Serverless v2 untuk melacak serta melaporkan item pekerjaan. Contoh menggunakan AWS SDK for Python (Boto3).

Menggunakan layanan ini dengan AWS SDK

AWS kit pengembangan perangkat lunak (SDK) tersedia untuk banyak bahasa pemrograman populer. Setiap SDK menyediakan API, contoh kode, dan dokumentasi yang memudahkan developer untuk membangun aplikasi dalam bahasa pilihan mereka.

Dokumentasi SDK	Contoh kode
AWS SDK for C++	AWS SDK for C++ contoh kode

Dokumentasi SDK	Contoh kode
AWS SDK for Go	AWS SDK for Go contoh kode
AWS SDK for Java	AWS SDK for Java contoh kode
AWS SDK for JavaScript	AWS SDK for JavaScript contoh kode
AWS SDK for Kotlin	AWS SDK for Kotlin contoh kode
AWS SDK for .NET	AWS SDK for .NET contoh kode
AWS SDK for PHP	AWS SDK for PHP contoh kode
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) contoh kode
AWS SDK for Ruby	AWS SDK for Ruby contoh kode
AWS SDK for Rust	AWS SDK for Rust contoh kode
AWS SDK untuk SAP ABAP	AWS SDK untuk SAP ABAP contoh kode
AWS SDK for Swift	AWS SDK for Swift contoh kode

Untuk contoh yang spesifik untuk layanan ini, lihat Contoh kode untuk Aurora menggunakan SDK AWS.

Ketersediaan contoh

Tidak menemukan yang Anda cari? Minta contoh kode menggunakan tautan Berikan umpan balik di bagian bawah halaman ini.

Mengonfigurasi klaster DB Amazon Aurora Anda

Bagian ini menunjukkan cara mengatur klaster DB Aurora Anda. Sebelum membuat klaster DB Aurora, tentukan kelas instans DB yang akan menjalankan klaster DB. Juga, tentukan di mana cluster DB akan berjalan dengan memilih AWS Region. Selanjutnya, buat klaster DB. Jika Anda memiliki data di luar Aurora, Anda dapat memigrasikan data ke dalam klaster DB Aurora.

Topik

- Membuat klaster DB Amazon Aurora
- Membuat sumber daya Amazon Aurora dengan AWS CloudFormation
- Menghubungkan ke klaster DB Amazon Aurora
- Bekerja dengan grup parameter
- Memigrasikan data ke klaster DB Amazon Aurora
- Membuat ElastiCache cache Amazon menggunakan pengaturan instans DB cluster Aurora DB

Membuat klaster DB Amazon Aurora

Klaster DB Amazon Aurora terdiri dari instans DB, yang kompatibel dengan MySQL atau PostgreSQL, dan volume klaster yang menyimpan data untuk klaster DB, yang disalin ke tiga Zona Ketersediaan sebagai satu volume virtual. Secara default, klaster Aurora DB berisi instans DB primer yang melakukan pembacaan dan penulisan, dan, secara opsional, hingga 15 Replika Aurora (instans DB pembaca). Untuk informasi selengkapnya tentang klaster Aurora DB, lihat Klaster DB Amazon Aurora.

Aurora memiliki dua jenis utama klaster DB:

- Terprovisi Aurora – Anda memilih kelas instans DB untuk instans penulis dan pembaca berdasarkan beban kerja yang Anda harapkan. Untuk informasi selengkapnya, lihat Kelas instans DB Aurora. Terprovisi Aurora memiliki beberapa opsi, termasuk basis data global Aurora. Untuk informasi selengkapnya, lihat Menggunakan basis data global Amazon Aurora.
- Aurora Serverless – Aurora Serverless v1 dan Aurora Serverless v2 merupakan konfigurasi penskalaan otomatis sesuai permintaan untuk Aurora. Kapasitas disesuaikan secara otomatis berdasarkan permintaan aplikasi. Anda hanya dikenai biaya untuk sumber daya yang menggunakan klaster DB Anda. Otomatisasi ini sangat berguna untuk lingkungan dengan beban kerja yang sangat bervariasi dan tidak dapat diprediksi. Lihat informasi yang lebih lengkap di Menggunakan Amazon Aurora Serverless v1 dan Menggunakan Aurora Serverless v2.

Di bagian berikut ini, Anda dapat mengetahui cara membuat klaster Aurora DB. Untuk memulai, pertama-tama lihat Prasyarat klaster DB.

Untuk petunjuk tentang menghubungkan ke klaster Aurora DB Anda, lihat Menghubungkan ke klaster DB Amazon Aurora.

Daftar Isi

- Prasyarat klaster DB
  - Konfigurasikan jaringan untuk klaster DB
    - Konfigurasikan konektivitas jaringan otomatis dengan instans EC2
    - Konfigurasikan jaringan secara manual
  - Prasyarat tambahan
- Membuat klaster DB
  - Membuat instance DB primer (penulis)

- Pengaturan untuk klaster Aurora DB
- Pengaturan yang tidak berlaku untuk klaster DB Amazon Aurora
- Pengaturan yang tidak berlaku untuk instans DB Amazon Aurora

Prasyarat klaster DB

> **Important**
>
> Sebelum dapat membuat klaster Aurora DB, Anda harus menyelesaikan tugas-tugas dalam Menyiapkan lingkungan Anda untuk Amazon Aurora.

Berikut ini adalah prasyarat yang harus diselesaikan sebelum membuat klaster DB.

Topik

- Konfigurasikan jaringan untuk klaster DB
- Prasyarat tambahan

Konfigurasikan jaringan untuk klaster DB

Anda dapat membuat cluster Amazon Aurora DB hanya di cloud pribadi virtual (VPC) berdasarkan layanan VPC Amazon, di Wilayah AWS yang memiliki setidaknya dua Availability Zone. Grup subnet DB yang Anda pilih untuk klaster DB harus mencakup setidaknya dua Zona Ketersediaan. Konfigurasi ini memastikan bahwa klaster DB Anda selalu memiliki setidaknya satu instans DB tersedia untuk failover jika terjadi kegagalan Zona Ketersediaan.

Jika Anda berencana untuk mengatur konektivitas antara klaster DB baru Anda dan instans EC2 di VPC yang sama, Anda dapat melakukannya selama pembuatan klaster DB. Jika Anda berencana untuk terhubung ke klaster DB Anda dari sumber daya selain instans EC2 di VPC yang sama, Anda dapat mengonfigurasi koneksi jaringan secara manual.

Topik

- Konfigurasikan konektivitas jaringan otomatis dengan instans EC2
- Konfigurasikan jaringan secara manual

Konfigurasikan konektivitas jaringan otomatis dengan instans EC2

Saat membuat klaster Aurora DB, Anda dapat menggunakannya AWS Management Console untuk mengatur konektivitas antara instans Amazon EC2 dan cluster DB baru. Ketika Anda melakukannya, RDS mengonfigurasi VPC dan pengaturan jaringan Anda secara otomatis. Klaster DB tersebut dibuat dalam VPC yang sama dengan instans EC2, sehingga instans EC2 dapat mengakses klaster DB.

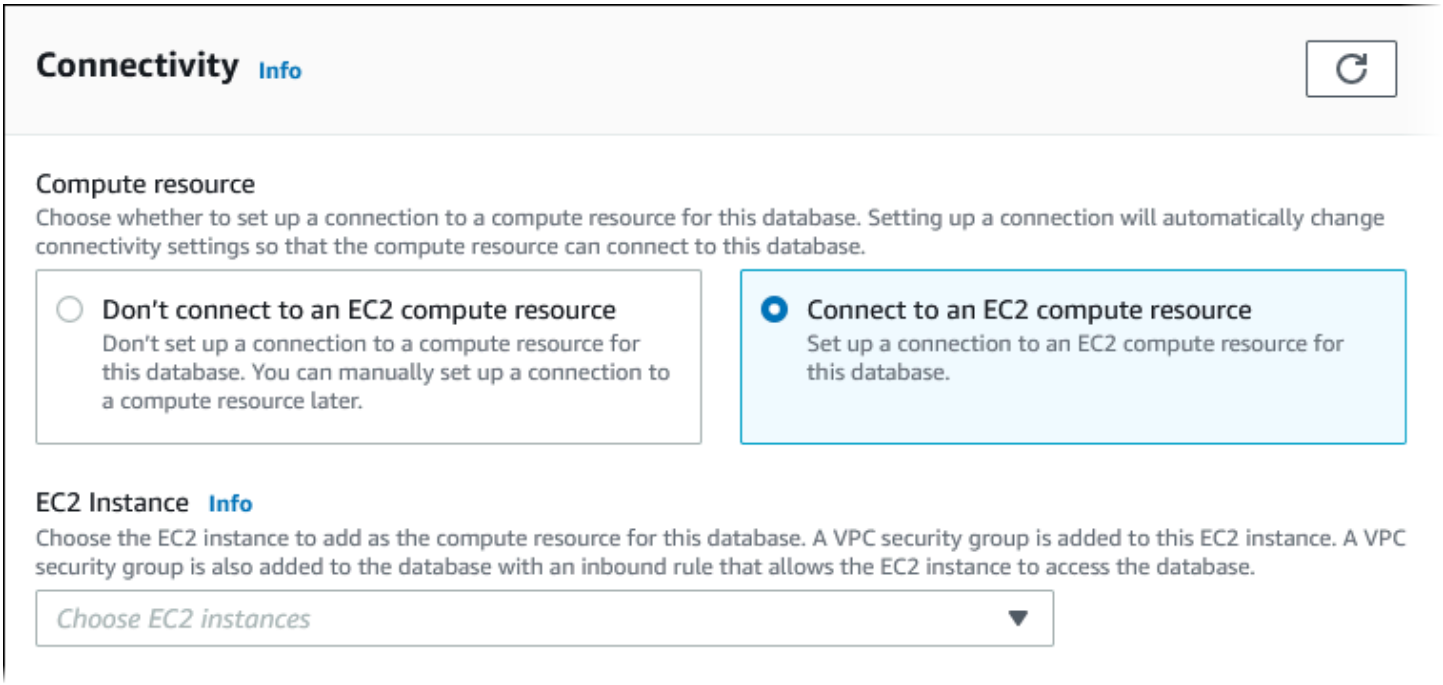
Berikut ini adalah persyaratan untuk menghubungkan instans EC2 dengan klaster DB:

- Instans EC2 harus ada Wilayah AWS sebelum Anda membuat cluster DB.

  Jika tidak ada instans EC2 di Wilayah AWS, konsol menyediakan tautan untuk membuatnya.
- Saat ini, klaster DB tidak dapat berupa klaster DB Aurora Serverless atau bagian dari basis data global Aurora.
- Pengguna yang membuat instans DB harus memiliki izin untuk melakukan operasi berikut:
  - `ec2:AssociateRouteTable`
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateRouteTable`
  - `ec2:CreateSubnet`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeRouteTables`
  - `ec2:DescribeSecurityGroups`
  - `ec2:DescribeSubnets`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

Menggunakan opsi ini membuat klaster DB privat. Klaster DB menggunakan grup subnet DB hanya dengan subnet privat untuk membatasi akses ke sumber daya dalam VPC.

Untuk menghubungkan instans EC2 ke klaster DB, pilih Hubungkan ke sumber daya komputasi EC2 di bagian Konektivitas pada halaman Buat basis data.



Saat Anda memilih Hubungkan ke sumber daya komputasi EC2, RDS menetapkan opsi berikut secara otomatis. Anda tidak dapat mengubah pengaturan ini kecuali Anda memilih untuk tidak mengatur konektivitas dengan instans EC2 dengan memilih Jangan hubungkan ke sumber daya komputasi EC2.

Opsi konsol	Pengaturan otomatis
Jenis jaringan	RDS menetapkan jenis jaringan ke IPv4. Saat ini, mode tumpukan ganda tidak didukung saat Anda mengatur koneksi antara instans EC2 dan klaster DB.
Cloud Privat Virtual (VPC)	RDS mengatur VPC ke yang terkait dengan instans EC2.
Grup subnet DB	RDS membutuhkan grup subnet DB dengan subnet privat di Zona Ketersediaan yang sama dengan instans EC2. Jika grup subnet DB yang memenuhi persyaratan ini ada, RDS menggunakan grup subnet DB yang ada. Secara default, opsi ini diatur ke Pengaturan otomatis. Ketika Anda memilih Pengaturan otomatis dan tidak ada grup subnet DB yang memenuhi persyaratan ini, tindakan berikut terjadi. RDS menggunakan tiga subnet privat yang tersedia di

Opsi konsol	Pengaturan otomatis
	tiga Zona Ketersediaan dengan salah satu Zona Ketersediaan sama dengan instans EC2. Jika subnet privat tidak tersedia di Zona Ketersediaan, RDS membuat subnet privat di Zona Ketersediaan. Kemudian RDS membuat grup subnet DB. Ketika subnet privat tersedia, RDS menggunakan tabel rute yang terkait dengan subnet dan menambahkan subnet apa pun yang dibuatnya ke tabel rute ini. Ketika tidak ada subnet privat yang tersedia, RDS membuat tabel rute tanpa akses gateway internet dan menambahkan subnet yang dibuatnya ke tabel rute. RDS juga memungkinkan Anda untuk menggunakan grup subnet DB yang ada. Pilih Pilih yang ada jika Anda ingin menggunakan grup subnet DB pilihan Anda yang sudah ada.
Akses publik	RDS memilih Tidak sehingga klaster DB tidak dapat diakses publik. Untuk keamanan, ini adalah praktik terbaik untuk menjaga basis data tetap privat dan memastikannya tidak dapat diakses dari internet.

Opsi konsol	Pengaturan otomatis
Grup keamanan VPC (firewall)	RDS membuat grup keamanan baru yang terkait dengan klaster DB. Grup keamanan diberi nama `rds-ec2-`*n*, dengan *n* berupa angka. Grup keamanan ini mencakup aturan masuk dengan grup keamanan VPC EC2 (firewall) sebagai sumbernya. Grup keamanan yang terkait dengan klaster DB ini memungkinkan instans EC2 untuk mengakses klaster DB. RDS juga membuat grup keamanan baru yang terkait dengan instans EC2. Grup keamanan diberi nama `ec2-rds-`*n*, dengan *n* berupa angka. Grup keamanan ini mencakup aturan keluar dengan grup keamanan VPC dari klaster DB sebagai sumbernya. Grup keamanan ini memungkinkan instans EC2 untuk mengirim lalu lintas ke klaster DB. Anda dapat menambahkan grup keamanan baru lainnya dengan memilih Buat baru dan mengetik nama grup keamanan baru. Anda dapat menambahkan grup keamanan yang ada dengan memilih Pilih yang ada dan memilih grup keamanan untuk ditambahkan.
Zona Ketersediaan	Jika Anda tidak membuat Replika Aurora dalam Ketersediaan & durabilitas selama pembuatan klaster DB (deployment AZ Tunggal), RDS memilih Zona Ketersediaan dari instans EC2. Saat Anda membuat Replika Aurora selama pembuatan klaster DB (deployment Multi-AZ), RDS memilih Zona Ketersediaan dari instans EC2 untuk satu instans DB di klaster DB. RDS secara acak memilih Zona Ketersediaan yang berbeda untuk instans DB lainnya di klaster DB. Instans DB primer atau Replika Aurora dibuat di Zona Ketersediaan yang sama dengan instans EC2. Ada kemungkinan biaya lintas Zona Ketersediaan jika instans DB primer dan instans EC2 berada di Zona Ketersediaan yang berbeda.

Untuk informasi selengkapnya tentang pengaturan ini, lihat Pengaturan untuk klaster Aurora DB.

Jika Anda membuat perubahan apa pun pada pengaturan ini setelah klaster DB dibuat, perubahan tersebut dapat memengaruhi koneksi antara instans EC2 dan klaster DB.

Konfigurasikan jaringan secara manual

Jika Anda berencana untuk terhubung ke klaster DB Anda dari sumber daya selain instans EC2 di VPC yang sama, Anda dapat mengonfigurasi koneksi jaringan secara manual. Jika Anda menggunakan AWS Management Console untuk membuat klaster DB, Anda dapat meminta Amazon RDS untuk membuat VPC secara otomatis. Atau Anda dapat menggunakan VPC yang ada atau membuat VPC baru untuk klaster Aurora DB Anda. Pendekatan apa pun yang Anda ambil, VPC Anda harus memiliki setidaknya satu subnet di masing-masing dari setidaknya dua Zona Ketersediaan untuk digunakan dengan klaster DB Amazon Aurora.

Secara default, Amazon RDS membuat instans DB primer dan Replika Aurora di Zona Ketersediaan secara otomatis untuk Anda. Untuk memilih Zona Ketersediaan tertentu, Anda perlu mengubah pengaturan deployment Multi-AZ Ketersediaan & durabilitas menjadi Jangan buat Replika Aurora. Tindakan ini akan membuka pengaturan Zona Ketersediaan yang memungkinkan Anda memilih dari beberapa Zona Ketersediaan di VPC Anda. Namun, kami sangat menyarankan agar Anda mempertahankan pengaturan default serta membiarkan Amazon RDS membuat deployment Multi-AZ dan memilih Zona Ketersediaan untuk Anda. Dengan demikian, klaster Aurora DB Anda akan dibuat dengan failover cepat dan fitur ketersediaan tinggi yang merupakan dua manfaat utama Aurora.

Jika Anda tidak memiliki VPC default atau Anda belum membuat VPC, Anda dapat meminta Amazon RDS untuk membuat VPC secara otomatis ketika Anda membuat klaster DB menggunakan konsol. Jika tidak, Anda harus melakukan hal berikut:

- Buat VPC dengan setidaknya satu subnet di masing-masing setidaknya dua Availability Zones di Wilayah AWS mana Anda ingin menyebarkan cluster DB Anda. Lihat informasi yang lebih lengkap di Bekerja dengan instans DB dalam VPC dan Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4).
- Tentukan grup keamanan VPC yang mengizinkan koneksi ke klaster DB Anda. Lihat informasi yang lebih lengkap di Berikan akses ke klaster DB dalam VPC dengan membuat grup keamanan dan Mengontrol akses dengan grup keamanan.
- Tentukan grup subnet RDS yang mendefinisikan setidaknya dua subnet di VPC yang dapat digunakan oleh klaster DB. Untuk informasi selengkapnya, lihat Bekerja dengan grup subnet DB.

Untuk informasi selengkapnya tentang VPC, lihat Amazon VPC dan Amazon Aurora. Untuk tutorial yang mengonfigurasi jaringan untuk klaster DB privat, lihat Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4).

Jika Anda ingin terhubung ke sumber daya yang tidak berada di VPC yang sama dengan klaster Aurora DB, lihat skenario yang sesuai dalam Skenario untuk mengakses klaster DB di VPC.

Prasyarat tambahan

Sebelum Anda membuat klaster DB Anda, pertimbangkan prasyarat tambahan berikut:

- Jika Anda terhubung ke kredensyal AWS menggunakan AWS Identity and Access Management (IAM), AWS akun Anda harus memiliki kebijakan IAM yang memberikan izin yang diperlukan untuk melakukan operasi Amazon RDS. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Amazon Aurora.

  Jika Anda menggunakan IAM untuk mengakses konsol Amazon RDS, Anda harus terlebih dahulu masuk ke AWS Management Console dengan kredensyal pengguna Anda. Kemudian, buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
- Jika Anda ingin menyesuaikan parameter konfigurasi untuk klaster DB, Anda harus menentukan grup parameter klaster DB dan grup parameter DB dengan pengaturan parameter yang diperlukan. Untuk informasi tentang membuat atau memodifikasi grup parameter klaster DB atau grup parameter DB, lihat Bekerja dengan grup parameter.
- Tentukan nomor port TCP/IP yang akan ditentukan untuk klaster DB Anda. Firewall di beberapa perusahaan memblokir koneksi ke port default (3306 untuk MySQL, 5432 untuk PostgreSQL) untuk Aurora. Jika firewall perusahaan Anda memblokir port default, pilih port lain untuk klaster DB Anda. Semua instans dalam klaster DB menggunakan port yang sama.
- Jika versi mesin utama untuk database Anda telah mencapai akhir RDS dari tanggal dukungan standar, Anda harus menggunakan opsi Extended Support CLI atau parameter RDS API. Untuk informasi selengkapnya, lihat RDS Extended Support diPengaturan untuk klaster Aurora DB.

Membuat klaster DB

Anda dapat membuat cluster Aurora DB menggunakan AWS Management Console, AWS CLI, atau RDS API.

Konsol

Anda dapat membuat cluster DB menggunakan AWS Management Console dengan Easy create diaktifkan atau tidak diaktifkan. Dengan Pembuatan mudah aktif, Anda hanya menentukan jenis mesin DB, ukuran instans DB, dan pengidentifikasi instans DB. Pembuatan mudah menggunakan pengaturan default untuk opsi konfigurasi lainnya. Dengan Pembuatan mudah tidak aktif, Anda menentukan lebih banyak opsi konfigurasi saat Anda membuat basis data, termasuk opsi untuk ketersediaan, keamanan, pencadangan, dan pemeliharaan.

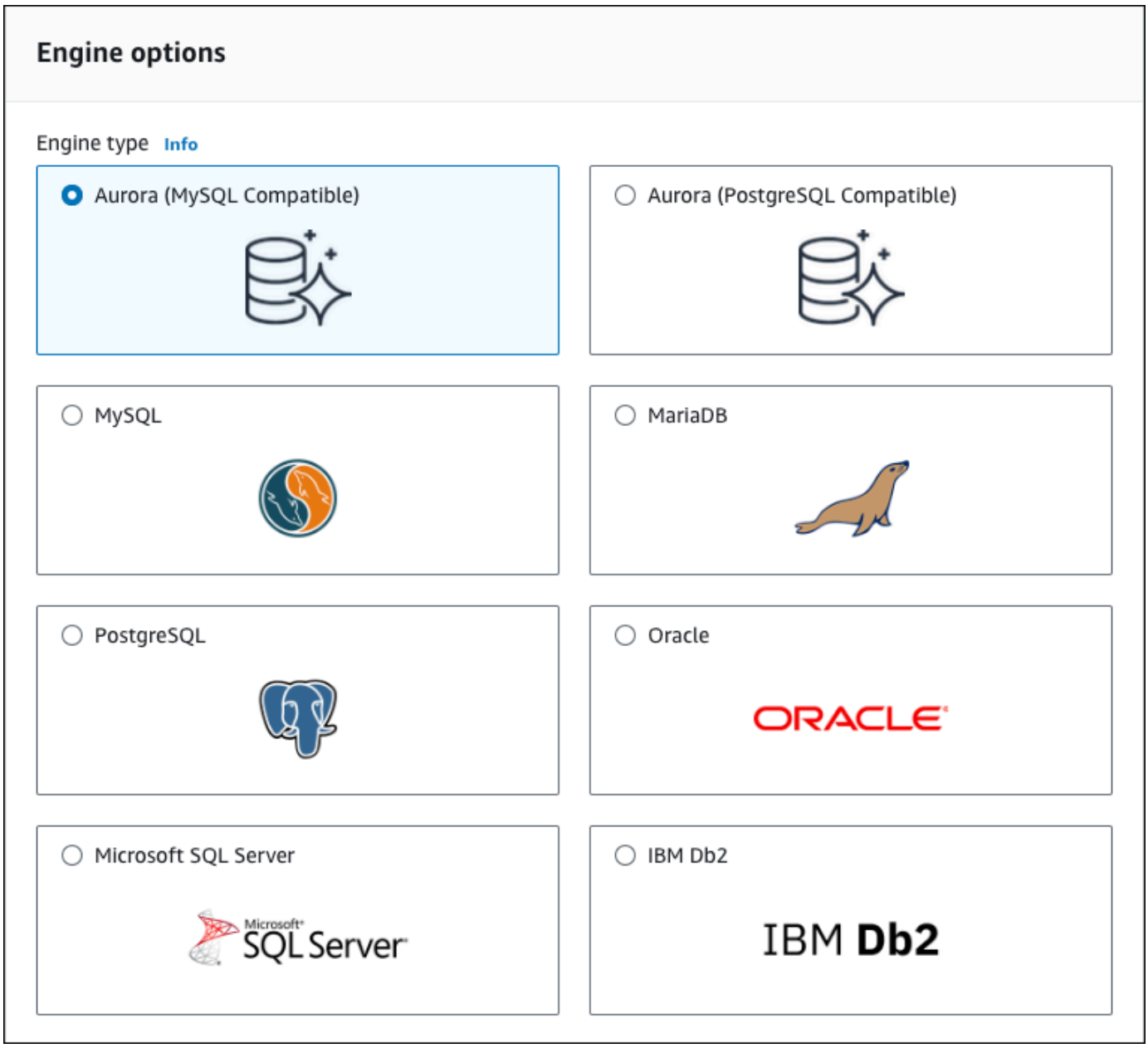
> **Note**
>
> Untuk contoh ini, Pembuatan standar aktif, dan Pembuatan mudah tidak diaktifkan. Untuk informasi tentang membuat cluster DB dengan Easy create diaktifkan, lihatMemulai dengan Amazon Aurora.

Untuk membuat klaster Aurora DB menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas AWS Management Console, pilih AWS Wilayah di mana Anda ingin membuat cluster DB.

   Aurora tidak tersedia di semua AWS Wilayah. Untuk daftar AWS Wilayah tempat Aurora tersedia, lihat. Ketersediaan wilayah
3. Di panel navigasi, pilih Basis Data.
4. Pilih Buat basis data.
5. Untuk Pilih metode pembuatan basis data, pilih Pembuatan standar.
6. Untuk Jenis engine, pilih salah satu dari berikut ini:

   - Aurora (Kompatibel dengan MySQL)
   - Aurora (Kompatibel dengan PostgreSQL)



7. Pilih Versi mesin.

   Untuk informasi selengkapnya, lihat Versi Amazon Aurora. Anda dapat menggunakan filter untuk memilih versi yang kompatibel dengan fitur yang Anda inginkan, seperti Aurora Serverless v2. Untuk informasi selengkapnya, lihat Menggunakan Aurora Serverless v2.

8. Di Templat, pilih templat yang sesuai dengan kasus penggunaan Anda.

9. Untuk memasukkan kata sandi master, lakukan hal berikut:

   a. Di bagian Pengaturan, perluas Pengaturan Kredensial.

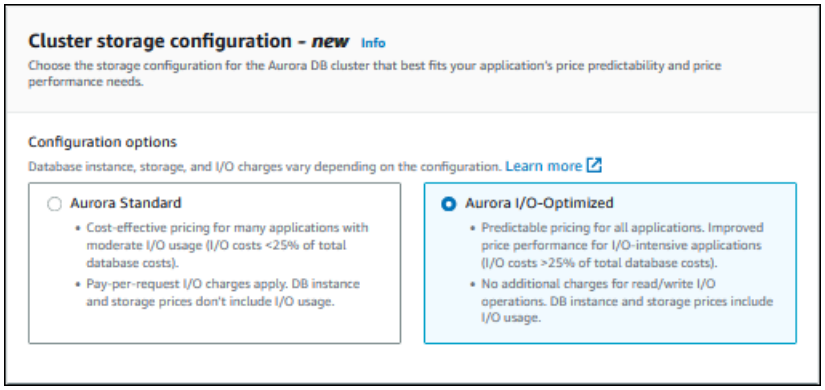
b. Hapus kotak centang Buat otomatis kata sandi.

c. (Opsional) Ubah nilai Nama pengguna master dan masukkan kata sandi yang sama di Kata sandi master dan Konfirmasikan kata sandi.

Secara default, instans DB baru menggunakan kata sandi yang dibuat secara otomatis untuk pengguna utama.

10. Di bagian Konektivitas di bagian Grup keamanan VPC (firewall), jika Anda memilih Buat baru, grup keamanan VPC dibuat dengan aturan masuk yang memungkinkan alamat IP komputer lokal Anda mengakses basis data.

11. Untuk Konfigurasi penyimpanan klaster, pilih Aurora I/O-Optimized atau Aurora Standard. Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.



12. (Opsional) Atur koneksi ke sumber daya komputasi untuk klaster DB ini.

Anda dapat mengonfigurasi konektivitas antara instans Amazon EC2 dan klaster DB baru selama pembuatan klaster DB. Untuk informasi selengkapnya, lihat Konfigurasikan konektivitas jaringan otomatis dengan instans EC2.

13. Untuk bagian yang lainnya, tentukan pengaturan klaster DB Anda. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

14. Pilih Buat basis data.

Jika Anda memilih untuk menggunakan kata sandi yang dibuat otomatis, tombol Lihat detail kredensial muncul pada halaman Basis data.

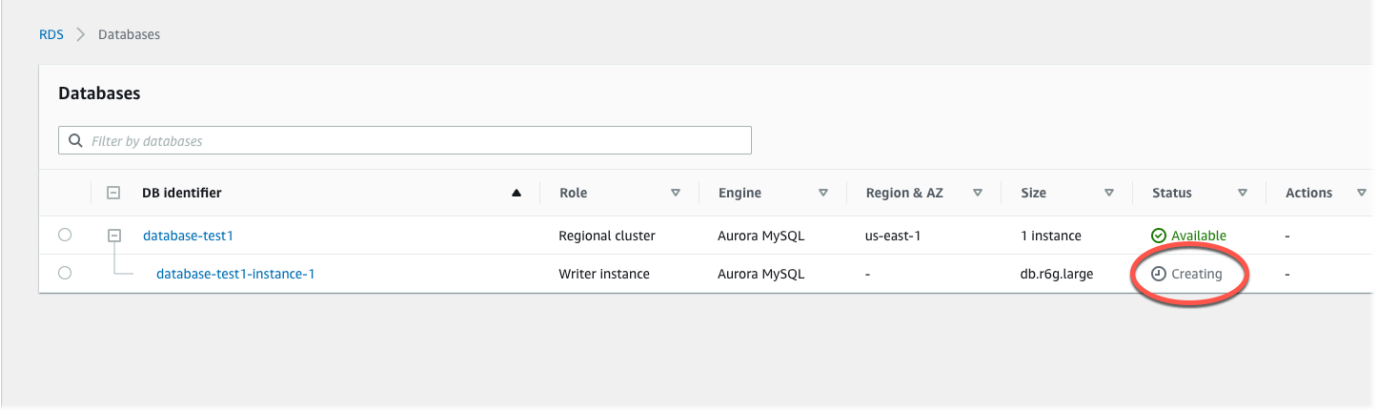
Untuk melihat nama pengguna dan kata sandi master klaster DB, pilih Lihat detail kredensial.

Untuk terhubung ke instans DB sebagai pengguna master, gunakan nama pengguna dan kata sandi yang muncul.

> **Important**
>
> Anda tidak dapat melihat kata sandi pengguna master lagi. Jika Anda tidak mencatatnya, Anda mungkin harus mengubahnya. Jika perlu mengubah kata sandi pengguna utama setelah instans DB tersedia, Anda dapat mengubah instans DB untuk melakukannya. Untuk informasi selengkapnya tentang cara memodifikasi instans DB, lihat Memodifikasi klaster DB Amazon Aurora.

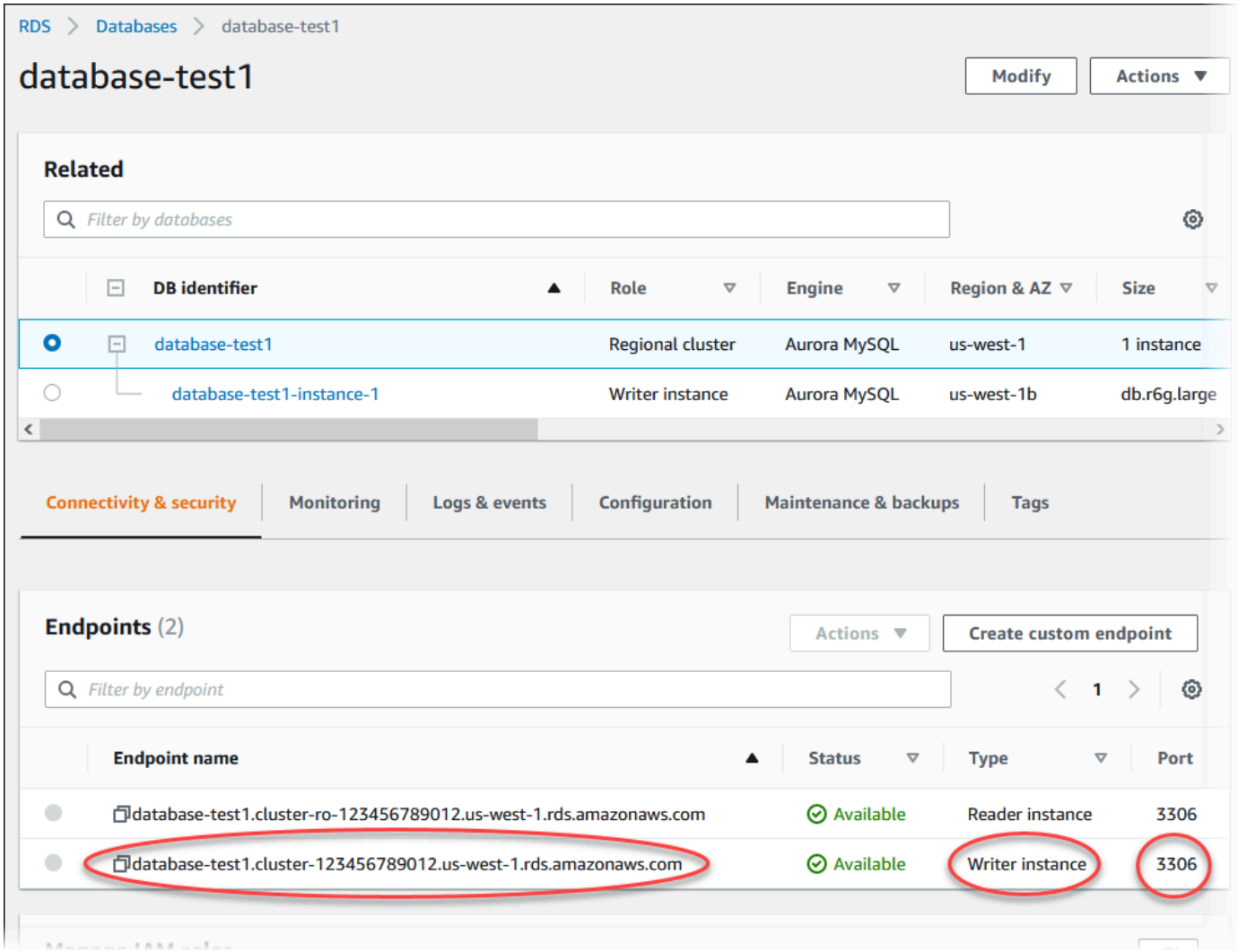
15. Untuk Basis data, pilih nama klaster Aurora DB baru.

    Pada konsol RDS, detail untuk klaster DB baru akan muncul. Klaster DB dan instans DB-nya memiliki status membuat hingga klaster DB siap digunakan.



    Saat statusnya berubah menjadi tersedia untuk keduanya, Anda dapat terhubung ke klaster DB. Tergantung pada kelas instans DB dan jumlah penyimpanan, diperlukan waktu hingga 20 menit sebelum klaster DB baru tersedia.

    Untuk melihat klaster yang baru dibuat, pilih Basis data dari panel navigasi di konsol Amazon RDS. Lalu pilih klaster DB untuk menampilkan detail klaster DB. Untuk informasi selengkapnya, lihat Melihat klaster DB Amazon Aurora.



Pada tab Koneksi & keamanan, catat port dan titik akhir instans DB penulis. Gunakan titik akhir dan port klaster dalam string koneksi JDBC dan ODBC Anda untuk aplikasi apa pun yang melakukan operasi tulis atau baca.

AWS CLI

> **Note**
>
> Sebelum Anda dapat membuat cluster Aurora DB menggunakan AWS CLI, Anda harus memenuhi prasyarat yang diperlukan, seperti membuat VPC dan grup subnet RDS DB. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.

Anda dapat menggunakan AWS CLI untuk membuat cluster Aurora MySQL DB atau cluster Aurora PostgreSQL DB.

Untuk membuat cluster DB MySQL Aurora menggunakan AWS CLI

Saat Anda membuat klaster DB atau instans DB yang kompatibel dengan Aurora MySQL 8.0 atau 5.7, Anda menentukan `aurora-mysql` untuk opsi `--engine`.

Selesaikan langkah-langkah berikut:

1. Identifikasi grup subnet DB dan ID grup keamanan VPC untuk cluster DB baru Anda, lalu panggil create-db-cluster AWS CLI perintah untuk membuat cluster DB MySQL Aurora.

   Misalnya, perintah berikut membuat klaster DB yang kompatibel dengan MySQL 8.0 bernama `sample-cluster`. Klaster ini menggunakan versi mesin default dan jenis penyimpanan Aurora I/O-Optimized.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster \
       --engine aurora-mysql --engine-version 8.0 \
       --storage-type aurora-iopt1 \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   Untuk Windows:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
       --engine aurora-mysql --engine-version 8.0 ^
       --storage-type aurora-iopt1 ^
       --master-username user-name --manage-master-user-password ^
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   Perintah berikut membuat klaster DB yang kompatibel dengan MySQL 5.7 bernama `sample-cluster`. Klaster ini menggunakan versi mesin default dan jenis penyimpanan Aurora Standard.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster  \
       --engine aurora-mysql --engine-version 5.7 \
       --storage-type aurora \
   ```

```
    --master-username user-name --manage-master-user-password \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

Untuk Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster sample-cluster  ^
    --engine aurora-mysql --engine-version 5.7 ^
    --storage-type aurora ^
    --master-username user-name --manage-master-user-password ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

2. Jika Anda menggunakan konsol untuk membuat klaster DB, Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan AWS CLI untuk membuat cluster DB, Anda harus secara eksplisit membuat instance utama untuk cluster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB. Sampai Anda membuat instans DB primer, titik akhir klaster DB tetap dalam status `Creating`.

   Panggil create-db-instance AWS CLI perintah untuk membuat instance utama untuk cluster DB Anda. Sertakan nama klaster DB sebagai nilai opsi `--db-cluster-identifier`.

   > **Note**
   >
   > Anda tidak dapat mengatur opsi `--storage-type` untuk instans DB. Anda dapat mengaturnya hanya untuk klaster DB.

   Misalnya, perintah berikut membuat instans DB yang kompatibel dengan MySQL 5.7 atau MySQL 8.0 bernama `sample-instance`.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance \
        --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
   class db.r5.large
   ```

   Untuk Windows:

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance ^
   ```

```
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-
class db.r5.large
```

Untuk membuat cluster DB PostgreSQL Aurora menggunakan AWS CLI

1. Identifikasi grup subnet DB dan ID grup keamanan VPC untuk cluster DB baru Anda, lalu panggil create-db-cluster AWS CLI perintah untuk membuat cluster Aurora PostgreSQL DB.

   Misalnya, perintah berikut membuat klaster DB baru bernama `sample-cluster`. Klaster ini menggunakan versi mesin default dan jenis penyimpanan Aurora I/O-Optimized.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster \
       --engine aurora-postgresql \
       --storage-type aurora-iopt1 \
       --master-username user-name --manage-master-user-password \
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

   Untuk Windows:

   ```
   aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
       --engine aurora-postgresql ^
       --storage-type aurora-iopt1 ^
       --master-username user-name --manage-master-user-password ^
       --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
   ```

2. Jika Anda menggunakan konsol untuk membuat klaster DB, Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan AWS CLI untuk membuat cluster DB, Anda harus secara eksplisit membuat instance utama untuk cluster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB. Sampai Anda membuat instans DB primer, titik akhir klaster DB tetap dalam status `Creating`.

   Panggil create-db-instance AWS CLI perintah untuk membuat instance utama untuk cluster DB Anda. Sertakan nama klaster DB sebagai nilai opsi `--db-cluster-identifier`.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-instance --db-instance-identifier sample-instance \
   ```

```
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
instance-class db.r5.large
```

Untuk Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance ^
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-
instance-class db.r5.large
```

Contoh ini menentukan opsi `--manage-master-user-password` untuk menghasilkan kata sandi pengguna master dan mengelolanya di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. Alternatifnya, Anda dapat menggunakan opsi `--master-password` untuk menentukan dan mengelola kata sandi sendiri.

API RDS

> **Note**
>
> Sebelum Anda dapat membuat cluster Aurora DB menggunakan AWS CLI, Anda harus memenuhi prasyarat yang diperlukan, seperti membuat VPC dan grup subnet RDS DB. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.

Identifikasi ID grup subnet DB dan grup keamanan VPC untuk klaster DB baru Anda, lalu panggil operasi CreateDBCluster untuk membuat klaster DB.

Saat Anda membuat klaster DB atau instans DB Aurora MySQL versi 2 atau 3, tentukan `aurora-mysql` untuk parameter `Engine`.

Saat Anda membuat klaster DB atau instans DB Aurora PostgreSQL, tentukan `aurora-postgresql` untuk parameter `Engine`.

Jika Anda menggunakan konsol untuk membuat klaster DB, Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan API RDS untuk membuat klaster DB, Anda harus secara eksplisit membuat instans primer untuk klaster DB Anda menggunakan CreateDBInstance. Instans primer adalah instans pertama yang dibuat dalam klaster DB. Sampai Anda membuat instans DB primer, titik akhir klaster DB tetap dalam status `Creating`.

Membuat instance DB primer (penulis)

Jika Anda menggunakan AWS Management Console untuk membuat cluster DB, Amazon RDS secara otomatis membuat instance utama (penulis) untuk cluster DB Anda. Jika Anda menggunakan AWS CLI atau RDS API untuk membuat cluster DB, Anda harus secara eksplisit membuat instance utama untuk cluster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB. Sampai Anda membuat instans DB primer, titik akhir klaster DB tetap dalam status `Creating`.

Untuk informasi selengkapnya, lihat Membuat klaster DB.

> **Note**
>
> Jika Anda memiliki cluster DB tanpa instance DB penulis, juga disebut cluster tanpa kepala, Anda tidak dapat menggunakan konsol untuk membuat instance penulis. Anda harus menggunakan AWS CLI atau RDS API.

Contoh berikut menggunakan create-db-instance AWS CLI perintah untuk membuat instance penulis untuk klaster Aurora PostgreSQL DB bernama. `headless-test`

```
aws rds create-db-instance \
    --db-instance-identifier no-longer-headless \
    --db-cluster-identifier headless-test \
    --engine aurora-postgresql \
    --db-instance-class db.t4g.medium
```

Pengaturan untuk klaster Aurora DB

Tabel berikut berisi detail pengaturan yang Anda pilih saat membuat klaster Aurora DB.

> **Note**
>
> Pengaturan tambahan tersedia jika Anda membuat klaster DB Aurora Serverless v1. Untuk informasi tentang pengaturan ini, lihat Membuat klaster DB Aurora Serverless v1. Selain itu, beberapa pengaturan tidak tersedia untuk Aurora Serverless v1 karena batasan Aurora Serverless v1. Untuk informasi selengkapnya, lihat Batasan Aurora Serverless v1.

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Peningkatan versi minor otomatis	Pilih Aktifkan peningkatan versi minor otomatis jika Anda ingin klaster Aurora DB Anda menerima peningkatan versi minor pilihan di mesin DB secara otomatis saat tersedia. Pengaturan Peningkatan versi minor otomatis berlaku untuk klaster DB Aurora PostgreSQL dan Aurora MySQL. Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora PostgreSQL, lihat Pembaruan Amazon Aurora PostgreSQL. Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora MySQL, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL.	Atur nilai ini untuk setiap instans DB di klaster Aurora Anda. Jika ada instans DB di klaster Anda yang menonaktifkan pengaturan ini, klaster tersebut tidak ditingkatkan secara otomatis. Menggunakan AWS CLI, jalankan `create-db-instance` dan atur `--auto-minor-version-upgrade|--no-auto-minor-version-upgrade` opsi. Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `AutoMinorVersionUpgrade`.
AWS KMS key	Hanya tersedia jika Enkripsi diatur ke Aktifkan enkripsi. Pilih AWS KMS key yang akan digunakan untuk mengenkripsi klaster DB ini. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--kms-key-id` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `KmsKeyId`.
Backtrack	Hanya berlaku untuk Aurora MySQL. Pilih Aktifkan Backtrack untuk mengaktifkan backtrack atau Nonaktifkan Backtrack untuk	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--backtrack-window` opsi.

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
	menonaktifkan backtrack. Dengan backtrack, Anda dapat memundurkan klaster DB ke waktu tertentu, tanpa membuat klaster DB baru. Ini dinonaktifkan secara default. Jika Anda mengaktifkan backtracking, tentukan juga jumlah waktu yang Anda inginkan untuk dapat melakukan backtracking klaster DB Anda (jendela backtrack target). Untuk informasi selengkapnya, lihat Melakukan backtracking klaster DB Aurora.	Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `BacktrackWindow` .
Otoritas sertifikat	Otoritas sertifikat (CA) untuk sertifikat server yang digunakan oleh instans DB di klaster DB. Untuk informasi selengkapnya, lihat .	Menggunakan AWS CLI, jalankan `create-db-instance` dan atur `--ca-certificate-identifier` opsi. Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `CACertificateIdentifier` .
Konfigurasi penyimpanan klaster	Jenis penyimpanan untuk klaster DB: Aurora I/O-Optimized atau Aurora Standard. Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--storage-type` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `StorageType` .

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Salin tanda ke snapshot	Pilih opsi ini untuk menyalin tag instans DB apa pun ke snapshot DB saat Anda membuat snapshot. Untuk informasi selengkapnya, lihat Memberi tag pada sumber daya Amazon RDS.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--copy-tags-to-snapshot | --no-copy-tags-to-snapshot` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `CopyTagsToSnapshot` .

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Autentikasi basis data	Autentikasi basis data yang ingin Anda gunakan. Untuk MySQL: • Pilih Autentikasi kata sandi untuk mengautentikasi pengguna basis data dengan kata sandi basis data saja. • Pilih Kata sandi dan autentikasi basis data IAM untuk mengautentikasi pengguna basis data dengan kata sandi basis data dan kredensial pengguna melalui pengguna dan peran IAM. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM. Untuk PostgreSQL: • Pilih Autentikasi basis data IAM untuk mengautentikasi pengguna basis data dengan kata sandi basis data dan kredensial pengguna melalui pengguna dan peran. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM. • Pilih Autentikasi Kerberos untuk mengautentikasi kata sandi basis data dan kredensial pengguna menggunakan autentikasi Kerberos. Untuk informasi selengkapnya, lihat	Untuk menggunakan otentikasi database IAM dengan AWS CLI, jalankan `create-db-cluster` dan atur opsi. `--enable-iam-database-authentication | --no-enable-iam-database-authentication` Untuk menggunakan autentikasi basis data IAM dengan API RDS, panggil `CreateDBCluster` dan atur parameter `EnableIAMDatabaseAuthentication` . Untuk menggunakan otentikasi Kerberos dengan AWS CLI, jalankan `create-db-cluster` dan atur opsi dan. `--domain --domain-iam-role-name` Untuk menggunakan autentikasi Kerberos dengan API RDS, panggil `CreateDBCluster` dan atur parameter `Domain` dan `DomainIAMRoleName` .

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
	Menggunakan autentikasi Kerberos dengan Aurora PostgreSQL.	
Port basis data	Tentukan port untuk aplikasi dan utilitas yang akan digunakan untuk mengakses basis data. Klaster DB Aurora MySQL diatur secara default ke port MySQL default, 3306, dan klaster DB Aurora PostgreSQL diatur secara default ke port PostgreSQL default, 5432. Firewall di beberapa perusahaan memblokir koneksi ke port default tersebut. Jika firewall perusahaan Anda memblokir port default ini, pilih port lain untuk klaster DB baru.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--port` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `Port`.

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Pengidentifikasi klaster DB	Masukkan nama untuk cluster DB Anda yang unik untuk akun Anda di AWS Wilayah yang Anda pilih. Pengidentifikasi ini digunakan dalam alamat titik akhir klaster untuk klaster DB Anda. Untuk informasi tentang titik akhir klaster, lihat Manajemen koneksi Amazon Aurora. Pengidentifikasi klaster DB memiliki batasan berikut: • Pengidentifikasi ini harus berisi 1 hingga 63 karakter alfanumerik atau tanda hubung. • Karakter pertamanya harus berupa huruf. • Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut. • Ini harus unik untuk semua cluster DB per AWS akun, per AWS Wilayah.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--db-cluster-identifier` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `DBCluster Identifier` .

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Grup parameter klaster DB	Pilih grup parameter klaster DB. Aurora memiliki grup parameter klaster DB default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter klaster DB Anda sendiri. Untuk informasi selengkapnya tentang grup parameter klaster DB, lihat Bekerja dengan grup parameter.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--db-cluster-parameter-group-name` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `DBClusterParameterGroupName`.
Kelas instans DB	Hanya berlaku untuk jenis kapasitas terprovisi. Pilih kelas instans DB yang menentukan persyaratan pemrosesan dan memori untuk setiap instans dalam klaster DB. Untuk informasi selengkapnya tentang kelas instans DB, lihat Kelas instans DB Aurora.	Atur nilai ini untuk setiap instans DB di klaster Aurora Anda. Menggunakan AWS CLI, jalankan `create-db-instance` dan atur `--db-instance-class` opsi. Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `DBInstanceClass`.
Grup parameter DB	Pilih grup parameter. Aurora memiliki grup parameter default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter Anda sendiri. Untuk informasi selengkapnya tentang grup parameter, lihat Bekerja dengan grup parameter.	Atur nilai ini untuk setiap instans DB di klaster Aurora Anda. Menggunakan AWS CLI, jalankan `create-db-instance` dan atur `--db-parameter-group-name` opsi. Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `DBParameterGroupName`.

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Grup subnet DB	Grup subnet DB yang Anda ingin gunakan untuk klaster DB. Pilih Pilih yang ada untuk menggunakan grup subnet DB yang ada. Kemudian pilih grup subnet yang diperlukan dari daftar dropdown Grup subnet DB yang ada. Pilih Pengaturan otomatis untuk membiarkan RDS memilih grup subnet DB yang kompatibel. Jika tidak ada, RDS membuat grup subnet baru untuk klaster Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--db-subnet-group-name` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `DBSubnetGroupName` .
Aktifkan perlindungan penghapusan	Pilih Aktifkan perlindungan penghapusan agar klaster DB Anda tidak terhapus. Jika Anda membuat klaster DB produksi dengan konsol, perlindungan penghapusan diaktifkan secara default.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--deletion-protection | --no-deletion-protection` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `DeletionProtection` .

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Aktifkan enkripsi	Pilih `Enable encryption` untuk mengaktifkan enkripsi saat diam untuk klaster DB ini. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--storage-encrypted | --no-storage-encrypted` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `StorageEncrypted`.
Aktifkan Pemantauan yang Ditingkatkan	Pilih Aktifkan pemantauan yang ditingkatkan untuk mengaktifkan metrik pengumpulan secara waktu nyata untuk sistem operasi tempat klaster DB Anda berjalan. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan.	Atur nilai-nilai ini untuk setiap instans DB di klaster Aurora Anda. Menggunakan AWS CLI, menjalankan `create-db-instance` dan mengatur `--monitoring-interval` dan `--monitoring-role-arn` pilihan. Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `MonitoringInterval` dan `MonitoringRoleArn`.

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Aktifkan API Data RDS	Pilih Aktifkan API Data RDS untuk mengaktifkan RDS Data API (Data API). Data API menyediakan endpoint HTTP yang aman untuk menjalankan pernyataan SQL tanpa mengelola koneksi. Untuk informasi selengkapnya, lihat Menggunakan RDS Data API.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--enable-http-endpoint | --no-enable-http-endpoint` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `EnableHttpEndpoint` .
Jenis mesin	Pilih mesin basis data yang akan digunakan untuk klaster DB ini.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--engine` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `Engine`.
Versi mesin	Hanya berlaku untuk jenis kapasitas terprovisi. Pilih nomor versi mesin DB Anda.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--engine-version` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `EngineVersion` .

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Prioritas failover	Pilih prioritas failover untuk instans. Jika Anda tidak memilih nilai, nilai default-nya adalah tier-1. Prioritas ini akan menentukan urutan promosi Aurora Replika saat melakukan pemulihan dari kegagalan instans primer. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora.	Atur nilai ini untuk setiap instans DB di klaster Aurora Anda. Menggunakan AWS CLI, jalankan `create-db-instance` dan atur `--promotion-tier` opsi. Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `PromotionTier` .

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Nama basis data awal	Masukkan nama untuk basis data default Anda. Jika Anda tidak memberikan nama untuk klaster DB Aurora MySQL, Amazon RDS tidak membuat basis data pada klaster DB yang sedang Anda buat. Jika Anda tidak memberikan nama untuk klaster DB Aurora PostgreSQL, Amazon RDS membuat basis data bernama `postgres`. Untuk Aurora MySQL, nama basis data default memiliki batasan ini: • Harus berisi antara 1–64 karakter alfanumerik. • Tidak boleh menggunakan kata yang dicadangkan oleh mesin basis data. Untuk Aurora PostgreSQL, nama basis data default memiliki batasan ini: • Harus berisi antara 1–63 karakter alfanumerik. • Harus dimulai dengan huruf. Karakter selanjutnya dapat berupa huruf, garis bawah, atau digit (0–9). • Tidak boleh menggunakan kata yang dicadangkan oleh mesin basis data.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--database-name` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `DatabaseName`.

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
	Untuk membuat basis data tambahan, hubungkan ke klaster DB dan gunakan perintah SQL CREATE DATABASE. Untuk informasi selengkapnya tentang menghubungkan ke klaster DB, lihat Menghubungkan ke klaster DB Amazon Aurora.	
Ekspor log	Di bagian Log ekspor, pilih log yang ingin Anda mulai terbitkan ke Amazon CloudWatch Logs. Untuk informasi selengkapnya tentang menerbitkan log MySQL Aurora ke Log, lihat. CloudWatch Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch Untuk informasi selengkapnya tentang memublikasikan log PostgreSQL Aurora ke Log, lihat. CloudWatch Menerbitkan log Aurora PostgreSQL ke Amazon Logs CloudWatch	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--enable-cloudwatch-logs-exports` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `EnableCloudwatchLogsExports` .
Jendela pemeliharaan	Klik Pilih jendela dan tentukan rentang waktu mingguan saat pemeliharaan sistem dapat dilakukan. Atau pilih Tidak ada preferensi agar Amazon RDS menetapkan periode secara acak.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--preferred-maintenance-window` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `PreferredMaintenanceWindow` .

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Kelola kredensi master di AWS Secrets Manager	Pilih Kelola kredensial master di AWS Secrets Manager untuk mengelola kata sandi pengguna master dalam rahasia di Secrets Manager. Anda juga dapat memilih kunci KMS yang akan digunakan untuk melindungi rahasia. Pilih dari kunci KMS di akun Anda, atau masukkan kunci dari akun yang berbeda. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager.	Menggunakan AWS CLI, menjalankan `create-db-cluster`dan mengatur `--manage-master-user-password | --no-manage-master-user-password` dan `--master-user-secret-kms-key-id` pilihan. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `ManageMasterUserPassword` dan `MasterUserSecretKmsKeyId` .
Kata sandi master	Masukkan kata sandi untuk masuk ke klaster DB Anda: • Untuk Aurora MySQL, kata sandi harus berisi 8–41 karakter ASCII yang dapat dicetak. • Untuk Aurora PostgreSQL, kata sandi harus berisi 8–99 karakter ASCII yang dapat dicetak. • Kata sandi tidak dapat berisi `/`, `"`, `@`, atau spasi.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--master-user-password` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `MasterUserPassword` .

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Nama pengguna master	Masukkan nama yang akan digunakan sebagai nama pengguna master untuk masuk ke klaster DB Anda: • Untuk Aurora MySQL, nama harus berisi 1–16 karakter alfanumerik. • Untuk Aurora PostgreSQL, nama harus berisi 1–63 karakter alfanumerik. • Karakter pertama harus berupa huruf. • Nama tidak boleh menggunakan kata yang dicadangkan oleh mesin basis data. Anda tidak dapat mengubah nama pengguna master setelah klaster DB dibuat.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--master-username` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `MasterUsername` .

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Deployment Multi-AZ	Hanya berlaku untuk jenis kapasitas terprovisi. Tentukan apakah Anda ingin membuat Replika Aurora di Zona Ketersediaan lainnya untuk dukungan failover. Jika Anda memilih Buat Replika di Zona yang Berbeda, Amazon RDS akan membuat Replika Aurora untuk Anda dalam klaster DB Anda di Zona Ketersediaan yang berbeda dari instans primer untuk klaster DB Anda. Untuk informasi selengkapnya tentang beberapa Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--availability-zones` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `AvailabilityZones` .

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Jenis jaringan	Protokol alamat IP yang didukung oleh klaster DB. IPv4 untuk menentukan bahwa sumber daya dapat berkomunikasi dengan klaster DB hanya melalui protokol alamat IPv4. Mode tumpukan ganda untuk menentukan bahwa sumber daya dapat berkomunikasi dengan klaster DB melalui IPv4, IPv6, atau keduanya. Gunakan mode tumpukan ganda jika Anda memiliki sumber daya yang harus berkomunikasi dengan klaster DB Anda melalui protokol alamat IPv6. Untuk menggunakan mode tumpukan ganda, pastikan ada setidaknya dua subnet yang mencakup dua Zona Ketersediaan yang mendukung protokol jaringan IPv4 dan IPv6. Selain itu, pastikan Anda mengaitkan blok CIDR IPv6 dengan subnet di grup subnet DB yang Anda tentukan. Untuk informasi selengkapnya, lihat Penentuan alamat IP Amazon Aurora.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--network-type` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `NetworkType` .

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Akses publik	Pilih Dapat diakses publik untuk memberikan alamat IP publik pada klaster DB, atau pilih Tidak dapat diakses publik. Instans dalam klaster DB Anda dapat berupa campuran antara instans DB publik dan privat. Untuk informasi selengkapnya tentang menyembunyikan instans dari akses publik, lihat Menyembunyikan instans DB dalam VPC dari internet. Untuk terhubung ke instans DB dari luar Amazon VPC-nya, instans DB harus dapat diakses secara publik, akses harus diberikan menggunakan aturan masuk grup keamanan instans DB, dan persyaratan lain harus dipenuhi. Untuk informasi selengkapnya, lihat Tidak dapat terhubung ke instans DB Amazon RDS. Jika instans DB Anda tidak dapat diakses publik, Anda juga dapat menggunakan koneksi VPN AWS Site-to-Site AWS Direct Connect atau koneksi untuk mengaksesnya dari jaringan pribadi. Untuk informasi selengkapnya, lihat Privasi lalu lintas antarjaringan.	Atur nilai ini untuk setiap instans DB di klaster Aurora Anda. Menggunakan AWS CLI, jalankan `create-db-instance` dan atur `--publicly-accessible | --no-publicly-accessible` opsi. Menggunakan API RDS, panggil `CreateDBInstance` dan tetapkan parameter `PubliclyAccessible` .

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
RDS Extended Support	Pilih Aktifkan RDS Extended Support untuk memungkinkan versi mesin utama yang didukung untuk terus berjalan melewati akhir Aurora dari tanggal dukungan standar. Saat Anda membuat cluster DB, Amazon Aurora default ke RDS Extended Support. Untuk mencegah pembuatan cluster DB baru setelah Aurora berakhir pada tanggal dukungan standar dan untuk menghindari biaya untuk RDS Extended Support, nonaktifkan pengaturan ini. Cluster DB Anda yang ada tidak akan dikenakan biaya hingga tanggal mulai penetapan harga RDS Extended Support. Untuk informasi selengkapnya, lihat Menggunakan Dukungan Diperpanjang Amazon RDS.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--engine-lifecycle-support` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `EngineLifecycleSupport` .
Proksi RDS	Pilih Buat Proksi RDS guna membuat proksi untuk klaster DB Anda. Amazon RDS secara otomatis membuat peran IAM dan rahasia Secrets Manager untuk proksi. Untuk informasi selengkapnya, lihat Menggunakan Proksi Amazon RDS untuk Aurora.	Tidak tersedia saat membuat klaster DB.

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Periode retensi	Pilih durasi waktu, dari 1 hingga 35 hari, saat Aurora harus mempertahankan salinan cadangan basis data. Salinan cadangan dapat digunakan untuk point-in-time mengembalikan (PITR) database Anda hingga yang kedua.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--backup-retention-period` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `BackupRetentionPeriod` .
Nyalakan DevOps Guru	Pilih Aktifkan DevOps Guru untuk mengaktifkan Amazon DevOps Guru untuk database Aurora Anda. Agar DevOps Guru for RDS dapat memberikan analisis terperinci tentang anomali kinerja, Performance Insights harus diaktifkan. Untuk informasi selengkapnya, lihat Menyiapkan DevOps Guru untuk RDS.	Anda dapat mengaktifkan DevOps Guru untuk RDS dari dalam konsol RDS, tetapi tidak dengan menggunakan RDS API atau CLI. Untuk informasi selengkapnya tentang mengaktifkan DevOps Guru, lihat Panduan Pengguna Amazon DevOps Guru.

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Aktifkan Wawasan Performa	Pilih Aktifkan Wawasan Performa untuk mengaktifkan Wawasan Performa Amazon RDS. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.	Atur nilai-nilai ini untuk setiap instans DB di klaster Aurora Anda. Menggunakan AWS CLI, jalankan `create-db-instance` dengan `--enable-performance-insights | --no-enable-performance-insights` dan tetapkan opsi `--performance-insights-kms-key-id` dan `--performance-insights-retention-period`. Menggunakan API RDS, panggil `CreateDBInstance` `EnablePerformanceInsights` dan tetapkan parameter `PerformanceInsightsKMSKeyId` dan `PerformanceInsightsRetentionPeriod`.
Cloud Privat Virtual (VPC)	Pilih VPC untuk meng-host klaster DB. Pilih Buat VPC Baru agar Amazon RDS membuat VPC untuk Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.	Untuk API AWS CLI dan, Anda menentukan ID grup keamanan VPC.

Pengaturan konsol	Deskripsi pengaturan	Opsi CLI dan parameter API RDS
Grup keamanan VPC (firewall)	Pilih Buat baru agar Amazon RDS membuat grup keamanan VPC untuk Anda. Atau klik Pilih yang ada dan tentukan satu atau beberapa grup keamanan VPC untuk mengamankan akses jaringan ke klaster DB. Saat Anda memilih Buat baru pada konsol RDS, grup keamanan baru dibuat dengan peran masuk yang memungkinkan akses ke instans DB dari alamat IP yang terdeteksi di browser Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.	Menggunakan AWS CLI, jalankan `create-db-cluster` dan atur `--vpc-security-group-ids` opsi. Menggunakan API RDS, panggil `CreateDBCluster` dan tetapkan parameter `VpcSecurityGroupIds` .

Pengaturan yang tidak berlaku untuk klaster DB Amazon Aurora

Pengaturan berikut dalam AWS CLI perintah `create-db-cluster`dan operasi RDS API `CreateDBCluster`tidak berlaku untuk cluster Amazon Aurora DB.

> **Note**
>
> AWS Management Console Tidak menampilkan pengaturan ini untuk cluster Aurora DB.

AWS CLI pengaturan	Pengaturan API RDS
`--allocated-storage`	`AllocatedStorage`
`--auto-minor-version-upgrade | --no-auto-minor-version-upgrade`	`AutoMinorVersionUpgrade`

AWS CLI pengaturan	Pengaturan API RDS
`--db-cluster-instance-class`	`DBClusterInstanceClass`
`--enable-performance-insights | --no-enable-performance-insights`	`EnablePerformanceInsights`
`--iops`	`Iops`
`--monitoring-interval`	`MonitoringInterval`
`--monitoring-role-arn`	`MonitoringRoleArn`
`--option-group-name`	`OptionGroupName`
`--performance-insights-kms-key-id`	`PerformanceInsightsKMSKeyId`
`--performance-insights-retention-period`	`PerformanceInsightsRetentionPeriod`
`--publicly-accessible | --no-publicly-accessible`	`PubliclyAccessible`

Pengaturan yang tidak berlaku untuk instans DB Amazon Aurora

Pengaturan berikut dalam AWS CLI perintah `create-db-instance`dan operasi RDS API `CreateDBInstance`tidak berlaku untuk instans DB Amazon Aurora DB cluster.

> **Note**
>
> AWS Management Console Tidak menampilkan pengaturan ini untuk instans Aurora DB.

AWS CLI pengaturan	Pengaturan API RDS
`--allocated-storage`	`AllocatedStorage`

AWS CLI pengaturan	Pengaturan API RDS
--availability-zone	AvailabilityZone
--backup-retention-period	BackupRetentionPeriod
--backup-target	BackupTarget
--character-set-name	CharacterSetName
--character-set-name	CharacterSetName
--custom-iam-instance-profile	CustomIamInstanceProfile
--db-security-groups	DBSecurityGroups
--deletion-protection | --no-deletion-protection	DeletionProtection
--domain	Domain
--domain-iam-role-name	DomainIAMRoleName
--enable-cloudwatch-logs-exports	EnableCloudwatchLogsExports
--enable-customer-owned-ip | --no-enable-customer-owned-ip	EnableCustomerOwnedIp
--enable-iam-database-authentication | --no-enable-iam-database-authentication	EnableIAMDatabaseAuthentication
--engine-version	EngineVersion
--iops	Iops
--kms-key-id	KmsKeyId
--master-username	MasterUsername
--master-user-password	MasterUserPassword

AWS CLI pengaturan	Pengaturan API RDS
`--max-allocated-storage`	`MaxAllocatedStorage`
`--multi-az | --no-multi-az`	`MultiAZ`
`--nchar-character-set-name`	`NcharCharacterSetName`
`--network-type`	`NetworkType`
`--option-group-name`	`OptionGroupName`
`--preferred-backup-window`	`PreferredBackupWindow`
`--processor-features`	`ProcessorFeatures`
`--storage-encrypted | --no-storage-encrypted`	`StorageEncrypted`
`--storage-type`	`StorageType`
`--tde-credential-arn`	`TdeCredentialArn`
`--tde-credential-password`	`TdeCredentialPassword`
`--timezone`	`Timezone`
`--vpc-security-group-ids`	`VpcSecurityGroupIds`

Membuat sumber daya Amazon Aurora dengan AWS CloudFormation

Amazon Aurora terintegrasi dengan AWS CloudFormation, layanan yang membantu Anda memodelkan dan mengatur sumber daya AWS, sehingga Anda dapat menghabiskan lebih sedikit waktu untuk membuat dan mengelola sumber daya dan infrastruktur Anda. Anda membuat templat yang menjelaskan semua sumber daya AWS yang Anda inginkan (seperti klaster basis data dan grup parameter klaster basis data), dan AWS CloudFormation menyediakan dan mengonfigurasi sumber daya tersebut untuk Anda.

Saat menggunakan AWS CloudFormation, Anda dapat menggunakan kembali templat Anda untuk mengatur sumber daya Aurora secara konsisten dan berulang kali. Jelaskan sumber daya Anda satu kali, lalu sediakan sumber daya yang sama berulang-ulang dalam beberapa akun dan Wilayah AWS.

Aurora dan templat AWS CloudFormation

Untuk menyediakan dan mengonfigurasi sumber daya bagi Aurora dan layanan terkait, Anda harus memahami templat AWS CloudFormation. Templat adalah file teks dengan format JSON atau YAML. Templat ini menjelaskan sumber daya yang ingin Anda sediakan di tumpukan AWS CloudFormation. Jika tidak terbiasa dengan JSON atau YAML, Anda dapat menggunakan Desainer AWS CloudFormation untuk membantu Anda memulai dengan templat AWS CloudFormation. Lihat informasi selengkapnya di Apa yang dimaksud dengan Desainer AWS CloudFormation? dalam Panduan Pengguna AWS CloudFormation.

Aurora mendukung pembuatan sumber daya di AWS CloudFormation. Lihat informasi selengkapnya, termasuk contoh templat JSON dan YAML untuk sumber daya ini, di referensi jenis sumber daya RDS dalam Panduan Pengguna AWS CloudFormation.

Pelajari selengkapnya tentang AWS CloudFormation

Untuk mempelajari selengkapnya tentang AWS CloudFormation, lihat sumber daya berikut:

- AWS CloudFormation
- Panduan Pengguna AWS CloudFormation
- Referensi API AWS CloudFormation
- Panduan Pengguna Antarmuka Baris Perintah AWS CloudFormation

Menghubungkan ke klaster DB Amazon Aurora

Anda dapat menghubungkan ke klaster DB Aurora menggunakan alat yang sama yang Anda gunakan untuk menghubungkan ke basis data MySQL atau PostgreSQL. Anda menentukan string koneksi dengan skrip, utilitas, atau aplikasi apa pun yang terhubung ke instans DB MySQL atau PostgreSQL. Anda menggunakan kunci publik yang sama untuk koneksi Lapisan Soket Aman (SSL).

Dalam string koneksi, Anda biasanya menggunakan informasi host dan port dari titik akhir khusus yang terkait dengan klaster DB. Dengan titik akhir ini, Anda dapat menggunakan parameter koneksi yang sama terlepas dari berapa banyak instans DB yang ada di klaster. Anda juga menggunakan informasi host dan port dari instans DB tertentu di klaster DB Aurora Anda untuk tugas-tugas khusus, seperti pemecahan masalah.

> **Note**
>
> Untuk klaster DB Aurora Serverless, Anda terhubung ke titik akhir basis data bukan instans DB. Anda dapat menemukan titik akhir basis data untuk klaster DB Aurora Serverless pada tab Konektivitas & keamanan di AWS Management Console. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Serverless v1.

Terlepas dari mesin DB Aurora dan alat-alat khusus yang Anda gunakan untuk menangani klaster atau instans DB, titik akhir harus dapat diakses. Klaster DB Amazon Aurora hanya dapat dibuat di cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. Artinya, Anda mengakses titik akhir dari dalam VPC atau luar VPC menggunakan salah satu pendekatan berikut.

- Akses klaster DB Amazon Aurora di dalam VPC – Aktifkan akses ke klaster DB Amazon Aurora melalui VPC. Anda melakukannya dengan mengedit Aturan masuk pada Grup keamanan untuk VPC untuk mengizinkan akses ke klaster DB Aurora spesifik. Untuk mempelajari selengkapnya, termasuk cara mengonfigurasi VPC Anda untuk skenario klaster DB Aurora yang berbeda-beda, lihat VPC Amazon Virtual Private Cloud dan Amazon Aurora.
- Mengakses klaster DB Amazon Aurora di luar VPC – Untuk mengakses klaster DB Amazon Aurora dari luar VPC, gunakan alamat titik akhir publik klaster DB Amazon Aurora.

Untuk informasi selengkapnya, lihat Pemecahan masalah kegagalan koneksi Aurora.

Daftar Isi

- Menghubungkan ke klaster DB Amazon Aurora MySQL
  - Utilitas koneksi untuk Aurora MySQL
  - Menghubungkan dengan Aurora MySQL menggunakan utilitas MySQL
  - Menghubungkan dengan Driver Amazon Web Services (AWS) JDBC
  - Menghubungkan dengan SSL untuk Aurora MySQL
- Menghubungkan ke klaster DB Amazon Aurora PostgreSQL.
  - Utilitas koneksi untuk Aurora PostgreSQL
  - Menghubungkan dengan Driver AWS JDBC untuk PostgreSQL
- Pemecahan masalah kegagalan koneksi Aurora

Menghubungkan ke klaster DB Amazon Aurora MySQL

Untuk mengautentikasi ke cluster DB MySQL Aurora Anda, Anda dapat menggunakan nama pengguna MySQL dan otentikasi kata sandi atau otentikasi basis data (IAM). AWS Identity and Access Management Untuk informasi selengkapnya tentang menggunakan autentikasi nama pengguna dan kata sandi MySQL, lihat Access control and account management dalam dokumentasi MySQL. Untuk informasi selengkapnya tentang menggunakan autentikasi basis data IAM, lihat Autentikasi basis data IAM.

Saat Anda memiliki koneksi ke klaster DB Amazon Aurora dengan kompatibilitas MySQL 8.0, Anda dapat menjalankan perintah SQL yang kompatibel dengan MySQL versi 8.0. Versi minimum yang kompatibel adalah MySQL 8.0.23. Untuk informasi selengkapnya tentang sintaksis SQL MySQL 8.0, lihat Panduan referensi MySQL 8.0. Untuk informasi tentang batasan yang berlaku untuk Aurora MySQL versi 3, lihat Perbandingan Aurora MySQL versi 3 dan MySQL 8.0 Community Edition.

Saat Anda memiliki koneksi ke klaster DB Amazon Aurora dengan kompatibilitas MySQL 5.7, Anda dapat menjalankan perintah SQL yang kompatibel dengan MySQL versi 5.7. Untuk informasi selengkapnya tentang sintaksis SQL MySQL 5.7, lihat Panduan referensi MySQL 5.7. Untuk informasi tentang batasan yang berlaku untuk Aurora MySQL 5.7, lihat Aurora MySQL versi 2 yang kompatibel dengan MySQL 5.7.

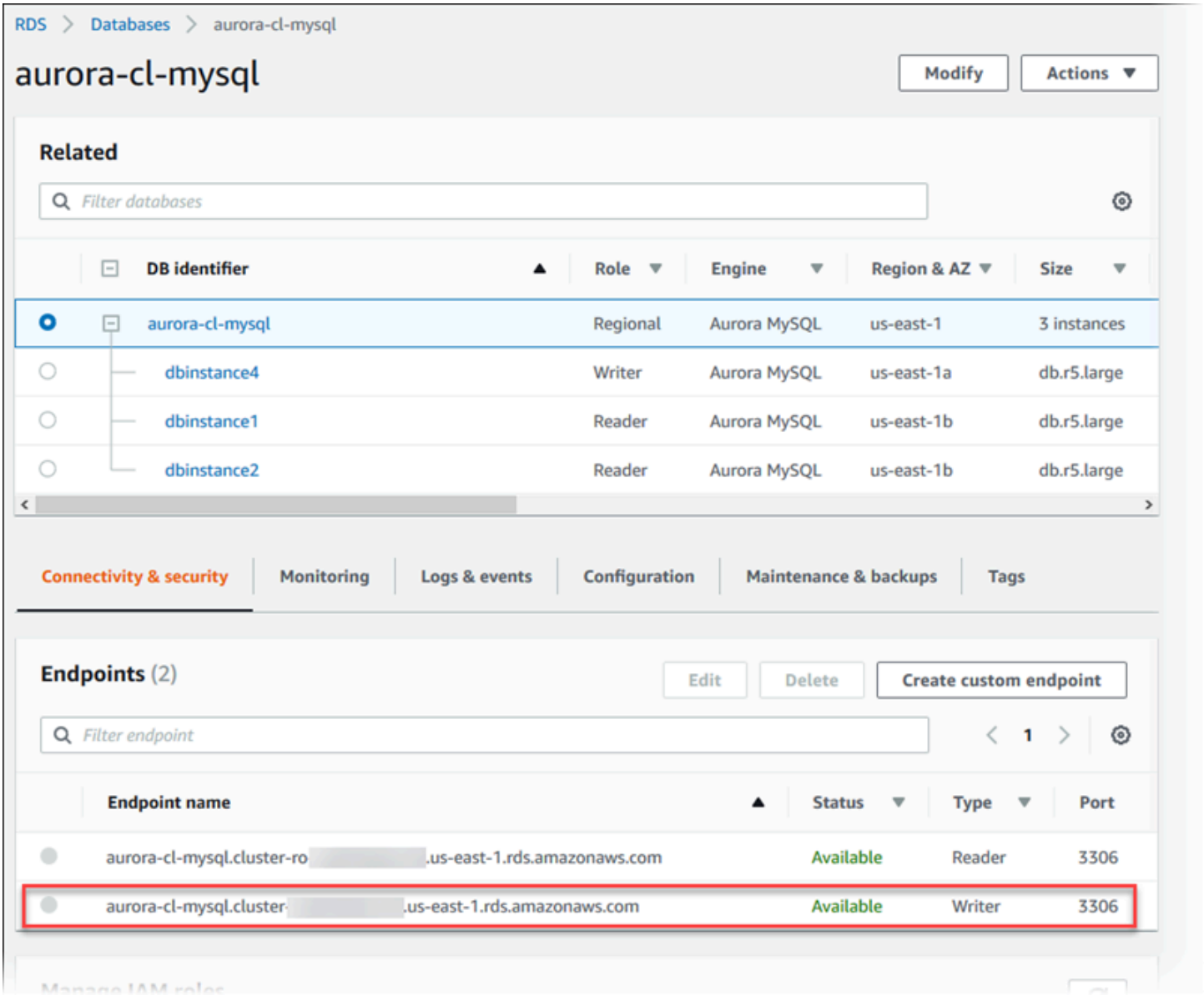
> **Note**
>
> Untuk panduan berguna dan mendetail tentang menghubungkan ke klaster DB Amazon Aurora MySQL, Anda dapat melihat buku panduan Manajemen koneksi Aurora.

Dalam tampilan detail untuk klaster DB Anda, Anda dapat menemukan titik akhir klaster, yang dapat Anda gunakan dalam string koneksi MySQL Anda. Titik akhir terdiri dari nama domain dan port untuk klaster DB Anda. Misalnya, jika nilai titik akhir adalah `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com:3306`, maka Anda menentukan nilai berikut dalam string koneksi MySQL:

- Untuk host atau nama host, tentukan `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`
- Untuk port, tentukan `3306` atau nilai port yang Anda gunakan saat Anda membuat klaster DB

Titik akhir klaster menghubungkan Anda ke instans primer untuk klaster DB. Anda dapat melakukan operasi baca dan tulis menggunakan titik akhir klaster. Klaster DB Anda juga dapat memiliki hingga 15 Replika Aurora yang mendukung akses hanya baca ke data di klaster DB Anda. Instans primer dan setiap Replika Aurora memiliki titik akhir unik yang tidak bergantung pada titik akhir klaster dan memungkinkan Anda terhubung ke instans DB tertentu dalam klaster secara langsung. Titik akhir klaster selalu mengarah ke instans primer. Jika instans primer gagal dan diganti, maka titik akhir klaster mengarah ke instans primer baru.

Untuk melihat titik akhir klaster (titik akhir penulis), pilih Basis data di konsol Amazon RDS dan pilih nama klaster DB untuk menampilkan detail klaster DB.



Topik

- Utilitas koneksi untuk Aurora MySQL
- Menghubungkan dengan Aurora MySQL menggunakan utilitas MySQL
- Menghubungkan dengan Driver Amazon Web Services (AWS) JDBC
- Menghubungkan dengan SSL untuk Aurora MySQL

Utilitas koneksi untuk Aurora MySQL

Beberapa utilitas koneksi yang dapat Anda gunakan adalah sebagai berikut:

- Baris perintah – Anda dapat terhubung ke klaster DB Amazon Aurora dengan menggunakan alat seperti utilitas baris perintah MySQL. Untuk informasi selengkapnya tentang menggunakan utilitas MySQL, lihat mysql — the MySQL command-line client dalam dokumentasi MySQL.
- GUI – Anda dapat menggunakan utilitas MySQL Workbench untuk terhubung dengan menggunakan antarmuka UI. Untuk informasi selengkapnya, lihat halaman Unduh MySQL Workbench.
- Aplikasi - Anda dapat menggunakan Driver Amazon Web Services (AWS) JDBC untuk menghubungkan aplikasi klien Anda ke cluster DB MySQL Aurora. Untuk informasi selengkapnya, lihat Menghubungkan dengan Driver Amazon Web Services (AWS) JDBC.

Menghubungkan dengan Aurora MySQL menggunakan utilitas MySQL

Gunakan prosedur berikut: Prosedur ini mengasumsikan bahwa Anda mengonfigurasi klaster DB Anda di subnet privat di VPC Anda. Anda terhubung menggunakan instans Amazon EC2 yang dikonfigurasi sesuai dengan tutorial dalam Tutorial: Membuat server web dan klaster DB Amazon Aurora.

> **Note**
>
> Prosedur ini tidak memerlukan penginstalan server web dalam tutorial tersebut, tetapi memerlukan penginstalan MariaDB 10.5.

Untuk terhubung ke klaster DB menggunakan utilitas MySQL

1. Masuk ke instans EC2 yang Anda gunakan untuk terhubung ke klaster DB Anda.

   Anda akan melihat output seperti yang berikut ini.

```
Last login: Thu Jun 23 13:32:52 2022 from xxx.xxx.xxx.xxx

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-10-0-xxx.xxx ~]$
```

2. Ketikkan perintah berikut pada baris perintah untuk terhubung ke instans DB primer dari klaster DB Anda.

   Untuk parameter `-h`, ganti nama DNS titik akhir untuk instans primer Anda. Untuk parameter `-u`, ganti ID pengguna untuk akun pengguna basis data.

   ```
   mysql -h primary-instance-endpoint.AWS_account.AWS_Region.rds.amazonaws.com -P 3306
    -u database_user -p
   ```

   Contoh:

   ```
   mysql -h my-aurora-cluster-instance.c1xy5example.123456789012.eu-
   central-1.rds.amazonaws.com -P 3306 -u admin -p
   ```

3. Masukkan kata sandi untuk pengguna basis data.

   Anda akan melihat output seperti yang berikut ini.

   ```
   Welcome to the MariaDB monitor.  Commands end with ; or \g.
   Your MySQL connection id is 1770
   Server version: 8.0.23 Source distribution

   Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

   Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

   MySQL [(none)]>
   ```

4. Masukkan perintah SQL Anda.

Menghubungkan dengan Driver Amazon Web Services (AWS) JDBC

Amazon Web Services (AWS) JDBC Driver telah didesain ulang sebagai pembungkus JDBC tingkat lanjut. Pembungkus ini melengkapi dan memperluas fungsionalitas driver JDBC yang ada untuk membantu aplikasi memanfaatkan fitur database berkerumun seperti Aurora MySQL. Driver ini kompatibel dengan driver MySQL Connector/J. Untuk menginstal atau meningkatkan konektor Anda, ganti file konektor MySQL .jar (terletak di aplikasi CLASSPATH) dengan file JDBC Driver .jar, dan AWS perbarui awalan URL koneksi dari ke. `jdbc:mysql://` `jdbc:aws-wrapper:mysql://`

Driver AWS JDBC memanfaatkan sepenuhnya kemampuan failover Aurora MySQL. Jika terjadi failover, driver ini akan mengueri klaster secara langsung untuk topologi baru alih-alih menggunakan resolusi DNS. Dengan mengueri klaster secara langsung, driver ini dapat terhubung ke basis data primer baru lebih cepat dengan cara yang andal dan dapat diprediksi. Pendekatan ini membantu menghindari potensi penundaan yang disebabkan oleh resolusi DNS.

Untuk informasi selengkapnya tentang Driver AWS JDBC dan petunjuk lengkap untuk menggunakannya, lihat repositori Amazon Web Services (AWS) JDBC Driver. GitHub

Driver AWS JDBC mendukung otentikasi database IAM. Untuk informasi selengkapnya, lihat Plugin Autentikasi AWS IAM di repositori Driver AWS JDBC. GitHub Untuk mengetahui informasi selengkapnya tentang autentikasi basis data IAM, lihat Autentikasi basis data IAM.

> **Note**
>
> Versi 3.0.3 dari utilitas MariaDB Connector/J menjatuhkan dukungan untuk cluster Aurora DB, jadi kami sangat menyarankan pindah ke Driver JDBC. AWS Driver AWS JDBC menawarkan peningkatan kecepatan failover untuk cluster Aurora MySQL DB.
> Driver AWS JDBC untuk MySQL akan mencapai akhir dukungan pada 25 Juli 2024. Kami menyarankan Anda mulai menggunakan Driver AWS JDBC baru sebelum tanggal ini. Untuk informasi selengkapnya, lihat Jadwal rilis dan kebijakan pemeliharaan di repositori Amazon Web Services JDBC Driver for MySQL. GitHub

Menghubungkan dengan SSL untuk Aurora MySQL

Anda dapat menggunakan enkripsi SSL pada koneksi ke instans DB Amazon Aurora MySQL. Untuk informasi, lihat Menggunakan TLS dengan klaster DB Aurora MySQL.

Untuk terhubung menggunakan SSL, gunakan utilitas MySQL yang dijelaskan dalam prosedur berikut. Jika Anda menggunakan autentikasi basis data IAM, Anda harus menggunakan koneksi SSL. Untuk informasi, lihat Autentikasi basis data IAM.

> **Note**
>
> Untuk terhubung ke titik akhir klaster menggunakan SSL, utilitas koneksi klien Anda harus mendukung Nama Alternatif Subjek (SAN). Jika koneksi klien Anda tidak mendukung SAN, Anda dapat terhubung langsung ke instans di klaster DB Aurora Anda. Untuk informasi selengkapnya tentang titik akhir Aurora, lihat Manajemen koneksi Amazon Aurora.

Untuk terhubung ke klaster DB dengan SSL menggunakan utilitas MySQL

1. Unduh kunci publik untuk sertifikat penandatanganan Amazon RDS.

   Untuk informasi tentang mengunduh sertifikat, lihat .

2. Ketikkan perintah berikut pada baris perintah untuk terhubung ke instans primer dari klaster DB dengan SSL menggunakan utilitas MySQL. Untuk parameter `-h`, ganti nama DNS titik akhir untuk instans primer Anda. Untuk parameter `-u`, ganti ID pengguna untuk akun pengguna basis data. Untuk parameter `--ssl-ca`, ganti nama file sertifikat SSL yang sesuai. Ketikkan kata sandi pengguna master saat diminta.

   ```
   mysql -h mycluster-primary.123456789012.us-east-1.rds.amazonaws.com -u admin_user -p --ssl-ca=[full path]global-bundle.pem --ssl-verify-server-cert
   ```

Anda akan melihat output seperti yang berikut ini.

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 350
Server version: 8.0.26-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Untuk petunjuk umum tentang membuat string koneksi RDS for MySQL dan menemukan kunci publik untuk koneksi SSL, lihat Menghubungkan ke instans DB yang menjalankan mesin basis data MySQL.

Menghubungkan ke klaster DB Amazon Aurora PostgreSQL.

Anda dapat terhubung ke instans DB di klaster DB Amazon Aurora PostgreSQL menggunakan alat yang sama yang Anda gunakan untuk menghubungkan ke basis data PostgreSQL. Sebagai bagian dari hal ini, Anda menggunakan kunci publik yang sama untuk koneksi Lapisan Soket Aman (SSL). Anda dapat menggunakan informasi titik akhir dan port dari instans primer atau Replika Aurora di klaster DB Aurora PostgreSQL Anda dalam string koneksi skrip, utilitas, atau aplikasi apa pun yang terhubung ke instans DB PostgreSQL. Dalam string koneksi, tentukan alamat DNS dari instans primer atau titik akhir Replika Aurora sebagai parameter host. Tentukan nomor port dari titik akhir sebagai parameter port.

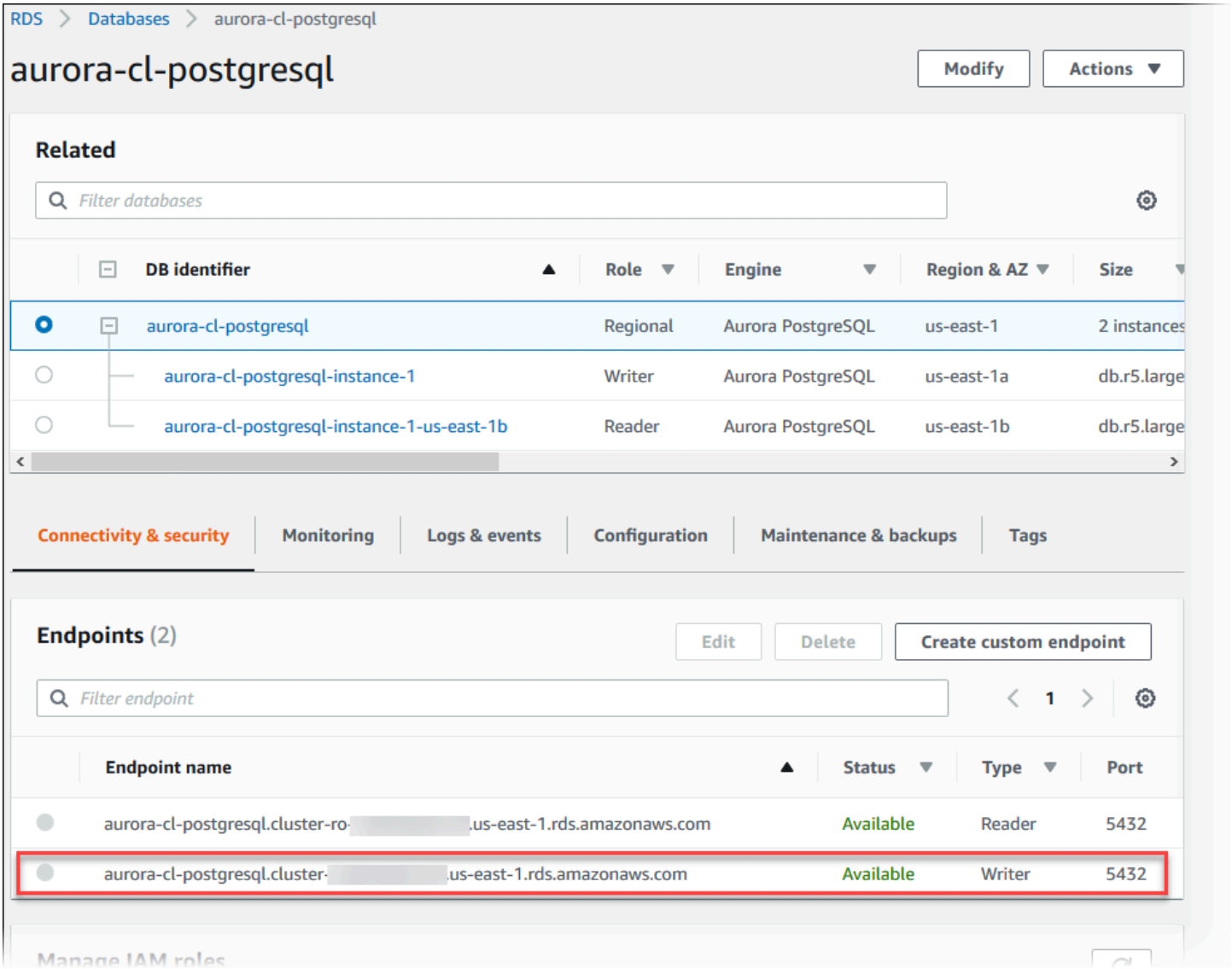
Saat Anda memiliki koneksi ke instans DB di klaster DB Amazon Aurora PostgreSQL, Anda dapat menjalankan perintah SQL apa pun yang kompatibel dengan PostgreSQL.

Dalam tampilan detail untuk klaster DB Aurora PostgreSQL, Anda dapat menemukan nama, status, jenis, dan nomor port titik akhir klaster. Anda menggunakan titik akhir dan nomor port di string koneksi PostgreSQL Anda. Misalnya, jika nilai titik akhir adalah `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`, maka Anda menentukan nilai berikut dalam string koneksi PostgreSQL:

- Untuk host atau nama host, tentukan `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com`
- Untuk port, tentukan `5432` atau nilai port yang Anda gunakan saat Anda membuat klaster DB

Titik akhir klaster menghubungkan Anda ke instans primer untuk klaster DB. Anda dapat melakukan operasi baca dan tulis menggunakan titik akhir klaster. Klaster DB Anda juga dapat memiliki hingga 15 Replika Aurora yang mendukung akses hanya baca ke data di klaster DB Anda. Setiap instans DB dalam klaster Aurora (yaitu, instans primer dan setiap Replika Aurora) memiliki titik akhir unik yang independen dari titik akhir klaster. Titik akhir unik ini memungkinkan Anda terhubung ke instans DB tertentu dalam klaster secara langsung. Titik akhir klaster selalu mengarah ke instans primer. Jika instans primer gagal dan diganti, titik akhir klaster mengarah ke instans primer baru.

Untuk melihat titik akhir klaster (titik akhir penulis), pilih Basis data di konsol Amazon RDS dan pilih nama klaster DB untuk menampilkan detail klaster DB.



Utilitas koneksi untuk Aurora PostgreSQL

Beberapa utilitas koneksi yang dapat Anda gunakan adalah sebagai berikut:

- Baris perintah – Anda dapat terhubung ke klaster DB Aurora PostgreSQL dengan menggunakan alat seperti psql, terminal interaktif PostgreSQL. Untuk informasi selengkapnya tentang menggunakan terminal interaktif PostgreSQL, lihat psql dalam dokumentasi PostgreSQL.
- GUI – Anda dapat menggunakan utilitas pgAdmin untuk terhubung ke klaster DB Aurora PostgreSQL dengan menggunakan antarmuka UI. Untuk informasi selengkapnya, lihat halaman Unduhan dari situs web pgAdmin.

- Aplikasi – Anda dapat menggunakan Driver AWS JDBC for PostgreSQL untuk menghubungkan aplikasi Anda ke klaster DB Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Menghubungkan dengan Driver AWS JDBC untuk PostgreSQL.

Menghubungkan dengan Driver AWS JDBC untuk PostgreSQL

Driver AWS JDBC untuk PostgreSQL adalah pembungkus klien yang dirancang untuk membantu Anda memanfaatkan sepenuhnya fitur ketersediaan tinggi Aurora PostgreSQL. Untuk informasi selengkapnya tentang Driver AWS JDBC untuk PostgreSQL dan petunjuk lengkap untuk menggunakannya, lihat Driver JDBC untuk repositori PostgreSQL.AWS GitHub

Driver AWS JDBC untuk PostgreSQL memperluas driver PGJDBC komunitas. Driver ini telah dirancang untuk memanfaatkan sepenuhnya kemampuan failover klaster Aurora PostgreSQL. Jika terjadi failover, AWS JDBC Driver for PostgreSQL akan mengueri klaster secara langsung untuk topologi baru alih-alih menggunakan resolusi DNS. Dengan menanyakan cluster secara langsung, Driver AWS JDBC untuk PostgreSQL dapat terhubung ke primer baru lebih cepat dengan cara yang andal dan dapat diprediksi, menghindari potensi penundaan yang disebabkan oleh resolusi DNS.

Driver AWS JDBC untuk PostgreSQL mendukung otentikasi database (IAM) dan. AWS Identity and Access Management AWS Secrets Manager Untuk informasi selengkapnya tentang penggunaan mekanisme otentikasi ini dengan driver, lihat Plugin dan AWS Secrets Manager Plugin Otentikasi AWS IAM di Driver AWS JDBC untuk repositori PostgreSQL. GitHub

Untuk mengetahui informasi selengkapnya tentang autentikasi basis data IAM, lihat Autentikasi basis data IAM. Untuk informasi selengkapnya tentang Secrets Manager, lihat Panduan Pengguna AWS Secrets Manager.

Pemecahan masalah kegagalan koneksi Aurora

Penyebab umum kegagalan koneksi ke klaster DB Aurora baru adalah sebagai berikut:

- Grup keamanan di VPC tidak mengizinkan akses – VPC Anda perlu mengizinkan koneksi dari perangkat Anda atau dari instans Amazon EC2 dengan konfigurasi grup keamanan yang tepat di VPC. Untuk mengatasinya, modifikasi aturan masuk grup keamanan VPC Anda untuk mengizinkan koneksi. Sebagai contoh, lihat Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4).
- Port diblokir oleh aturan firewall – Periksa nilai port yang dikonfigurasi untuk klaster DB Aurora Anda. Jika aturan firewall memblokir port tersebut, Anda dapat membuat ulang instans menggunakan port yang berbeda.

- Konfigurasi IAM tidak lengkap atau salah – Jika Anda membuat instans DB Aurora Anda untuk menggunakan autentikasi berbasis IAM, pastikan bahwa autentikasi tersebut dikonfigurasi dengan benar. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.

Untuk informasi selengkapnya tentang pemecahan masalah koneksi Aurora DB, lihat Tidak dapat terhubung ke instans DB Amazon RDS.

Bekerja dengan grup parameter

Parameter basis data menentukan konfigurasi basis data. Misalnya, parameter basis data dapat menentukan jumlah sumber daya, seperti memori, untuk dialokasikan ke basis data.

Anda mengelola konfigurasi basis data dengan mengaitkan instans DB dan klaster DB Aurora dengan grup parameter. Aurora menentukan grup parameter dengan pengaturan default. Anda dapat juga menentukan grup parameter Anda sendiri dengan pengaturan yang disesuaikan.

Topik

- Ikhtisar grup parameter
- Bekerja dengan grup parameter klaster DB
- Bekerja dengan grup parameter DB dalam instance DB
- Membandingkan grup parameter DB
- Menentukan parameter DB

Ikhtisar grup parameter

Grup parameter klaster DB bertindak sebagai kontainer untuk nilai konfigurasi mesin yang diterapkan pada setiap instans DB di klaster DB Aurora. Misalnya, model penyimpanan bersama Aurora mengharuskan setiap instans DB dalam klaster Aurora menggunakan pengaturan yang sama untuk parameter seperti `innodb_file_per_table`. Dengan demikian, parameter yang memengaruhi tata letak penyimpanan fisik adalah bagian dari grup parameter klaster. Grup parameter klaster DB juga mencakup nilai default untuk semua parameter tingkat instans.

Grup parameter DB bertindak sebagai kontainer untuk nilai konfigurasi mesin yang diterapkan pada satu atau beberapa instans DB. Grup parameter DB berlaku untuk instans DB di Amazon RDS maupun Aurora. Pengaturan konfigurasi ini berlaku untuk properti yang dapat beragam di antara instans DB dalam klaster Aurora, misalnya ukuran untuk buffer memori.

Topik

- Grup parameter kustom dan default
- Parameter klaster DB statis dan dinamis
- Parameter instans DB statis dan dinamis

- Parameter set karakter
- Nilai parameter dan parameter yang didukung

Grup parameter kustom dan default

Jika Anda membuat instans DB tanpa menentukan grup parameter DB, instans DB akan menggunakan grup parameter DB default. Demikian pula, jika Anda membuat klaster DB Aurora tanpa menentukan grup parameter klaster DB, klaster DB tersebut akan menggunakan grup parameter klaster DB default. Setiap grup parameter default berisi default mesin basis data dan default sistem Amazon RDS berdasarkan mesin, kelas komputasi, dan penyimpanan yang dialokasikan dari instans.

Anda tidak dapat memodifikasi pengaturan parameter dari grup parameter default. Anda dapat melakukan hal berikut sebagai gantinya:

1. Buat grup parameter baru.
2. Ubah pengaturan parameter yang Anda inginkan. Tidak semua parameter mesin DB dalam grup parameter memenuhi syarat untuk dimodifikasi.
3. Ubah instans DB atau cluster DB Anda untuk mengaitkan grup parameter baru.

   Untuk mengetahui informasi tentang cara memodifikasi klaster atau instans DB, lihat Memodifikasi klaster DB Amazon Aurora.

   > **Note**
   >
   > Jika Anda telah memodifikasi instans DB Anda untuk menggunakan grup parameter kustom, dan Anda memulai instans DB, RDS secara otomatis me-reboot instans DB sebagai bagian dari proses startup.

RDS menerapkan parameter statis dan dinamis yang dimodifikasi dalam grup parameter yang baru terkait hanya setelah instance DB di-boot ulang. Namun, jika Anda memodifikasi parameter dinamis dalam grup parameter DB setelah Anda mengaitkannya dengan instans DB, perubahan ini akan diterapkan segera tanpa reboot. Untuk informasi selengkapnya tentang cara mengubah grup parameter DB, lihat Memodifikasi klaster DB Amazon Aurora.

Jika Anda memperbarui parameter dalam grup parameter DB, perubahan berlaku untuk semua instans DB yang terkait dengan grup parameter tersebut. Demikian pula, jika Anda memperbarui

parameter dalam grup parameter klaster DB Aurora, perubahan berlaku untuk semua klaster DB Aurora yang terkait dengan grup parameter klaster DB tersebut.

Jika Anda tidak ingin membuat grup parameter dari awal, Anda dapat menyalin grup parameter yang ada dengan AWS CLI copy-db-parameter-groupperintah atau perintah copy-db-cluster-parameter-group. Anda mungkin mendapati bahwa menyalin grup parameter berguna dalam beberapa kasus. Misalnya, Anda mungkin ingin menyertakan sebagian besar parameter dan nilai kustom grup parameter yang ada dalam grup parameter baru.

Parameter klaster DB statis dan dinamis

Parameter klaster DB bersifat statis atau dinamis. Perbedaannya terletak pada hal berikut:

- Saat Anda mengubah parameter statis dan menyimpan grup parameter klaster DB, perubahan parameter akan berlaku setelah Anda me-reboot instans DB secara manual di setiap klaster DB terkait. Bila Anda menggunakan AWS Management Console untuk mengubah nilai parameter cluster DB statis, selalu digunakan `pending-reboot` untuk`ApplyMethod`.
- Saat Anda mengubah parameter dinamis, perubahan parameter akan langsung diterapkan secara default, tanpa harus di-reboot. Saat Anda menggunakan konsol, konsol tersebut selalu menggunakan `immediate` untuk `ApplyMethod`. Untuk menunda perubahan parameter hingga setelah Anda me-reboot instans DB di cluster DB terkait, gunakan AWS CLI atau RDS API. Atur `ApplyMethod` ke `pending-reboot` untuk perubahan parameter.

Untuk informasi selengkapnya tentang menggunakan AWS CLI untuk mengubah nilai parameter, lihat modify-db-cluster-parameter-group. Untuk informasi selengkapnya tentang penggunaan RDS API untuk mengubah nilai parameter, lihat ClusterParameterGroupModifyDB.

Jika Anda mengubah grup parameter klaster DB yang terkait dengan klaster DB, reboot instans DB di klaster DB tersebut. Dengan me-reboot, perubahan akan diterapkan ke semua instans DB di klaster DB. Untuk menentukan apakah instans DB dari klaster DB harus di-reboot untuk menerapkan perubahan, jalankan perintah AWS CLI berikut.

```
aws rds describe-db-clusters --db-cluster-identifier db_cluster_identifier
```

Periksa nilai `DBClusterParameterGroupStatus` untuk instans DB primer dalam output. Jika nilainya adalah `pending-reboot`, reboot instans DB dari klaster DB.

Parameter instans DB statis dan dinamis

Parameter instans DB bersifat statis atau dinamis. Perbedaannya terletak dalam hal berikut:

- Saat Anda mengubah parameter statis dan menyimpan grup parameter DB, perubahan parameter akan diterapkan setelah Anda me-reboot instans DB terkait secara manual. Untuk parameter statis, konsol selalu menggunakan `pending-reboot` untuk `ApplyMethod`.
- Saat Anda mengubah parameter dinamis, perubahan parameter akan langsung diterapkan secara default, tanpa harus di-reboot. Saat Anda menggunakan AWS Management Console untuk mengubah nilai parameter instans DB, selalu digunakan `immediate` untuk parameter dinamis `ApplyMethod` for. Untuk menunda perubahan parameter hingga setelah Anda me-reboot instans DB terkait, gunakan AWS CLI atau RDS API. Atur `ApplyMethod` ke `pending-reboot` untuk perubahan parameter.

Untuk informasi selengkapnya tentang menggunakan AWS CLI untuk mengubah nilai parameter, lihat modify-db-parameter-group. Untuk informasi selengkapnya tentang penggunaan RDS API untuk mengubah nilai parameter, lihat ParameterGroupModifyDB.

Jika instans DB tidak menggunakan perubahan terbaru pada grup parameter DB terkait, konsol menunjukkan status reboot-tertunda untuk grup parameter DB. Status ini tidak menyebabkan reboot otomatis selama periode pemeliharaan berikutnya. Untuk menerapkan perubahan parameter terbaru pada instans DB tersebut, reboot instans DB secara manual.

Parameter set karakter

Sebelum Anda membuat klaster DB, atur parameter apa pun yang terkait dengan set karakter atau kolasi basis data di grup parameter Anda. Lakukan juga sebelum Anda membuat basis data di dalamnya. Dengan cara ini, Anda memastikan basis data default dan basis data baru menggunakan nilai kolasi dan set karakter yang Anda tentukan. Jika Anda mengubah parameter kolasi atau set karakter, perubahan parameter tidak diterapkan ke basis data yang sudah ada.

Untuk beberapa mesin DB, Anda dapat mengubah nilai kolasi atau set karakter untuk basis data yang sudah ada menggunakan perintah `ALTER DATABASE`, contohnya:

```
ALTER DATABASE database_name CHARACTER SET character_set_name COLLATE collation;
```

Untuk informasi selengkapnya tentang cara mengubah nilai set kolasi atau set karakter untuk basis data, periksa dokumentasi untuk mesin DB Anda.

Nilai parameter dan parameter yang didukung

Untuk menentukan parameter yang didukung untuk mesin DB, lihat parameter di grup parameter DB dan grup parameter klaster DB yang digunakan oleh instans DB atau klaster DB. Untuk informasi

selengkapnya, lihat Melihat nilai parameter untuk grup parameter DB dan Melihat nilai parameter untuk grup parameter klaster DB.

Anda juga dapat menentukan parameter bilangan bulat dan Boolean menggunakan ekspresi, rumus, dan fungsi. Fungsi dapat mencakup ekspresi log matematis. Namun, tidak semua parameter mendukung ekspresi, rumus, dan fungsi untuk nilai parameter. Untuk informasi selengkapnya, lihat Menentukan parameter DB.

Untuk basis data global Aurora, Anda dapat menentukan pengaturan konfigurasi yang berbeda untuk setiap klaster Aurora. Pastikan bahwa pengaturannya cukup serupa untuk menghasilkan perilaku yang konsisten jika Anda mempromosikan klaster sekunder untuk menjadi klaster primer. Misalnya, gunakan pengaturan yang sama untuk zona waktu dan set karakter di semua klaster basis data global Aurora.

Pengaturan parameter yang tidak tepat dalam grup parameter dapat menimbulkan dampak buruk yang tidak diinginkan, termasuk penurunan performa dan ketidakstabilan sistem. Selalu berhati-hatilah saat memodifikasi parameter basis data, dan cadangkan data Anda sebelum memodifikasi grup parameter. Cobalah menerapkan perubahan pengaturan grup parameter pada instans DB atau klaster DB uji coba sebelum menerapkan perubahan grup parameter tersebut ke instans DB atau klaster DB produksi.

Bekerja dengan grup parameter klaster DB

Klaster DB Amazon Aurora menggunakan grup parameter klaster DB. Bagian berikut menjelaskan cara mengonfigurasi dan mengelola grup parameter klaster DB.

Topik

- Parameter instans DB dan klaster DB Amazon Aurora
- Membuat grup parameter klaster DB
- Mengaitkan grup parameter klaster DB dengan klaster DB
- Mengubah parameter dalam grup parameter klaster DB
- Mengatur ulang parameter dalam grup parameter klaster DB
- Menyalin grup parameter klaster DB
- Mencantumkan grup parameter klaster DB
- Melihat nilai parameter untuk grup parameter klaster DB
- Menghapus grup parameter cluster DB

Parameter instans DB dan klaster DB Amazon Aurora

Aurora menggunakan sistem pengaturan konfigurasi dua tingkat:

- Parameter dalam Grup parameter klaster DB berlaku untuk setiap instans DB dalam klaster DB. Data Anda disimpan dalam subsistem penyimpanan bersama Aurora. Oleh karena itu, semua parameter yang terkait dengan tata letak fisik data tabel harus sama untuk semua instans DB dalam klaster Aurora. Demikian juga, karena instans Aurora DB dihubungkan dengan replikasi, semua parameter untuk pengaturan replikasi harus identik di seluruh klaster Aurora.
- Parameter dalam Grup parameter DB berlaku untuk satu instans DB dalam klaster DB Aurora. Parameter ini terkait dengan sejumlah aspek, seperti penggunaan memori yang dapat divariasikan di seluruh instans DB dalam klaster Aurora yang sama. Misalnya, klaster sering kali berisi instans DB dengan kelas instans AWS yang berbeda.

Setiap klaster Aurora dikaitkan dengan grup parameter klaster DB. Grup parameter ini menetapkan nilai default untuk setiap nilai konfigurasi untuk mesin DB yang sesuai. Grup parameter klaster mencakup default untuk parameter tingkat klaster dan tingkat instans. Setiap instans DB dalam klaster yang disediakan atau klaster Aurora Serverless v2 mewarisi pengaturan dari grup parameter klaster DB tersebut.

Setiap instans DB juga terkait dengan grup parameter DB. Nilai dalam grup parameter DB dapat menggantikan nilai default dari grup parameter klaster. Misalnya, jika satu instans dalam klaster mengalami masalah, Anda dapat menetapkan grup parameter DB kustom ke instans tersebut. Grup parameter kustom mungkin memiliki pengaturan khusus untuk parameter yang terkait dengan proses debug atau penyesuaian performa.

Aurora menetapkan grup parameter default saat Anda membuat klaster atau instans DB baru, berdasarkan mesin dan versi basis data yang ditentukan. Sebagai gantinya, Anda dapat menentukan grup parameter kustom. Anda membuat sendiri grup parameter tersebut, dan Anda dapat mengedit nilai parameter. Anda dapat menentukan grup parameter kustom ini saat pembuatan. Anda juga dapat mengubah klaster atau instans DB nantinya untuk menggunakan grup parameter kustom.

Untuk instans yang disediakan dan instans Aurora Serverless v2, nilai konfigurasi apa pun yang dimodifikasi di grup parameter klaster DB akan mengganti nilai default dalam grup parameter DB. Jika Anda mengedit nilai yang sesuai dalam grup parameter DB, nilai tersebut akan mengganti pengaturan dari grup parameter klaster DB.

Pengaturan parameter DB apa pun yang Anda modifikasi akan diprioritaskan daripada nilai grup parameter klaster DB, bahkan jika Anda mengubah parameter konfigurasi kembali ke nilai defaultnya. Anda dapat melihat parameter mana yang diganti dengan menggunakan describe-db-parametersAWS CLI perintah atau operasi API DescribedBParameters RDS. Bidang `Source` berisi nilai `user` jika Anda telah memodifikasi parameter tersebut. Untuk mengatur ulang satu atau beberapa parameter sehingga nilai dari grup parameter cluster DB diutamakan, gunakan reset-db-parameter-group AWS CLI perintah atau operasi ResetDB ParameterGroup RDS API.

Parameter instans DB dan klaster DB yang tersedia untuk Anda di Aurora bervariasi, bergantung pada kompatibilitas mesin basis data.

Mesin basis data	Parameter
Aurora MySQL	Lihat Parameter konfigurasi Aurora MySQL. Untuk klaster Aurora Serverless, lihat detail tambahan di Menggunakan grup parameter untuk Aurora Serverless v2 dan Grup parameter untuk Aurora Serverless v1.
Aurora PostgreSQL	Lihat Parameter Amazon Aurora PostgreSQL. Untuk klaster Aurora Serverless, lihat detail tambahan di Menggunakan grup parameter untuk Aurora Serverless v2 dan Grup parameter untuk Aurora Serverless v1.

> **Note**
>
> Klaster Aurora Serverless v1 hanya memiliki grup parameter klaster DB, bukan grup parameter DB. Untuk klaster Aurora Serverless v2, Anda membuat semua perubahan pada parameter kustom dalam grup parameter klaster DB.
> Aurora Serverless v2 menggunakan grup parameter klaster DB dan grup parameter DB. Dengan Aurora Serverless v2, Anda dapat memodifikasi hampir semua parameter konfigurasi. Aurora Serverless v2 mengganti pengaturan beberapa parameter konfigurasi terkait kapasitas, sehingga beban kerja Anda tidak terganggu saat instans Aurora Serverless v2 diturunkan skalanya.

Untuk mempelajari lebih lanjut tentang pengaturan konfigurasi Aurora Serverless dan pengaturan mana yang dapat dimodifikasi, lihat Menggunakan grup parameter untuk Aurora Serverless v2 dan Grup parameter untuk Aurora Serverless v1.

Membuat grup parameter klaster DB

Anda dapat membuat grup parameter cluster DB baru menggunakan AWS Management Console, AWS CLI, atau RDS API.

Setelah Anda membuat grup parameter klaster DB, tunggu minimal 5 menit sebelum membuat klaster DB yang menggunakan grup parameter klaster DB tersebut. Dengan demikian, Amazon RDS dapat sepenuhnya membuat grup parameter sebelum digunakan oleh klaster DB baru. Anda dapat menggunakan halaman grup Parameter di konsol Amazon RDS atau describe-db-cluster-parametersperintah untuk memverifikasi bahwa grup parameter cluster DB Anda dibuat.

Batasan berikut berlaku untuk nama grup parameter klaster DB:

- Nama harus berisi 1 sampai 255 huruf, angka, atau tanda hubung.

  Nama grup parameter default boleh menyertakan titik, seperti `default.aurora-mysql5.7`. Namun, nama grup parameter kustom tidak boleh menyertakan titik.
- Karakter pertamanya harus berupa huruf.
- Nama tidak boleh diakhiri dengan tanda hubung atau berisi dua tanda hubung berturut-turut.

Konsol

Untuk membuat grup parameter klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Pilih Buat grup parameter.

   Jendela Buat grup parameter akan muncul.
4. Dalam daftar Kelompok grup parameter, pilih kelompok grup parameter DB.
5. Dalam daftar Type, pilih grup parameter cluster DB.

6. Di kotak Nama grup, masukkan nama grup parameter klaster DB yang baru.
7. Di kotak Deskripsi, masukkan deskripsi untuk grup parameter klaster DB yang baru.
8. Pilih Buat.

AWS CLI

Untuk membuat grup parameter cluster DB, gunakan AWS CLI `create-db-cluster-parameter-group`perintah.

Contoh berikut membuat grup parameter klaster DB yang bernama mydbclusterparametergroup untuk Aurora MySQL versi 5.7 dengan deskripsi "Grup parameter klaster baru saya".

Sertakan parameter wajib berikut:

- `--db-cluster-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

Untuk mencantumkan semua kelompok grup parameter yang tersedia, gunakan perintah berikut:

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> Output berisi duplikat.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
    --description "My new cluster parameter group"
```

Untuk Windows:

```
aws rds create-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new cluster parameter group"
```

Perintah ini menghasilkan output yang serupa dengan yang berikut:

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "mydbclusterparametergroup",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "My new cluster parameter group",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
    }
}
```

API RDS

Untuk membuat grup parameter klaster DB, gunakan tindakan `CreateDBClusterParameterGroup` API RDS.

Sertakan parameter wajib berikut:

- `DBClusterParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

Mengaitkan grup parameter klaster DB dengan klaster DB

Anda dapat membuat sendiri grup parameter klaster DB dengan pengaturan yang disesuaikan. Anda dapat mengaitkan grup parameter cluster DB dengan cluster DB menggunakan AWS Management Console, the AWS CLI, atau RDS API. Anda dapat melakukannya saat membuat atau memodifikasi klaster DB.

Untuk informasi tentang cara membuat grup parameter klaster DB, lihat Membuat grup parameter klaster DB. Untuk informasi tentang cara membuat klaster DB, lihat Membuat klaster DB Amazon Aurora. Untuk informasi tentang cara memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

> **Note**
>
> Untuk Aurora PostgreSQL 15.2, 14.7, 13.10, 12.14, dan semua 11 versi, ketika Anda mengubah grup parameter cluster DB yang terkait dengan cluster DB, reboot setiap instance replika untuk menerapkan perubahan.
> Untuk menentukan apakah instance DB utama dari cluster DB harus di-boot ulang untuk menerapkan perubahan, jalankan perintah berikut: AWS CLI
> `aws rds describe-db-clusters --db-cluster-identifier` *`db_cluster_identifier`*
> Periksa nilai `DBClusterParameterGroupStatus` untuk instans DB primer dalam output. Jika nilainya adalah `pending-reboot`, reboot instans DB primer dari klaster DB.

Konsol

Untuk mengaitkan grup parameter klaster DB dengan klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, lalu pilih klaster DB yang ingin dimodifikasi.
3. Pilih Modifikasi. Halaman Modifikasi klaster DB akan muncul.
4. Ubah pengaturan Grup parameter klaster DB.
5. Pilih Lanjutkan dan periksa ringkasan modifikasi.

   Perubahan akan langsung diterapkan, terlepas dari pengaturan Penjadwalan modifikasi.
6. Di halaman konfirmasi, tinjau perubahan Anda. Jika sudah benar, pilih Modifikasi klaster untuk menyimpan perubahan Anda.

   Atau, pilih Kembali untuk mengedit perubahan, atau pilih Batal untuk membatalkan perubahan.

AWS CLI

Untuk mengaitkan grup parameter cluster DB dengan cluster DB, gunakan AWS CLI `modify-db-cluster`perintah dengan opsi berikut:

- `--db-cluster-name`
- `--db-cluster-parameter-group-name`

Contoh berikut mengaitkan grup parameter DB `mydbclpg` dengan klaster DB `mydbcluster`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --db-cluster-parameter-group-name mydbclpg
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-parameter-group-name mydbclpg
```

API RDS

Untuk mengaitkan grup parameter klaster DB dengan klaster DB, gunakan operasi `ModifyDBCluster` API RDS dengan parameter berikut:

- `DBClusterIdentifier`
- `DBClusterParameterGroupName`

Mengubah parameter dalam grup parameter klaster DB

Anda dapat mengubah nilai parameter dalam grup parameter klaster DB buatan pelanggan. Anda tidak dapat mengubah nilai parameter dalam grup parameter klaster DB default. Perubahan pada parameter dalam grup parameter klaster DB buatan pelanggan diterapkan ke semua klaster DB yang dikaitkan dengan grup parameter klaster DB.

Konsol

Untuk mengubah grup parameter klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter yang ingin Anda modifikasi.
4. Untuk Tindakan grup parameter, pilih Edit.

5. Ubah nilai parameter yang ingin Anda modifikasi. Anda dapat menggulir parameter menggunakan tombol panah di bagian kanan atas kotak dialog.

   Anda tidak dapat mengubah nilai dalam grup parameter default.
6. Pilih Simpan perubahan.
7. Reboot instance DB utama (penulis) di cluster untuk menerapkan perubahan padanya.
8. Kemudian reboot instans DB pembaca untuk menerapkan perubahan pada mereka.

AWS CLI

Untuk memodifikasi grup parameter cluster DB, gunakan AWS CLI `modify-db-cluster-parameter-group`perintah dengan parameter yang diperlukan berikut:

- `--db-cluster-parameter-group-name`
- `--parameters`

Contoh berikut memodifikasi nilai `server_audit_logging` dan `server_audit_logs_upload` dalam grup parameter klaster DB yang bernama mydbclusterparametergroup.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" \

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

Perintah menghasilkan output seperti berikut:

```
DBCLUSTERPARAMETERGROUP  mydbclusterparametergroup
```

API RDS

Untuk memodifikasi grup parameter klaster DB, gunakan perintah `ModifyDBClusterParameterGroup` API RDS dengan parameter wajib berikut:

- `DBClusterParameterGroupName`
- `Parameters`

Mengatur ulang parameter dalam grup parameter klaster DB

Anda dapat mengatur ulang parameter ke nilai defaultnya dalam grup parameter klaster DB buatan pelanggan. Perubahan pada parameter dalam grup parameter klaster DB buatan pelanggan diterapkan ke semua klaster DB yang dikaitkan dengan grup parameter klaster DB.

> **Note**
>
> Dalam grup parameter klaster DB default, parameter selalu ditetapkan ke nilai defaultnya.

Konsol

Untuk mengatur ulang parameter dalam grup parameter klaster DB ke nilai defaultnya

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter.
4. Untuk Tindakan grup parameter, pilih Edit.
5. Pilih parameter yang ingin Anda atur ulang ke nilai defaultnya. Anda dapat menggulir parameter menggunakan tombol panah di bagian kanan atas kotak dialog.

   Anda tidak dapat mengatur ulang nilai dalam grup parameter default.
6. Pilih Atur ulang lalu konfirmasikan dengan memilih Atur ulang parameter.

7. Reboot instans DB primer dalam klaster DB untuk menerapkan perubahan pada semua instans DB dalam klaster DB.

AWS CLI

Untuk mengatur ulang parameter dalam grup parameter cluster DB ke nilai defaultnya, gunakan AWS CLI `reset-db-cluster-parameter-group`perintah dengan opsi wajib berikut:`--db-cluster-parameter-group-name`.

Untuk mengatur ulang semua parameter dalam grup parameter klaster DB, tentukan opsi `--reset-all-parameters`. Untuk mengatur ulang parameter tertentu, tentukan opsi `--parameters`.

Contoh berikut mengatur ulang semua parameter dalam grup parameter DB yang bernama mydbparametergroup ke nilai defaultnya.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

Untuk Windows:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbparametergroup ^
    --reset-all-parameters
```

Contoh berikut mengatur ulang `server_audit_logging` dan `server_audit_logs_upload` ke nilai defaultnya dalam grup parameter klaster DB yang bernama mybdclusterparametergroup.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name mydbclusterparametergroup \
    --parameters "ParameterName=server_audit_logging,ApplyMethod=immediate" \
```

```
                "ParameterName=server_audit_logs_upload,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name mydbclusterparametergroup ^
    --parameters
 "ParameterName=server_audit_logging,ParameterValue=1,ApplyMethod=immediate" ^

 "ParameterName=server_audit_logs_upload,ParameterValue=1,ApplyMethod=immediate"
```

Perintah menghasilkan output seperti berikut:

```
DBClusterParameterGroupName  mydbclusterparametergroup
```

API RDS

Untuk mengatur ulang parameter dalam grup parameter klaster DB ke nilai default, gunakan perintah `ResetDBClusterParameterGroup` API RDS dengan parameter wajib berikut: `DBClusterParameterGroupName`.

Untuk mengatur ulang semua parameter dalam grup parameter klaster DB, atur parameter `ResetAllParameters` ke `true`. Untuk mengatur ulang parameter tertentu, tentukan parameter `Parameters`.

Menyalin grup parameter klaster DB

Anda dapat menyalin grup parameter klaster DB kustom yang Anda buat. Menyalin grup parameter adalah solusi yang mudah jika Anda sudah membuat grup parameter klaster DB dan Anda ingin menyertakan sebagian besar parameter dan nilai kustom dari grup tersebut dalam grup parameter klaster DB yang baru. Anda dapat menyalin grup parameter cluster DB dengan menggunakan perintah AWS CLI copy-db-cluster-parameter-group atau operasi ClusterParameterGroupCopyDB API RDS.

Setelah Anda membuat grup parameter klaster DB, tunggu minimal 5 menit sebelum membuat klaster DB yang menggunakan grup parameter klaster DB tersebut. Dengan demikian, Amazon RDS dapat sepenuhnya menyalin grup parameter sebelum digunakan oleh klaster DB baru. Anda dapat menggunakan halaman grup Parameter di konsol Amazon RDS atau describe-db-cluster-parametersperintah untuk memverifikasi bahwa grup parameter cluster DB Anda dibuat.

> **Note**
>
> Anda tidak dapat menyalin grup parameter default. Namun, Anda dapat membuat grup parameter baru yang didasarkan pada grup parameter default.
> Anda tidak dapat menyalin grup parameter cluster DB ke grup yang berbeda Akun AWS atau Wilayah AWS.

Konsol

Untuk menyalin grup parameter klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter kustom yang ingin Anda salin.
4. Untuk Tindakan grup parameter, pilih Salin.
5. Di Pengidentifikasi grup parameter DB baru, masukkan nama untuk grup parameter baru.
6. Di Deskripsi, masukkan deskripsi untuk grup parameter DB yang baru.
7. Pilih Salin.

AWS CLI

Untuk menyalin grup parameter cluster DB, gunakan AWS CLI `copy-db-cluster-parameter-group`perintah dengan parameter yang diperlukan berikut:

- `--source-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-identifier`
- `--target-db-cluster-parameter-group-description`

Contoh berikut membuat grup parameter klaster DB baru yang bernama `mygroup2` yang merupakan salinan dari grup parameter klaster DB `mygroup1`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds copy-db-cluster-parameter-group \
    --source-db-cluster-parameter-group-identifier mygroup1 \
    --target-db-cluster-parameter-group-identifier mygroup2 \
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

Untuk Windows:

```
aws rds copy-db-cluster-parameter-group ^
    --source-db-cluster-parameter-group-identifier mygroup1 ^
    --target-db-cluster-parameter-group-identifier mygroup2 ^
    --target-db-cluster-parameter-group-description "DB parameter group 2"
```

API RDS

Untuk menyalin grup parameter klaster DB, gunakan operasi `CopyDBClusterParameterGroup` API RDS dengan parameter wajib berikut:

- `SourceDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupIdentifier`
- `TargetDBClusterParameterGroupDescription`

Mencantumkan grup parameter klaster DB

Anda dapat mencantumkan grup parameter cluster DB yang telah Anda buat untuk AWS akun Anda.

> **Note**
>
> Grup parameter default secara otomatis dibuat dari template parameter default ketika Anda membuat klaster DB untuk mesin dan versi DB tertentu. Grup parameter default ini berisi pengaturan parameter pilihan dan tidak dapat dimodifikasi. Saat membuat grup parameter kustom, Anda dapat memodifikasi pengaturan parameter.

Konsol

Untuk mencantumkan semua grup parameter cluster DB untuk AWS akun

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

2. Di panel navigasi, pilih Grup parameter.

   Grup parameter klaster DB muncul dalam daftar dengan Grup parameter klaster DB untuk Jenis.

AWS CLI

Untuk mencantumkan semua grup parameter cluster DB untuk AWS akun, gunakan AWS CLI `describe-db-cluster-parameter-groups`perintah.

Example

Contoh berikut mencantumkan semua grup parameter klaster DB yang tersedia untuk akun AWS .

```
aws rds describe-db-cluster-parameter-groups
```

Contoh berikut menjelaskan grup parameter mydbclusterparametergroup.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-cluster-parameter-groups \
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

Untuk Windows:

```
aws rds describe-db-cluster-parameter-groups ^
    --db-cluster-parameter-group-name mydbclusterparametergroup
```

Perintah memberikan respons seperti berikut:

```
{
    "DBClusterParameterGroups": [
        {
            "DBClusterParameterGroupName": "mydbclusterparametergroup",
            "DBParameterGroupFamily": "aurora-mysql5.7",
            "Description": "My new cluster parameter group",
            "DBClusterParameterGroupArn": "arn:aws:rds:us-east-1:123456789012:cluster-
pg:mydbclusterparametergroup"
        }
    ]
}
```

API RDS

Untuk mencantumkan semua grup parameter cluster DB untuk AWS akun, gunakan `DescribeDBClusterParameterGroups`tindakan RDS API.

Melihat nilai parameter untuk grup parameter klaster DB

Anda dapat memperoleh daftar semua parameter dalam grup parameter klaster DB beserta nilainya.

Konsol

Untuk melihat nilai parameter untuk grup parameter klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.

   Grup parameter klaster DB muncul dalam daftar dengan Grup parameter klaster DB untuk Jenis.
3. Pilih nama grup parameter klaster DB untuk melihat daftar parameternya.

AWS CLI

Untuk melihat nilai parameter untuk grup parameter cluster DB, gunakan AWS CLI `describe-db-cluster-parameters`perintah dengan parameter yang diperlukan berikut.

- `--db-cluster-parameter-group-name`

Example

Contoh berikut mencantumkan parameter dan nilai parameter untuk grup parameter klaster DB yang bernama mydbclusterparametergroup, dalam format JSON.

Perintah memberikan respons seperti berikut:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-
name mydbclusterparametergroup
```

```
{
    "Parameters": [
```

```
        {
            "ParameterName": "allow-suspicious-udfs",
            "Description": "Controls whether user-defined functions that have only an
 xxx symbol for the main function can be loaded",
            "Source": "engine-default",
            "ApplyType": "static",
            "DataType": "boolean",
            "AllowedValues": "0,1",
            "IsModifiable": false,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },
        {
            "ParameterName": "aurora_binlog_read_buffer_size",
            "ParameterValue": "5242880",
            "Description": "Read buffer size used by master dump thread when the switch
 aurora_binlog_use_large_read_buffer is ON.",
            "Source": "engine-default",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "8192-536870912",
            "IsModifiable": true,
            "ApplyMethod": "pending-reboot",
            "SupportedEngineModes": [
                "provisioned"
            ]
        },

...
```

API RDS

Untuk melihat nilai parameter untuk grup parameter klaster DB, gunakan perintah `DescribeDBClusterParameters` API RDS dengan parameter wajib berikut.

- `DBClusterParameterGroupName`

Dalam beberapa kasus, nilai yang diizinkan untuk parameter tidak ditampilkan. Nilai ini selalu merupakan parameter yang sumbernya adalah mesin basis data default.

Untuk melihat nilai parameter ini, Anda dapat menjalankan pernyataan SQL berikut:

- MySQL:

```
-- Show the value of a particular parameter
mysql$ SHOW VARIABLES LIKE '%parameter_name%';

-- Show the values of all parameters
mysql$ SHOW VARIABLES;
```

- PostgreSQL:

```
-- Show the value of a particular parameter
postgresql=> SHOW parameter_name;

-- Show the values of all parameters
postgresql=> SHOW ALL;
```

Menghapus grup parameter cluster DB

Anda dapat menghapus grup parameter cluster DB menggunakan AWS Management Console, AWS CLI, atau RDS API. Grup parameter grup parameter cluster DB memenuhi syarat untuk dihapus hanya jika tidak terkait dengan cluster DB.

Konsol

Untuk menghapus grup parameter

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.

   Grup parameter muncul dalam daftar.
3. Pilih nama grup parameter cluster DB yang akan dihapus.
4. Pilih Tindakan dan kemudian Hapus.
5. Tinjau nama grup parameter dan kemudian pilih Hapus.

AWS CLI

Untuk menghapus grup parameter cluster DB, gunakan AWS CLI `delete-db-cluster-parameter-group`perintah dengan parameter yang diperlukan berikut.

- `--db-parameter-group-name`

Example

Contoh berikut menghapus kelompok parameter cluster DB bernama mydbparametergroup.

```
aws rds delete-db-cluster-parameter-group --db-parameter-group-name mydbparametergroup
```

API RDS

Untuk menghapus grup parameter cluster DB, gunakan `DeleteDBClusterParameterGroup`perintah RDS API dengan parameter wajib berikut.

- `DBParameterGroupName`

Bekerja dengan grup parameter DB dalam instance DB

Instans DB menggunakan grup parameter DB. Bagian berikut menjelaskan cara mengonfigurasi dan mengelola grup parameter instans DB.

Topik

- Membuat grup parameter DB
- Mengaitkan grup parameter DB dengan instans DB
- Memodifikasi parameter dalam grup parameter DB
- Mengatur ulang parameter dalam grup parameter DB ke nilai defaultnya
- Menyalin grup parameter DB
- Mencantumkan grup parameter DB
- Melihat nilai parameter untuk grup parameter DB
- Menghapus grup parameter DB

Membuat grup parameter DB

Anda dapat membuat grup parameter DB baru menggunakan AWS Management Console, AWS CLI, atau RDS API.

Batasan berikut berlaku untuk nama grup parameter DB:

- Nama harus berisi 1 sampai 255 huruf, angka, atau tanda hubung.

  Nama grup parameter default boleh menyertakan titik, seperti `default.mysql8.0`. Namun, nama grup parameter kustom tidak boleh menyertakan titik.
- Karakter pertamanya harus berupa huruf.
- Nama tidak boleh diakhiri dengan tanda hubung atau berisi dua tanda hubung berturut-turut.

Konsol

Untuk membuat grup parameter DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Pilih Buat grup parameter.

   Jendela Buat grup parameter akan muncul.
4. Dalam daftar Kelompok grup parameter, pilih kelompok grup parameter DB.
5. Dalam daftar Jenis, jika relevan, pilih Grup Parameter DB.
6. Di kotak Nama grup, masukkan nama grup parameter DB yang baru.
7. Di kotak Deskripsi, masukkan deskripsi untuk grup parameter DB yang baru.
8. Pilih Buat.

AWS CLI

Untuk membuat grup parameter DB, gunakan AWS CLI `create-db-parameter-group`perintah. Contoh berikut membuat grup parameter DB yang bernama mydbparametergroup untuk MySQL versi 8.0 dengan deskripsi "Grup parameter baru saya".

Sertakan parameter wajib berikut:

- `--db-parameter-group-name`
- `--db-parameter-group-family`
- `--description`

Untuk mencantumkan semua kelompok grup parameter yang tersedia, gunakan perintah berikut:

```
aws rds describe-db-engine-versions --query "DBEngineVersions[].DBParameterGroupFamily"
```

> **Note**
>
> Output berisi duplikat.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --db-parameter-group-family aurora-mysql5.7 \
    --description "My new parameter group"
```

Untuk Windows:

```
aws rds create-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --db-parameter-group-family aurora-mysql5.7 ^
    --description "My new parameter group"
```

Perintah ini menghasilkan output yang serupa dengan yang berikut:

```
DBPARAMETERGROUP  mydbparametergroup  aurora-mysql5.7  My new parameter group
```

API RDS

Untuk membuat grup parameter DB, gunakan operasi `CreateDBParameterGroup` API RDS.

Sertakan parameter wajib berikut:

- `DBParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

Mengaitkan grup parameter DB dengan instans DB

Anda dapat membuat grup parameter DB Anda sendiri dengan pengaturan yang disesuaikan. Anda dapat mengaitkan grup parameter DB dengan instans DB menggunakan AWS CLI,, atau RDS API. AWS Management Console Anda dapat melakukannya saat membuat atau memodifikasi instans DB.

Untuk informasi tentang cara membuat grup parameter DB, lihat Membuat grup parameter DB. Untuk informasi tentang cara memodifikasi instans DB, lihat Memodifikasi instans DB dalam klaster DB.

> **Note**
>
> Ketika Anda mengaitkan grup parameter DB baru dengan instans DB, parameter statis dan dinamis yang dimodifikasi diterapkan hanya setelah instans DB di-reboot. Namun, jika Anda memodifikasi parameter dinamis dalam grup parameter DB setelah Anda mengaitkannya dengan instans DB, perubahan ini diterapkan segera tanpa reboot.

Konsol

Untuk mengaitkan grup parameter DB dengan instans DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih instans DB yang ingin Anda modifikasi.
3. Pilih Modifikasi. Halaman Modifikasi instans DB akan muncul.
4. Ubah pengaturan Grup parameter DB.
5. Pilih Lanjutkan dan periksa ringkasan modifikasi.
6. (Opsional) Pilih Terapkan langsung untuk langsung menerapkan perubahan. Memilih opsi ini dapat menyebabkan penonaktifan dalam beberapa kasus.
7. Di halaman konfirmasi, tinjau perubahan Anda. Jika sudah benar, pilih Modifikasi instans DB untuk menyimpan perubahan Anda.

   Atau pilih Kembali untuk mengedit perubahan atau Batal untuk membatalkan perubahan.

AWS CLI

Untuk mengaitkan grup parameter DB dengan instans DB, gunakan AWS CLI `modify-db-instance`perintah dengan opsi berikut:

- `--db-instance-identifier`
- `--db-parameter-group-name`

Contoh berikut mengaitkan Grup parameter DB `mydbpg` dengan instans DB `database-1`. Perubahan langsung diterapkan dengan menggunakan `--apply-immediately`. Gunakan `--no-apply-immediately` untuk menerapkan perubahan selama masa pemeliharaan berikutnya.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier database-1 \
    --db-parameter-group-name mydbpg \
    --apply-immediately
```

Untuk Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier database-1 ^
    --db-parameter-group-name mydbpg ^
    --apply-immediately
```

API RDS

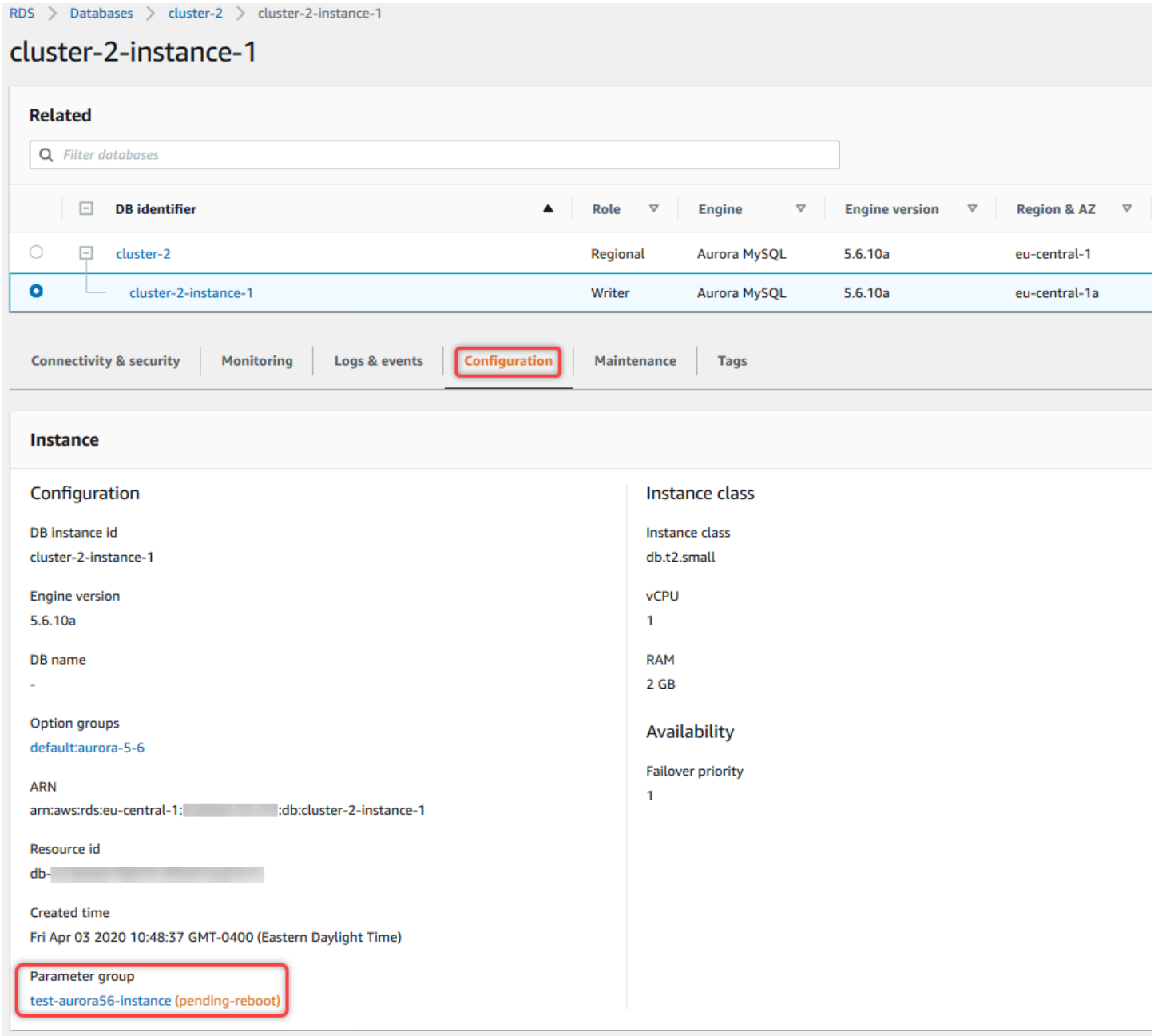
Untuk mengaitkan grup parameter DB dengan instans DB, gunakan operasi `ModifyDBInstance` API RDS dengan parameter berikut:

- `DBInstanceName`
- `DBParameterGroupName`

Memodifikasi parameter dalam grup parameter DB

Anda dapat memodifikasi nilai parameter dalam grup parameter DB buatan pelanggan; Anda tidak dapat mengubah nilai parameter dalam grup parameter DB default. Perubahan pada parameter dalam grup parameter DB buatan pelanggan diterapkan ke semua instans DB yang dikaitkan dengan grup parameter DB.

Perubahan pada beberapa parameter diterapkan langsung ke instans DB tanpa reboot. Perubahan pada parameter lain diterapkan hanya setelah instans DB di-reboot. Konsol RDS menampilkan status grup parameter DB yang terkait dengan instans DB pada tab Konfigurasi. Misalnya, instans DB tidak menggunakan perubahan terbaru pada grup parameter DB terkait. Jika demikian, konsol RDS menampilkan grup parameter DB dengan status Reboot tertunda. Untuk menerapkan perubahan parameter terbaru pada instans DB tersebut, reboot instans DB secara manual.



Konsol

Untuk memodifikasi grup parameter DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter yang ingin Anda modifikasi.
4. Untuk Tindakan grup parameter, pilih Edit.
5. Ubah nilai parameter yang ingin Anda modifikasi. Anda dapat menggulir parameter menggunakan tombol panah di bagian kanan atas kotak dialog.

   Anda tidak dapat mengubah nilai dalam grup parameter default.
6. Pilih Simpan perubahan.

AWS CLI

Untuk memodifikasi grup parameter DB, gunakan AWS CLI `modify-db-parameter-group`perintah dengan opsi yang diperlukan berikut:

- `--db-parameter-group-name`
- `--parameters`

Contoh berikut memodifikasi nilai `max_connections` dan `max_allowed_packet` dalam grup parameter DB yang bernama mydbparametergroup.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" \

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters
 "ParameterName=max_connections,ParameterValue=250,ApplyMethod=immediate" ^

 "ParameterName=max_allowed_packet,ParameterValue=1024,ApplyMethod=immediate"
```

Perintah menghasilkan output seperti berikut:

```
DBPARAMETERGROUP  mydbparametergroup
```

API RDS

Untuk memodifikasi grup parameter DB, gunakan operasi `ModifyDBParameterGroup` API RDS dengan parameter wajib berikut:

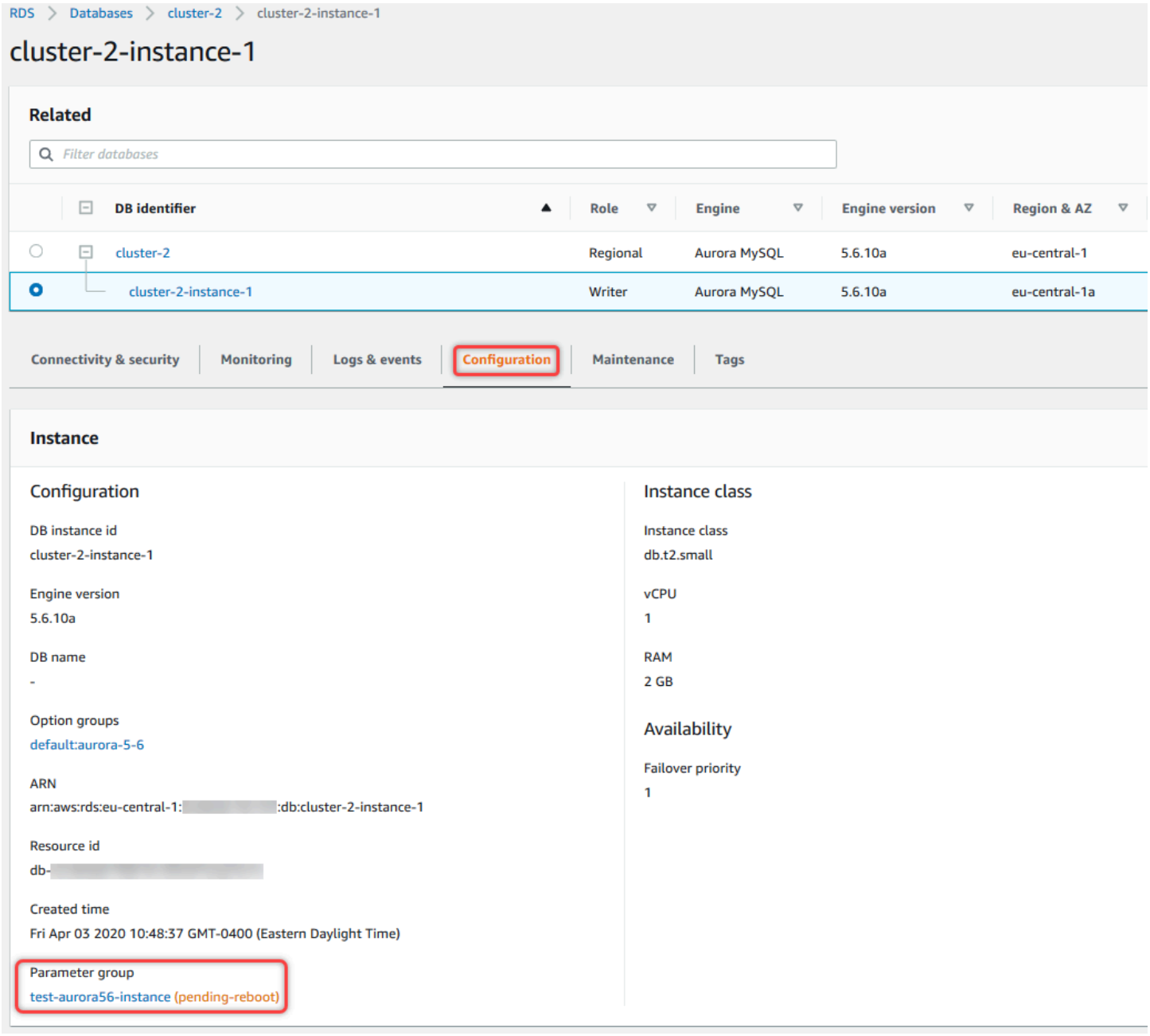
- `DBParameterGroupName`
- `Parameters`

Mengatur ulang parameter dalam grup parameter DB ke nilai defaultnya

Anda dapat mengatur ulang nilai parameter dalam grup parameter DB buatan pelanggan ke nilai defaultnya. Perubahan pada parameter dalam grup parameter DB buatan pelanggan diterapkan ke semua instans DB yang dikaitkan dengan grup parameter DB.

Saat Anda menggunakan konsol, Anda dapat mengatur ulang parameter tertentu ke nilai defaultnya. Namun, Anda tidak dapat dengan mudah mengatur ulang semua parameter dalam grup parameter DB sekaligus. Saat Anda menggunakan AWS CLI atau RDS API, Anda dapat mengatur ulang parameter tertentu ke nilai defaultnya. Anda juga dapat mengatur ulang semua parameter dalam grup parameter DB sekaligus.

Perubahan pada beberapa parameter diterapkan langsung ke instans DB tanpa reboot. Perubahan pada parameter lain diterapkan hanya setelah instans DB di-reboot. Konsol RDS menampilkan status grup parameter DB yang terkait dengan instans DB pada tab Konfigurasi. Misalnya, instans DB tidak menggunakan perubahan terbaru pada grup parameter DB terkait. Jika demikian, konsol RDS menampilkan grup parameter DB dengan status Reboot tertunda. Untuk menerapkan perubahan parameter terbaru pada instans DB tersebut, reboot instans DB secara manual.



> **Note**
>
> Dalam grup parameter DB default, parameter selalu ditetapkan ke nilai defaultnya.

Konsol

Untuk mengatur ulang parameter dalam grup parameter DB ke nilai defaultnya

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter.
4. Untuk Tindakan grup parameter, pilih Edit.
5. Pilih parameter yang ingin Anda atur ulang ke nilai defaultnya. Anda dapat menggulir parameter menggunakan tombol panah di bagian kanan atas kotak dialog.

   Anda tidak dapat mengatur ulang nilai dalam grup parameter default.
6. Pilih Atur ulang lalu konfirmasikan dengan memilih Atur ulang parameter.

AWS CLI

Untuk mengatur ulang beberapa atau semua parameter dalam grup parameter DB, gunakan AWS CLI `reset-db-parameter-group`perintah dengan opsi wajib berikut:`--db-parameter-group-name`.

Untuk mengatur ulang semua parameter dalam grup parameter DB, tentukan opsi `--reset-all-parameters`. Untuk mengatur ulang parameter tertentu, tentukan opsi `--parameters`.

Contoh berikut mengatur ulang semua parameter dalam grup parameter DB yang bernama mydbparametergroup ke nilai defaultnya.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --reset-all-parameters
```

Untuk Windows:

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
```

```
    --reset-all-parameters
```

Contoh berikut mengatur ulang opsi `max_connections` dan `max_allowed_packet` ke nilai defaultnya dalam grup parameter DB yang bernama mydbparametergroup.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds reset-db-parameter-group \
    --db-parameter-group-name mydbparametergroup \
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" \
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds reset-db-parameter-group ^
    --db-parameter-group-name mydbparametergroup ^
    --parameters "ParameterName=max_connections,ApplyMethod=immediate" ^
                 "ParameterName=max_allowed_packet,ApplyMethod=immediate"
```

Perintah menghasilkan output seperti berikut:

```
DBParameterGroupName  mydbparametergroup
```

API RDS

Untuk mengatur ulang parameter dalam grup parameter DB ke nilai defaultnya, gunakan perintah `ResetDBParameterGroup` API RDS dengan parameter wajib berikut: `DBParameterGroupName`.

Untuk mengatur ulang semua parameter dalam grup parameter DB, atur parameter `ResetAllParameters` ke `true`. Untuk mengatur ulang parameter tertentu, tentukan parameter `Parameters`.

Menyalin grup parameter DB

Anda dapat menyalin grup parameter DB kustom yang Anda buat. Menyalin grup parameter bisa menjadi solusi yang mudah. Contohnya adalah saat Anda telah membuat grup parameter DB dan ingin menyertakan sebagian besar parameter dan nilai kustomnya dalam grup parameter DB yang baru. Anda dapat menyalin grup parameter DB dengan menggunakan file AWS Management

Console. Anda juga dapat menggunakan AWS CLI copy-db-parameter-groupperintah atau operasi RDS API CopyDB ParameterGroup.

Setelah Anda menyalin grup parameter DB, tunggu minimal 5 menit sebelum membuat instans DB pertama Anda yang menggunakan grup parameter DB tersebut sebagai grup parameter default. Cara ini memungkinkan Amazon RDS menyelesaikan sepenuhnya tindakan penyalinan sebelum grup parameter digunakan. Tindakan ini harus dilakukan terutama untuk parameter yang sangat penting saat membuat basis data default untuk instans DB. Contohnya adalah kumpulan karakter untuk basis data default yang ditentukan oleh parameter `character_set_database`. Gunakan opsi Parameter Grup konsol Amazon RDS atau describe-db-parametersperintah untuk memverifikasi bahwa grup parameter DB Anda dibuat.

> **Note**
>
> Anda tidak dapat menyalin grup parameter default. Namun, Anda dapat membuat grup parameter baru yang didasarkan pada grup parameter default.
> Anda tidak dapat menyalin grup parameter DB ke grup parameter yang berbeda Akun AWS atau Wilayah AWS.

Konsol

Untuk menyalin grup parameter DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter kustom yang ingin Anda salin.
4. Untuk Tindakan grup parameter, pilih Salin.
5. Di Pengidentifikasi grup parameter DB baru, masukkan nama untuk grup parameter baru.
6. Di Deskripsi, masukkan deskripsi untuk grup parameter DB yang baru.
7. Pilih Salin.

AWS CLI

Untuk menyalin grup parameter DB, gunakan AWS CLI `copy-db-parameter-group`perintah dengan opsi yang diperlukan berikut:

- `--source-db-parameter-group-identifier`
- `--target-db-parameter-group-identifier`
- `--target-db-parameter-group-description`

Contoh berikut membuat grup parameter DB baru yang bernama `mygroup2` yang merupakan salinan dari grup parameter DB `mygroup1`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds copy-db-parameter-group \
    --source-db-parameter-group-identifier mygroup1 \
    --target-db-parameter-group-identifier mygroup2 \
    --target-db-parameter-group-description "DB parameter group 2"
```

Untuk Windows:

```
aws rds copy-db-parameter-group ^
    --source-db-parameter-group-identifier mygroup1 ^
    --target-db-parameter-group-identifier mygroup2 ^
    --target-db-parameter-group-description "DB parameter group 2"
```

API RDS

Untuk menyalin grup parameter DB, gunakan operasi `CopyDBParameterGroup` API RDS dengan parameter wajib berikut:

- `SourceDBParameterGroupIdentifier`
- `TargetDBParameterGroupIdentifier`
- `TargetDBParameterGroupDescription`

Mencantumkan grup parameter DB

Anda dapat mencantumkan grup parameter DB yang telah Anda buat untuk AWS akun Anda.

> **Note**
>
> Grup parameter default secara otomatis dibuat dari template parameter default ketika Anda membuat instans DB untuk mesin dan versi DB tertentu. Grup parameter default ini berisi pengaturan parameter pilihan dan tidak dapat dimodifikasi. Saat membuat grup parameter kustom, Anda dapat memodifikasi pengaturan parameter.

Konsol

Untuk mencantumkan semua grup parameter DB untuk AWS akun

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.

   Grup parameter DB akan muncul dalam daftar.

AWS CLI

Untuk membuat daftar semua grup parameter DB untuk AWS akun, gunakan AWS CLI `describe-db-parameter-groups`perintah.

Example

Contoh berikut mencantumkan semua grup parameter DB yang tersedia untuk akun AWS .

```
aws rds describe-db-parameter-groups
```

Perintah memberikan respons seperti berikut:

```
DBPARAMETERGROUP  default.mysql8.0     mysql8.0  Default parameter group for MySQL8.0
DBPARAMETERGROUP  mydbparametergroup   mysql8.0  My new parameter group
```

Contoh berikut menjelaskan grup parameter mydbparamgroup1.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-parameter-groups \
    --db-parameter-group-name mydbparamgroup1
```

Untuk Windows:

```
aws rds describe-db-parameter-groups ^
    --db-parameter-group-name mydbparamgroup1
```

Perintah memberikan respons seperti berikut:

```
DBPARAMETERGROUP  mydbparametergroup1  mysql8.0  My new parameter group
```

API RDS

Untuk mencantumkan semua grup parameter DB untuk AWS akun, gunakan `DescribeDBParameterGroups`operasi RDS API.

Melihat nilai parameter untuk grup parameter DB

Anda dapat memperoleh daftar semua parameter dalam grup parameter DB beserta nilainya.

Konsol

Untuk melihat nilai parameter untuk grup parameter DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.

   Grup parameter DB akan muncul dalam daftar.
3. Pilih nama grup parameter untuk melihat daftar parameternya.

AWS CLI

Untuk melihat nilai parameter untuk grup parameter DB, gunakan AWS CLI `describe-db-parameters`perintah dengan parameter yang diperlukan berikut.

- `--db-parameter-group-name`

Example

Contoh berikut mencantumkan parameter dan nilai parameter untuk grup parameter DB yang bernama mydbparametergroup.

```
aws rds describe-db-parameters --db-parameter-group-name mydbparametergroup
```

Perintah memberikan respons seperti berikut:

```
DBPARAMETER  Parameter Name            Parameter Value  Source           Data Type
 Apply Type  Is Modifiable
DBPARAMETER  allow-suspicious-udfs                      engine-default   boolean
 static      false
DBPARAMETER  auto_increment_increment                   engine-default   integer
 dynamic     true
DBPARAMETER  auto_increment_offset                      engine-default   integer
 dynamic     true
DBPARAMETER  binlog_cache_size         32768            system           integer
 dynamic     true
DBPARAMETER  socket                    /tmp/mysql.sock  system           string
 static      false
```

API RDS

Untuk melihat nilai parameter untuk grup parameter DB, gunakan perintah `DescribeDBParameters` API RDS dengan parameter wajib berikut.

- `DBParameterGroupName`

Menghapus grup parameter DB

Anda dapat menghapus grup parameter DB menggunakan AWS Management Console, AWS CLI, atau RDS API. Grup parameter memenuhi syarat untuk dihapus hanya jika tidak terkait dengan instans DB.

Konsol

Untuk menghapus grup parameter DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.

   Grup parameter DB akan muncul dalam daftar.
3. Pilih nama grup parameter yang akan dihapus.

4. Pilih Tindakan dan kemudian Hapus.
5. Tinjau nama grup parameter dan kemudian pilih Hapus.

AWS CLI

Untuk menghapus grup parameter DB, gunakan AWS CLI `delete-db-parameter-group`perintah dengan parameter yang diperlukan berikut.

- `--db-parameter-group-name`

Example

Contoh berikut menghapus kelompok parameter DB bernama mydbparametergroup.

```
aws rds delete-db-parameter-group --db-parameter-group-name mydbparametergroup
```

API RDS

Untuk menghapus grup parameter DB, gunakan `DeleteDBParameterGroup`perintah RDS API dengan parameter wajib berikut.

- `DBParameterGroupName`

Membandingkan grup parameter DB

Anda dapat menggunakan AWS Management Console untuk melihat perbedaan antara dua kelompok parameter DB.

Kedua grup parameter yang ditentukan harus merupakan grup parameter DB atau grup parameter klaster DB. Hal ini berlaku meskipun mesin dan versi DB sama. Misalnya, Anda tidak dapat membandingkan grup parameter DB `aurora-mysql8.0` (Aurora MySQL versi 3) dan grup parameter klaster DB `aurora-mysql8.0`.

Anda dapat membandingkan grup parameter DB Aurora MySQL dan RDS for MySQL, bahkan untuk versi yang berbeda, tetapi Anda tidak dapat membandingkan grup parameter DB Aurora PostgreSQL dan RDS for PostgreSQL.

Untuk membandingkan dua kelompok parameter DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup Parameter.
3. Dalam daftar, pilih grup parameter yang ingin Anda bandingkan.

   > **Note**
   >
   > Untuk membandingkan grup parameter default dengan grup parameter kustom, pilih grup parameter default pada tab Default terlebih dahulu, lalu pilih grup parameter kustom pada tab Kustom.

4. Dari Tindakan, pilih Bandingkan.

Menentukan parameter DB

Jenis parameter DB mencakup yang berikut ini:

- Bilangan Bulat
- Boolean
- String
- Panjang
- Ganda
- Stempel Waktu
- Objek dari tipe data lain yang ditentukan
- Rangkaian nilai tipe bilangan bulat, Boolean, string, panjang, ganda, stempel waktu, atau objek

Anda juga dapat menentukan parameter bilangan bulat dan Boolean menggunakan ekspresi, formula, dan fungsi.

Daftar Isi

- Formula parameter DB
  - Variabel formula parameter DB
  - Operator formula parameter DB

- Fungsi parameter DB
- Ekspresi log parameter DB
- Contoh nilai parameter DB

Formula parameter DB

Formula parameter DB adalah ekspresi yang menghasilkan nilai bilangan bulat atau nilai Boolean. Anda mengapit ekspresi dalam kurung: {}. Anda dapat menentukan formula untuk nilai parameter DB atau sebagai argumen untuk fungsi parameter DB.

Sintaks

```
{FormulaVariable}
{FormulaVariable*Integer}
{FormulaVariable*Integer/Integer}
{FormulaVariable/Integer}
```

Variabel formula parameter DB

Setiap variabel formula menghasilkan nilai bilangan bulat atau Boolean. Nama variabel bersifat peka huruf besar-kecil.

AllocatedStorage

Mengembalikan bilangan bulat yang mewakili ukuran, dalam byte, dari volume data.

DB InstanceClassMemory

Mengembalikan bilangan bulat untuk jumlah byte memori yang tersedia untuk proses basis data. Angka ini dihitung secara internal dengan memulai dengan jumlah total memori untuk kelas instans DB. Dari sini, perhitungan mengurangi memori yang disediakan untuk sistem operasi dan proses RDS yang mengelola instans. Oleh karena itu, angkanya selalu agak lebih rendah dari angka memori yang ditunjukkan pada tabel kelas instans di Kelas instans DB Aurora. Nilai yang pasti bergantung pada kombinasi faktor. Faktor ini termasuk kelas instans, mesin DB, dan apakah hal tersebut berlaku untuk instans RDS atau instans yang merupakan bagian dari klaster Aurora.

EndPointPort

Mengembalikan bilangan bulat yang mewakili port yang digunakan saat terhubung ke instans DB.

TrueIfReplica

Mengembalikan 1 jika instans DB adalah replika baca dan 0 jika tidak. Nilai ini adalah nilai default untuk parameter `read_only` di Aurora MySQL.

Operator formula parameter DB

Formula parameter DB mendukung dua operator: pembagian dan perkalian.

Operator pembagian: /

Membagi dividen dengan pembagi, mengembalikan hasil bagi bilangan bulat. Desimal dalam hasil bagi dipotong, tidak dibulatkan.

Sintaks

```
dividend / divisor
```

Argumen terbagi dan pembagi harus merupakan ekspresi bilangan bulat.

Operator perkalian: *

Mengalikan ekspresi, mengembalikan hasil ekspresi. Desimal dalam ekspresi dipotong, tidak dibulatkan.

Sintaks

```
expression * expression
```

Kedua ekspresi harus berupa bilangan bulat.

Fungsi parameter DB

Anda menentukan argumen fungsi parameter DB sebagai bilangan bulat atau formula.
Setiap fungsi harus memiliki setidaknya satu argumen. Tentukan beberapa argumen sebagai daftar yang dipisahkan koma. Daftar ini tidak dapat memiliki anggota yang kosong, seperti argument1,,argument3. Nama fungsi bersifat peka huruf besar-kecil.

JIKA

Mengembalikan argumen.

Sintaks

```
IF(argument1, argument2, argument3)
```

Mengembalikan argumen kedua jika argumen pertama ternyata benar setelah dievaluasi. Mengembalikan argumen ketiga.

TERBESAR

Mengembalikan nilai terbesar dari daftar bilangan bulat atau formula parameter.

Sintaks

```
GREATEST(argument1, argument2,...argumentn)
```

Mengembalikan bilangan bulat.

TERKECIL

Mengembalikan nilai terkecil dari daftar bilangan bulat atau formula parameter.

Sintaks

```
LEAST(argument1, argument2,...argumentn)
```

Mengembalikan bilangan bulat.

JUMLAH

Menambahkan nilai bilangan bulat atau formula parameter yang ditentukan.

Sintaks

```
SUM(argument1, argument2,...argumentn)
```

Mengembalikan bilangan bulat.

Ekspresi log parameter DB

Anda dapat mengatur nilai parameter DB bilangan bulat ke ekspresi log. Anda mengapit ekspresi dalam kurung: {}. Sebagai contoh:

```
{log(DBInstanceClassMemory/8187281418)*1000}
```

Fungsi `log` mewakili dasar log 2. Contoh ini juga menggunakan variabel formula `DBInstanceClassMemory`. Lihat Variabel formula parameter DB.

Contoh nilai parameter DB

Contoh-contoh ini menunjukkan penggunaan formula, fungsi, dan ekspresi untuk nilai parameter DB.

> **Warning**
>
> Pengaturan parameter yang tidak tepat dalam grup parameter DB dapat memiliki efek merugikan yang tidak diinginkan. Efek tersebut termasuk performa terdegradasi dan ketidakstabilan sistem. Selalu berhati-hati saat memodifikasi parameter basis data dan cadangkan data Anda sebelum memodifikasi grup parameter DB Anda. Cobalah perubahan grup parameter pada instance DB pengujian, yang dibuat menggunakan point-in-time-restores, sebelum menerapkan perubahan grup parameter tersebut ke instans DB produksi Anda.

Example Menggunakan fungsi parameter DB TERKECIL

Anda dapat menentukan fungsi `LEAST` dalam nilai parameter `table_definition_cache` Aurora MySQL. Gunakan untuk mengatur jumlah definisi tabel yang dapat disimpan dalam cache definisi menjadi yang lebih kecil antara `DBInstanceClassMemory`/393040 atau 20.000.

```
LEAST({DBInstanceClassMemory/393040}, 20000)
```

Memigrasikan data ke klaster DB Amazon Aurora

Anda memiliki beberapa opsi untuk memigrasikan data dari basis data yang sudah ada ke klaster DB Amazon Aurora, bergantung pada kompatibilitas mesin basis data. Opsi migrasi Anda juga bergantung pada basis data asal migrasi Anda dan ukuran data yang Anda migrasikan.

Memigrasikan data ke klaster DB Amazon Aurora MySQL

Anda dapat memigrasikan data dari salah satu sumber berikut ke klaster DB Amazon Aurora MySQL.

- Instans DB RDS for MySQL
- Basis data MySQL eksternal di luar Amazon RDS
- Basis data yang tidak kompatibel dengan MySQL

Untuk informasi selengkapnya, lihat Memigrasikan data ke klaster DB Amazon Aurora MySQL.

Memigrasikan data ke klaster DB Amazon Aurora PostgreSQL

Anda dapat memigrasikan data dari salah satu sumber berikut ke klaster DB Amazon Aurora PostgreSQL.

- Instans DB Amazon RDS PostgreSQL
- Basis data yang tidak kompatibel dengan PostgreSQL

Untuk informasi selengkapnya, lihat Memigrasikan data ke Amazon Aurora dengan kompatibilitas PostgreSQL.

Membuat ElastiCache cache Amazon menggunakan pengaturan instans DB cluster Aurora DB

ElastiCache adalah layanan caching dalam memori yang dikelola sepenuhnya yang menyediakan latensi baca dan tulis mikrodetik yang mendukung kasus penggunaan real-time yang fleksibel. ElastiCache dapat membantu Anda mempercepat kinerja aplikasi dan database. Anda dapat menggunakan ElastiCache sebagai penyimpanan data utama untuk kasus penggunaan yang tidak memerlukan daya tahan data, seperti papan peringkat game, streaming, dan analitik data. ElastiCache membantu menghilangkan kompleksitas yang terkait dengan penyebaran dan pengelolaan lingkungan komputasi terdistribusi. Untuk informasi selengkapnya, lihat Kasus ElastiCache Penggunaan Umum dan Bagaimana ElastiCache Dapat Membantu untuk Memcached dan Kasus ElastiCache Penggunaan Umum dan Bagaimana ElastiCache Dapat Membantu Redis. Anda dapat menggunakan konsol Amazon RDS untuk membuat ElastiCache cache.

Anda dapat mengoperasikan Amazon ElastiCache dalam dua format. Anda dapat memulai dengan cache nirserver atau memilih untuk merancang klaster cache Anda sendiri. Jika Anda memilih untuk mendesain cluster cache Anda sendiri, ElastiCache bekerja dengan mesin Redis dan Memcached. Jika Anda tidak yakin mesin mana yang ingin Anda gunakan, lihat Membandingkan Memcached dan Redis. Untuk informasi selengkapnya tentang Amazon ElastiCache, lihat Panduan ElastiCache Pengguna Amazon.

Topik

- Ikhtisar pembuatan ElastiCache cache dengan pengaturan instans DB cluster Aurora DB
- Membuat ElastiCache cache dengan pengaturan dari instance Aurora DB cluster DB

Ikhtisar pembuatan ElastiCache cache dengan pengaturan instans DB cluster Aurora DB

Anda dapat membuat ElastiCache cache dari Amazon RDS menggunakan pengaturan konfigurasi yang sama dengan instans Aurora DB yang baru dibuat atau yang sudah ada.

Beberapa kasus penggunaan untuk mengaitkan ElastiCache cache dengan Anda:

- Anda dapat menghemat biaya dan meningkatkan kinerja Anda ElastiCache dengan menggunakan RDS versus berjalan pada RDS saja.

- Anda dapat menggunakan ElastiCache cache sebagai penyimpanan data utama untuk aplikasi yang tidak memerlukan daya tahan data. Aplikasi Anda yang menggunakan Redis atau Memcached dapat digunakan ElastiCache dengan hampir tidak ada modifikasi.

- ElastiCache pengaturan konektivitas
- ElastiCache pengaturan keamanan

Anda juga dapat mengatur pengaturan konfigurasi cache sesuai dengan kebutuhan Anda.

Menyiapkan ElastiCache di aplikasi Anda

Aplikasi Anda harus diatur untuk memanfaatkan ElastiCache cache. Anda juga dapat mengoptimalkan dan meningkatkan kinerja cache dengan menyiapkan aplikasi Anda untuk menggunakan strategi caching tergantung pada kebutuhan Anda.

- Untuk mengakses ElastiCache cache dan memulai, lihat Memulai Amazon ElastiCache untuk Redis dan Memulai Amazon ElastiCache untuk Memcached.
- Untuk informasi lebih lanjut tentang strategi caching, lihat Strategi caching dan praktik terbaik untuk Memcached dan Strategi caching dan praktik terbaik untuk Redis.
- Untuk informasi selengkapnya tentang ketersediaan tinggi di ElastiCache klaster Redis, lihat Ketersediaan tinggi menggunakan grup replikasi.
- Anda mungkin dikenakan biaya yang terkait dengan penyimpanan cadangan, transfer data di dalam atau di seluruh wilayah, atau penggunaan. AWS Outposts Untuk detail harga, lihat ElastiCache harga Amazon.

Membuat ElastiCache cache dengan pengaturan dari instance Aurora DB cluster DB

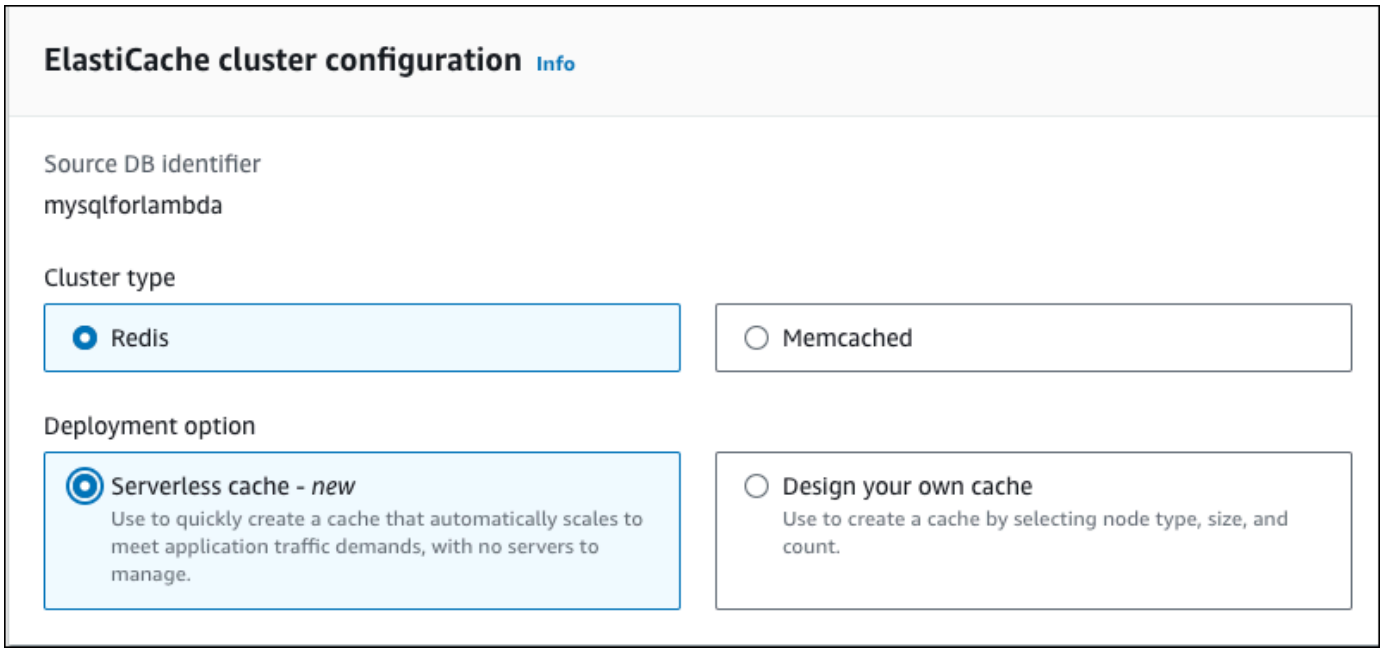
Buat ElastiCache cache dengan pengaturan dari cluster DB

1. Untuk membuat klaster DB, ikuti petunjuk di Membuat klaster DB Amazon Aurora.
2. Setelah membuat , konsol menampilkan jendela add-on yang Disarankan. Pilih Buat ElastiCache cluster dari RDS menggunakan pengaturan DB Anda.

   Untuk database yang ada, di halaman Databases, pilih cluster DB yang diperlukan.

Di bagian ElastiCache konfigurasi, pengidentifikasi Source DB menampilkan cluster DB mana yang mewarisi pengaturan ElastiCache cache.

3. Pilih apakah Anda ingin membuat klaster Redis atau Memcached. Untuk informasi selengkapnya, lihat Membandingkan Memcached dan Redis.



4. Setelah ini, pilih apakah Anda ingin membuat cache Tanpa Server atau Desain cache Anda sendiri. Untuk informasi selengkapnya, lihat Memilih antara opsi penerapan.

   Jika Anda memilih Cache tanpa server:

   a. Dalam pengaturan Cache, masukkan nilai untuk Nama dan Deskripsi.

   b. Di bawah Lihat pengaturan default, biarkan pengaturan default untuk membuat koneksi antara cache Anda dan cluster DB.

   c. Anda juga dapat mengedit pengaturan default dengan memilih Sesuaikan pengaturan default. Pilih pengaturan ElastiCache konektivitas, pengaturan ElastiCache keamanan, dan Batas penggunaan maksimum.

5. Jika Anda memilih Desain cache Anda sendiri:

   a. Jika Anda memilih klaster Redis, pilih apakah Anda ingin mempertahankan mode cluster Diaktifkan atau Dinonaktifkan. Untuk informasi selengkapnya, lihat Replikasi: Redis (Mode Klaster Dinonaktifkan) vs Redis (Mode Klaster Diaktifkan).

   b. Masukkan nilai untuk Nama, Deskripsi, dan Versi mesin.

Untuk Versi mesin, nilai default yang disarankan adalah versi mesin terbaru. Anda juga dapat memilih versi Engine untuk ElastiCache cache yang paling sesuai dengan kebutuhan Anda.

c. Pilih jenis simpul dalam opsi Jenis simpul. Untuk informasi selengkapnya, lihat Mengelola simpul.

   Jika Anda memilih untuk membuat klaster Redis dengan Mode klaster disetel ke Diaktifkan, maka masukkan jumlah serpihan (partisi/grup simpul) dalam opsi Jumlah serpihan.

   Masukkan jumlah replika setiap serpihan di Jumlah replika.

   > **Note**
   >
   > Jenis node yang dipilih, jumlah pecahan, dan jumlah replika semuanya memengaruhi kinerja cache dan biaya sumber daya Anda. Pastikan pengaturan ini sesuai dengan kebutuhan basis data Anda. Untuk informasi harga, lihat ElastiCache harga Amazon.

d. Pilih pengaturan ElastiCache konektivitas dan pengaturan ElastiCache keamanan. Anda dapat menyimpan pengaturan default atau menyesuaikan pengaturan ini sesuai kebutuhan Anda.

6. Verifikasi pengaturan default dan warisan ElastiCache cache Anda. Beberapa pengaturan tidak dapat diubah setelah pembuatan.

   > **Note**
   >
   > RDS mungkin menyesuaikan jendela cadangan ElastiCache cache Anda untuk memenuhi persyaratan jendela minimum 60 menit. Periode pencadangan basis data sumber Anda tetap sama.

7. Saat Anda siap, pilih Buat ElastiCache cache.

Konsol menampilkan spanduk konfirmasi untuk pembuatan ElastiCache cache. Ikuti tautan di spanduk ke ElastiCache konsol untuk melihat detail cache. ElastiCache Konsol menampilkan ElastiCache cache yang baru dibuat.

Mengelola klaster DB Amazon Aurora

Bagian ini menunjukkan cara mengatur klaster DB Aurora Anda. Aurora melibatkan klaster server basis data yang terhubung dalam topologi replikasi. Dengan demikian, mengelola Aurora sering kali melibatkan penerapan perubahan pada beberapa server dan memastikan bahwa semua Replika Aurora terus mengikuti server master. Karena Aurora secara transparan menskalakan penyimpanan yang mendasari saat data Anda tumbuh, pengelolaan Aurora memerlukan manajemen penyimpanan disk yang relatif kecil. Demikian pula, karena Aurora secara otomatis melakukan pencadangan berkelanjutan, klaster Aurora tidak memerlukan perencanaan atau waktu henti yang ekstensif untuk melakukan pencadangan.

Topik

- Menghentikan dan memulai klaster DB Amazon Aurora
- Secara otomatis menghubungkan sumber daya komputasi AWS dan klaster DB Aurora
- Memodifikasi klaster DB Amazon Aurora
- Menambahkan Replika Aurora ke klaster DB
- Mengelola performa dan penskalaan untuk klaster DB Aurora
- Mengkloning volume untuk klaster DB Amazon Aurora
- Mengintegrasikan Aurora dengan layanan AWS lainnya
- Memelihara klaster DB Amazon Aurora
- Mem-boot ulang klaster DB Amazon Aurora atau instans DB Amazon Aurora
- Menghapus instans DB atau klaster DB Aurora
- Memberi tag pada sumber daya Amazon RDS
- Bekerja dengan Amazon Resource Name (ARN) di Amazon RDS
- Pembaruan Amazon Aurora

Menghentikan dan memulai klaster DB Amazon Aurora

Menghentikan dan memulai klaster Amazon Aurora membantu Anda mengelola biaya untuk pengembangan dan pengujian lingkungan. Anda dapat menghentikan sementara semua instans DB dalam klaster Anda, dan tidak perlu menyiapkan dan merombak semua instans DB setiap kali menggunakan klaster.

Topik

- Ikhtisar menghentikan dan memulai klaster DB Aurora
- Pembatasan untuk menghentikan dan memulai klaster DB Aurora
- Menghentikan klaster DB Aurora
- Operasi yang mungkin dilakukan saat klaster DB Aurora dihentikan
- Memulai klaster DB Aurora

Ikhtisar menghentikan dan memulai klaster DB Aurora

Selama periode saat klaster Aurora tidak diperlukan, Anda dapat menghentikan semua instans dalam klaster itu sekaligus. Anda dapat memulai klaster lagi kapan pun diperlukan. Memulai dan menghentikan akan menyederhanakan proses penyiapan dan perombakan pada klaster yang digunakan untuk pengembangan, pengujian, atau aktivitas serupa yang tidak memerlukan ketersediaan yang berkelanjutan. Anda dapat melakukan semua prosedur AWS Management Console terkait hanya dengan satu tindakan, terlepas dari jumlah instans dalam klaster.

Saat klaster DB dihentikan, Anda hanya dikenakan biaya untuk penyimpanan klaster, snapshot manual, dan penyimpanan cadangan otomatis dalam periode penyimpanan yang ditentukan. Anda tidak dikenakan biaya untuk jam instans DB.

> **Important**
>
> Anda dapat menghentikan klaster DB hingga tujuh hari. Jika Anda tidak memulai klaster DB secara manual setelah tujuh hari, klaster DB Anda secara otomatis dimulai sehingga tidak tertinggal pembaruan pemeliharaan yang diperlukan.

Untuk meminimalkan biaya klaster Aurora yang bermuatan ringan, Anda dapat menghentikan klaster tersebut alih-alih menghapus semua Aurora Replicas. Untuk klaster dengan lebih dari satu atau dua instans, menghapus dan membuat ulang instans DB secara sering hanya praktis jika menggunakan

AWS CLI atau Amazon RDS API. Rangkaian operasi tersebut juga sulit dilakukan dalam urutan yang benar, misalnya untuk menghapus semua Aurora Replicas sebelum menghapus instans utama untuk menghindari pengaktifan mekanisme failover.

Jangan gunakan fungsi memulai dan menghentikan jika Anda perlu terus menjalankan klaster DB, tetapi klaster memiliki lebih banyak kapasitas dari yang Anda butuhkan. Jika klaster Anda terlalu mahal atau tidak terlalu sibuk, hapus satu instans DB atau lebih atau ubah semua instans DB Anda menjadi kelas instans kecil. Anda tidak dapat menghentikan instans DB Aurora secara terpisah.

Pembatasan untuk menghentikan dan memulai klaster DB Aurora

Beberapa klaster Aurora tidak dapat dihentikan dan dimulai:

- Anda tidak dapat menghentikan dan memulai klaster yang merupakan bagian dari basis data global Aurora.
- Anda tidak dapat menghentikan dan memulai klaster yang memiliki replika baca lintas-Wilayah.
- Anda tidak dapat menghentikan dan memulai klaster yang merupakan bagian dari deployment blue/green.
- Untuk klaster yang menggunakan fitur kueri paralel Aurora, versi MySQL Aurora minimum adalah 2.09.0.
- Anda tidak dapat menghentikan dan memulai klaster Aurora Serverless v1. Dengan Aurora Serverless v2, Anda tidak dapat menghentikan dan memulai klaster.

Jika klaster yang sudah ada tidak dapat dihentikan dan dimulai, tindakan Hentikan tidak akan tersedia dari menu Tindakan pada halaman Basis data atau halaman detail.

Menghentikan klaster DB Aurora

Untuk menggunakan klaster DB Aurora atau melakukan administrasi, selalu mulai dengan klaster DB Aurora yang berjalan, kemudian hentikan, lalu mulai lagi klaster tersebut. Saat klaster dihentikan, Anda akan dikenakan biaya untuk penyimpanan klaster, snapshot manual, dan penyimpanan cadangan otomatis dalam periode penyimpanan yang ditentukan, tetapi tidak untuk jam instans DB.

Operasi penghentian akan menghentikan instans Aurora Replica terlebih dahulu, lalu instans utama, untuk menghindari pengaktifan mekanisme failover.

Anda tidak dapat menghentikan klaster DB yang bertindak sebagai target replikasi untuk data dari klaster DB lain, atau bertindak sebagai master replikasi dan mengirim data ke klaster lainnya.

Anda tidak dapat menghentikan beberapa jenis klaster khusus tertentu. Saat ini, Anda tidak dapat menghentikan klaster yang merupakan bagian dari basis data global Aurora.

Konsol

Untuk menghentikan klaster Aurora

1. Masuk ke AWS Management Console, lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih salah satu klaster. Anda dapat melakukan operasi penghentian dari halaman ini, atau membuka halaman detail untuk klaster DB yang ingin Anda hentikan.
3. Untuk Tindakan, pilih Hentikan sementara.

   Jika klaster DB tidak dapat dihentikan dan dimulai, tindakan Hentikan sementara tidak akan tersedia dari menu Tindakan pada halaman Basis data atau halaman detail. Untuk jenis klaster yang tidak dapat Anda mulai dan hentikan, lihat Pembatasan untuk menghentikan dan memulai klaster DB Aurora.
4. Di jendela Hentikan sementara klaster DB, pilih pengakuan bahwa klaster DB akan dimulai ulang secara otomatis setelah 7 hari.
5. Pilih Hentikan sementara untuk menghentikan klaster DB, atau pilih Batalkan untuk membatalkan operasi.

AWS CLI

Untuk menghentikan instance DB dengan menggunakanAWS CLI, panggil stop-db-clusterperintah dengan parameter berikut:

- `--db-cluster-identifier` – nama klaster Aurora.

Example

```
aws rds stop-db-cluster --db-cluster-identifier mydbcluster
```

API RDS

Untuk menghentikan instans DB menggunakan API Amazon RDS, panggil operasi StopDBCluster dengan parameter berikut ini:

- `DBClusterIdentifier` – nama klaster Aurora.

Operasi yang mungkin dilakukan saat klaster DB Aurora dihentikan

Sementara klaster Aurora dihentikan, Anda dapat melakukan point-in-time pemulihan ke titik mana pun dalam jendela retensi cadangan otomatis yang Anda tentukan. Untuk detail tentang melakukan point-in-time pemulihan, lihatMemulihkan data.

Anda tidak dapat memodifikasi konfigurasi klaster DB Aurora, atau salah satu instans DB-nya saat klaster dihentikan. Anda juga tidak dapat menambahkan atau menghapus instans DB dari klaster, atau menghapus klaster tersebut jika masih terdapat instans DB terkait. Anda harus memulai klaster sebelum melakukan tindakan administratif tersebut.

Menghentikan klaster DB akan menghapus tindakan yang tertunda, kecuali untuk grup parameter klaster DB atau untuk grup parameter DB dari instans klaster DB.

Aurora menerapkan pemeliharaan terjadwal apa pun pada klaster yang dihentikan setelah dimulai lagi. Perlu dicatat bahwa setelah tujuh hari, Aurora secara otomatis memulai klaster yang dihentikan sehingga klaster tersebut tidak terlalu jauh tertinggal dalam status pemeliharaannya.

Aurora juga tidak melakukan cadangan otomatis apa pun, karena data yang mendasari tidak berubah saat klaster dihentikan. Aurora tidak memperpanjang periode penyimpanan cadangan saat klaster dihentikan.

Memulai klaster DB Aurora

Anda selalu memulai klaster DB Aurora diawali dengan klaster Aurora yang sudah berada dalam status terhenti. Saat Anda memulai klaster tersebut, semua instans DB menjadi tersedia lagi. Klaster tersebut menjaga pengaturan konfigurasi seperti titik akhir, grup parameter dan grup keamanan VPC.

Memulai ulang klaster DB biasanya memerlukan waktu beberapa menit.

Konsol

Untuk memulai klaster Aurora

1. Masuk ke AWS Management Console, lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih salah satu klaster. Anda dapat melakukan operasi mulai dari halaman ini, atau membuka halaman detail untuk klaster DB yang ingin Anda mulai.

3. Untuk Tindakan, pilih Mulai.

AWS CLI

Untuk memulai cluster DB dengan menggunakanAWS CLI, panggil start-db-clusterperintah dengan parameter berikut:

- `--db-cluster-identifier` – nama klaster Aurora. Nama ini adalah pengidentifikasi klaster tertentu yang Anda pilih saat membuat klaster, atau pengidentifikasi instans DB yang Anda pilih dengan `-cluster` yang ditambahkan ke bagian akhir.

Example

```
aws rds start-db-cluster --db-cluster-identifier mydbcluster
```

API RDS

Untuk memulai klaster DB Aurora menggunakan API Amazon RDS, panggil operasi StartDBCluster dengan parameter berikut ini:

- `DBCluster` – nama klaster Aurora. Nama ini adalah pengidentifikasi klaster tertentu yang Anda pilih saat membuat klaster, atau pengidentifikasi instans DB yang Anda pilih dengan `-cluster` yang ditambahkan ke bagian akhir.

Secara otomatis menghubungkan sumber daya komputasi AWS dan klaster DB Aurora

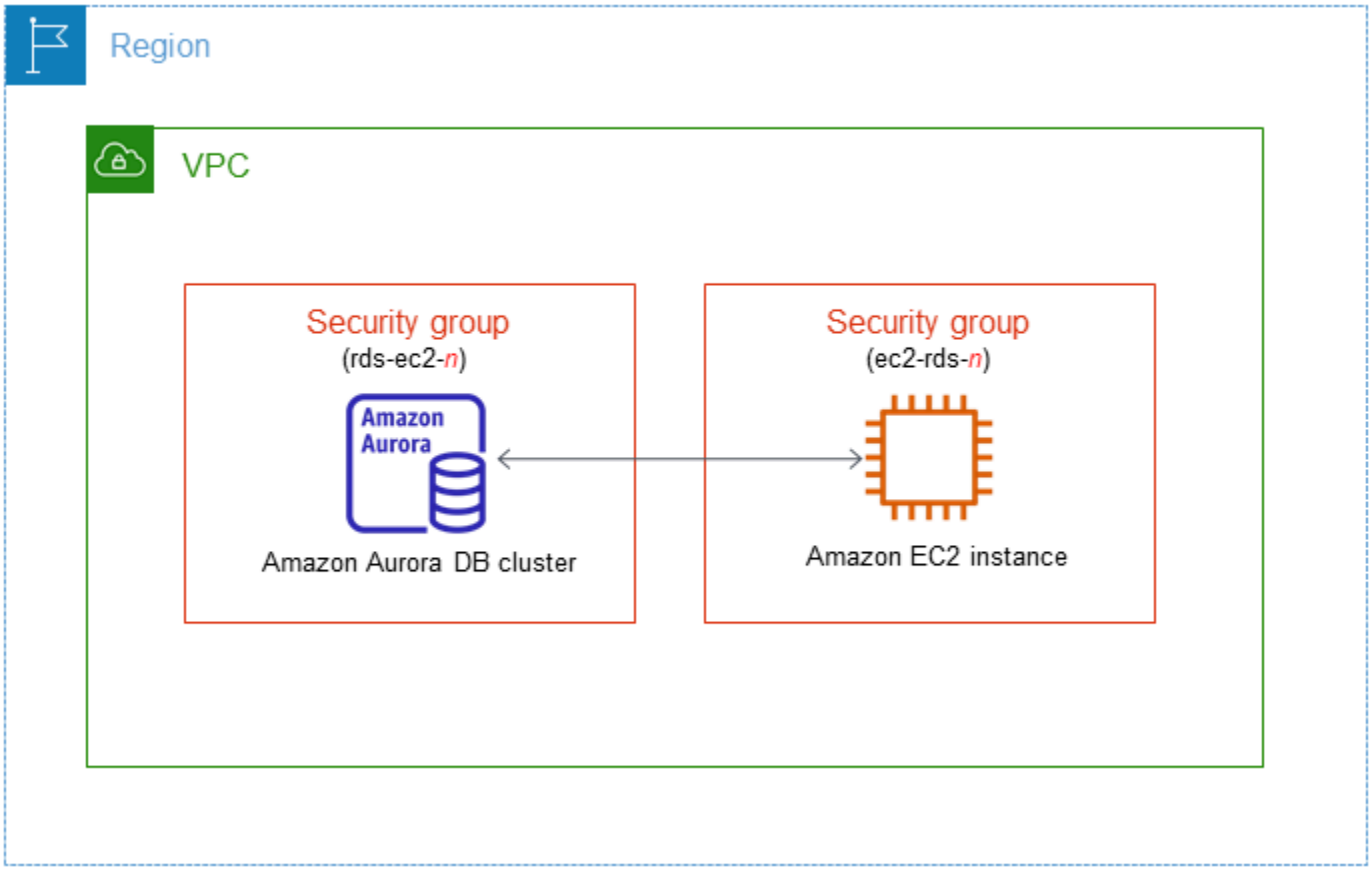
Anda dapat menghubungkan klaster DB Aurora dan AWS sumber daya komputasi seperti instans Amazon Elastic Compute Cloud (Amazon EC2) dan fungsi AWS Lambda secara otomatis.

Topik

- Menghubungkan secara otomatis instans EC2 dan klaster basis data Aurora
- Menghubungkan secara otomatis fungsi Lambda dan klaster DB Aurora

Menghubungkan secara otomatis instans EC2 dan klaster basis data Aurora

Anda dapat menggunakan konsol Amazon RDS untuk menyederhanakan penyiapan koneksi antara instans Amazon Elastic Compute Cloud (Amazon EC2) dan instans klaster basis data Aurora. Sering kali, klaster basis data Anda berada di subnet privat dan instans EC2 Anda berada dalam VPC di subnet publik. Anda dapat menggunakan klien SQL pada instans EC2 Anda untuk menghubungi klaster basis data Anda. Instans EC2 juga dapat menjalankan server atau aplikasi web yang mengakses klaster basis data privat Anda.



Jika Anda ingin menghubungkan dengan instans EC2 yang tidak berada di VPC yang sama dengan klaster DB Aurora, lihat skenario di Skenario untuk mengakses klaster DB di VPC.

Topik

- Ikhtisar konektivitas otomatis dengan instans EC2
- Menghubungkan secara otomatis instans EC2 dan klaster basis data Aurora
- Melihat sumber daya komputasi terhubung
- Menghubungi instans basis data yang menjalankan mesin basis data tertentu

Ikhtisar konektivitas otomatis dengan instans EC2

Saat Anda menyiapkan koneksi antara instans EC2 dan klaster basis data Aurora, Amazon RDS mengonfigurasikan secara otomatis grup keamanan VPC untuk instans EC2 Anda dan untuk klaster basis data Anda.

Berikut adalah persyaratan untuk menghubungkan instans EC2 dengan klaster basis data Aurora:

- Instans EC2 harus ada di VPC yang sama dengan klaster basis data.

  Jika tidak ada instans EC2 di VPC yang sama, maka konsol menyediakan tautan untuk membuatnya.
- Saat ini, klaster basis data tidak dapat menjadi sebuah klaster basis data Aurora Serverless atau bagian dari sebuah basis data global Aurora.
- Pengguna yang menyiapkan konektivitas harus memiliki izin untuk melakukan operasi-operasi Amazon EC2 berikut:
  - `ec2:AuthorizeSecurityGroupEgress`
  - `ec2:AuthorizeSecurityGroupIngress`
  - `ec2:CreateSecurityGroup`
  - `ec2:DescribeInstances`
  - `ec2:DescribeNetworkInterfaces`
  - `ec2:DescribeSecurityGroups`
  - `ec2:ModifyNetworkInterfaceAttribute`
  - `ec2:RevokeSecurityGroupEgress`

Jika instans basis data dan instans EC2 berada di Zona Ketersediaan yang berbeda, akun Anda mungkin dikenakan biaya lintas Zona Ketersediaan.

Saat Anda menyiapkan koneksi dengan instans EC2, Amazon RDS bertindak sesuai dengan konfigurasi grup keamanan saat ini yang terkait dengan klaster basis data dan instans EC2, seperti dijelaskan dalam tabel berikut.

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan EC2 saat ini	Tindakan RDS
Ada satu atau beberapa grup keamanan yang terkait dengan klaster basis data dengan nama yang cocok dengan pola `rds-ec2-`*n* (dengan *n* berupa angka). Grup keamanan yang cocok dengan pola belum diubah.	Ada satu atau beberapa grup keamanan yang terkait dengan instans EC2 dengan nama yang cocok dengan pola `ec2-rds-`*n* (dengan *n* berupa angka). Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini	RDS tidak mengambil tindakan. Koneksi sudah dikonfigu rasikan secara otomatis antara instans EC2 dan klaster basis data. Karena koneksi sudah ada antara instans EC2

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan EC2 saat ini	Tindakan RDS
Grup keamanan ini memiliki hanya satu aturan masuk dengan grup keamanan VPC instans EC2 sebagai sumbernya.	memiliki hanya satu aturan keluar dengan grup keamanan VPC klaster basis data sebagai sumbernya.	dan basis data RDS, grup keamanan tidak diubah.

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan EC2 saat ini	Tindakan RDS
Salah satu syarat berikut dipenuhi: • Tidak ada grup keamanan yang terkait dengan klaster basis data dengan nama yang cocok dengan pola `rds-ec2-`*n*. • Ada satu atau beberapa grup keamanan yang terkait dengan klaster basis data dengan nama yang cocok dengan pola `rds-ec2-`*n*. Namun, Amazon RDS tidak dapat menggunakan satu pun grup keamanan ini untuk koneksi dengan instans EC2. Amazon RDS tidak dapat menggunakan grup keamanan yang tidak memiliki satu aturan masuk dengan grup keamanan VPC instans EC2 sebagai sumbernya. Amazon RDS juga tidak dapat menggunak an grup keamanan yang telah diubah. Contoh-co ntoh perubahan meliputi penambahan aturan atau pengubahan port aturan yang ada.	Salah satu syarat berikut dipenuhi: • Tidak ada grup keamanan yang terkait dengan instans EC2 dengan nama yang cocok dengan pola `ec2-rds-`*n*. • Ada satu atau beberapa grup keamanan yang terkait dengan instans EC2 dengan nama yang cocok dengan pola `ec2-rds-`*n*. Namun, Amazon RDS tidak dapat menggunakan satu pun grup keamanan ini untuk koneksi dengan klaster basis data. Amazon RDS tidak dapat menggunakan grup keamanan yang tidak memiliki satu aturan keluar dengan grup keamanan VPC klaster basis data sebagai sumbernya. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah diubah.	RDS action: create new security groups

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan EC2 saat ini	Tindakan RDS
Ada satu atau beberapa grup keamanan yang terkait dengan klaster basis data dengan nama yang cocok dengan pola `rds-ec2-`*n*. Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini memiliki hanya satu aturan masuk dengan grup keamanan VPC instans EC2 sebagai sumbernya.	Ada satu atau beberapa grup keamanan yang terkait dengan instans EC2 dengan nama yang cocok dengan pola `ec2-rds-`*n*. Namun, Amazon RDS tidak dapat menggunakan satu pun grup keamanan ini untuk koneksi dengan klaster basis data. Amazon RDS tidak dapat menggunakan grup keamanan yang tidak memiliki satu aturan keluar dengan grup keamanan VPC klaster basis data sebagai sumbernya. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah diubah.	RDS action: create new security groups
Ada satu atau beberapa grup keamanan yang terkait dengan klaster basis data dengan nama yang cocok dengan pola `rds-ec2-`*n*. Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini memiliki hanya satu aturan masuk dengan grup keamanan VPC instans EC2 sebagai sumbernya.	Ada grup keamanan EC2 yang valid untuk koneksi, tetapi tidak terkait dengan instans EC2. Grup keamanan ini memiliki nama yang cocok dengan pola `ec2-rds-`*n*. Grup itu belum diubah. Grup ini memiliki hanya satu aturan keluar dengan grup keamanan VPC klaster basis data sebagai sumbernya.	RDS action: associate EC2 security group

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan EC2 saat ini	Tindakan RDS
Salah satu syarat berikut dipenuhi: • Tidak ada grup keamanan yang terkait dengan klaster basis data dengan nama yang cocok dengan pola `rds-ec2-`*n*. • Ada satu atau beberapa grup keamanan yang terkait dengan klaster basis data dengan nama yang cocok dengan pola `rds-ec2-`*n*. Namun, Amazon RDS tidak dapat menggunakan satu pun grup keamanan ini untuk koneksi dengan instans EC2. Amazon RDS tidak dapat menggunakan grup keamanan yang tidak memiliki satu aturan masuk dengan grup keamanan VPC instans EC2 sebagai sumbernya. Amazon RDS juga tidak dapat menggunak an grup keamanan yang telah diubah.	Ada satu atau beberapa grup keamanan yang terkait dengan instans EC2 dengan nama yang cocok dengan pola `ec2-rds-`*n*. Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini memiliki hanya satu aturan keluar dengan grup keamanan VPC klaster basis data sebagai sumbernya .	RDS action: create new security groups

Tindakan RDS : membuat grup keamanan baru

Amazon RDS melakukan tindakan-tindakan berikut:

- Membuat grup keamanan baru yang cocok dengan pola `rds-ec2-`*n*. Grup keamanan ini memiliki aturan masuk dengan grup keamanan VPC instans EC2 sebagai sumbernya. Grup keamanan ini dikaitkan dengan klaster basis data dan memungkinkan instans EC2 mengakses klaster basis data.
- Membuat grup keamanan baru yang cocok dengan pola `ec2-rds-`*n*. Grup keamanan ini memiliki aturan keluar dengan grup keamanan VPC dari cluster DB database sebagai target. Grup keamanan ini dikaitkan dengan instans EC2 dan memungkinkan instans EC2 mengirim lalu lintas ke klaster basis data.

Tindakan RDS : mengaitkan grup keamanan EC2

Amazon RDS mengaitkan grup keamanan EC2 yang valid dan sudah ada dengan instans EC2. Grup keamanan ini memungkinkan instans EC2 mengirim lalu lintas ke klaster basis data.

Menghubungkan secara otomatis instans EC2 dan klaster basis data Aurora

Sebelum menyiapkan koneksi antara instans EC2 dan klaster basis data Aurora, pastikan untuk memenuhi persyaratan yang dijelaskan di Ikhtisar konektivitas otomatis dengan instans EC2.

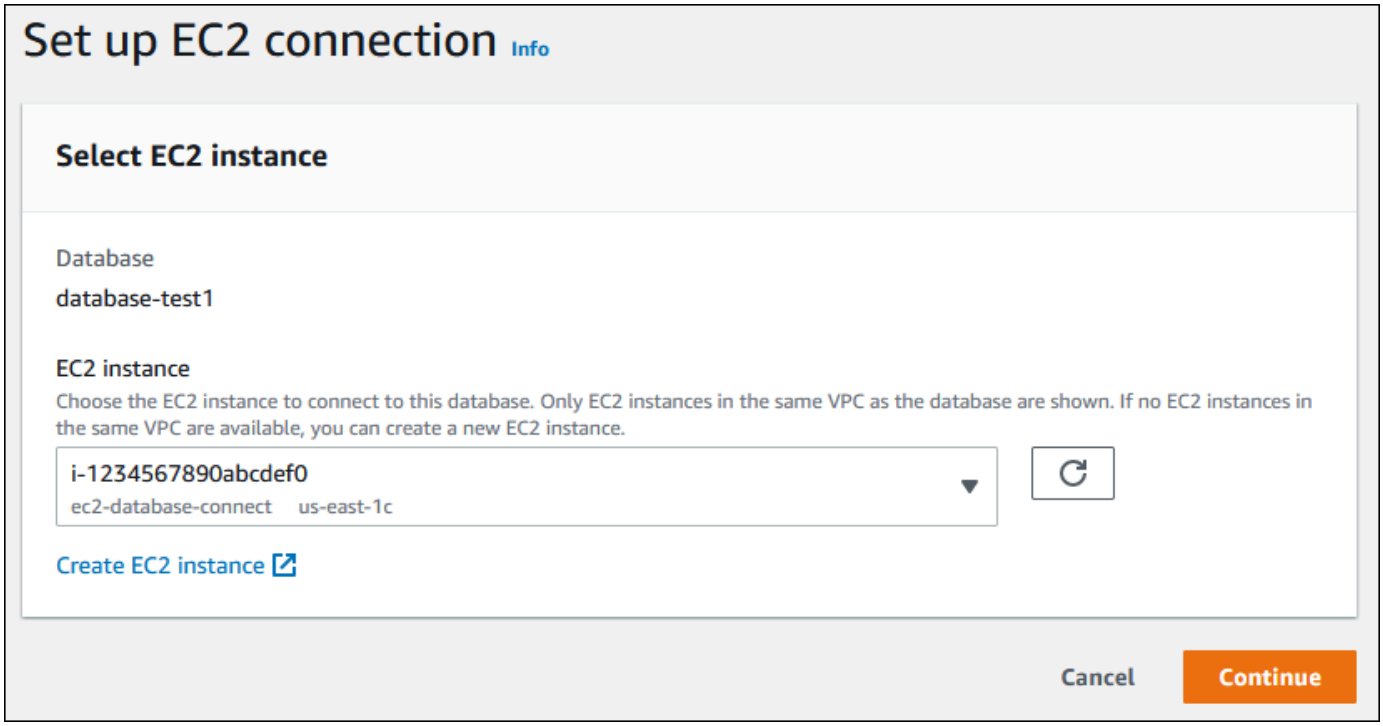
Jika Anda membuat perubahan pada grup keamanan setelah mengonfigurasikan konektivitas, perubahan itu dapat memengaruhi koneksi antara instans EC2 dan klaster basis data Aurora.

> **Note**
>
> Anda hanya dapat menyiapkan koneksi antara instans EC2 dan klaster DB Aurora secara otomatis dengan menggunakan AWS Management Console. Anda tidak dapat mengatur koneksi secara otomatis dengan AWS CLI atau RDS API.

Untuk menghubungkan secara otomatis instans EC2 dan klaster basis data Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, lalu pilih klaster basis data.
3. Untuk Tindakan, pilih Siapkan koneksi EC2.

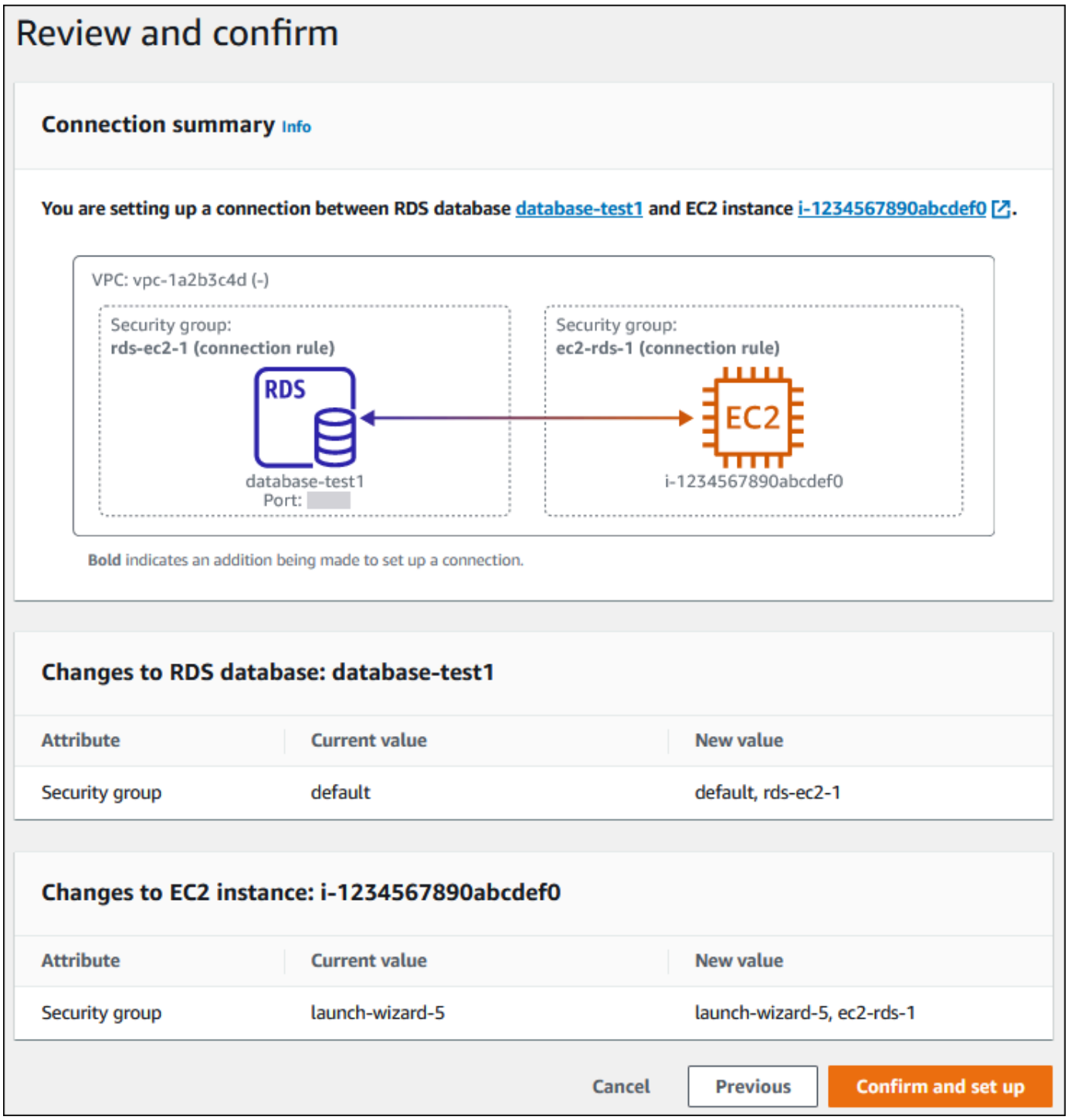
   Halaman Siapkan koneksi EC2 muncul.
4. Pada halaman Siapkan koneksi EC2, pilih instans EC2.



Jika tidak ada instans EC2 di VPC yang sama, pilih Buat instans EC2 untuk membuatnya. Dalam hal ini, pastikan bahwa instans EC2 baru berada di VPC yang sama dengan klaster basis data.

5. Pilih Lanjutkan.

   Halaman Tinjau dan tegaskan muncul.



6. Pada halaman Tinjau dan tegaskan, tinjau perubahan yang akan dilakukan RDS untuk menyiapkan konektivitas dengan instans EC2.

   Jika perubahan sudah benar, pilih Tegaskan dan siapkan.

   Jika perubahan masih salah, pilih Sebelumnya atau Batalkan.

Melihat sumber daya komputasi terhubung

Anda dapat menggunakan AWS Management Console untuk melihat sumber daya komputasi yang terhubung ke database DB cluster. Sumber daya yang ditampilkan meliputi koneksi sumber daya komputasi yang disiapkan secara otomatis. Anda dapat menyiapkan secara otomatis konektivitas dengan sumber daya komputasi dengan cara berikut:

- Anda dapat memilih sumber daya komputasi saat membuat basis data.

  Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.
- Anda dapat menyiapkan konektivitas antara basis data yang ada dan sumber daya komputasi.

  Untuk informasi selengkapnya, lihat Menghubungkan secara otomatis instans EC2 dan klaster basis data Aurora.

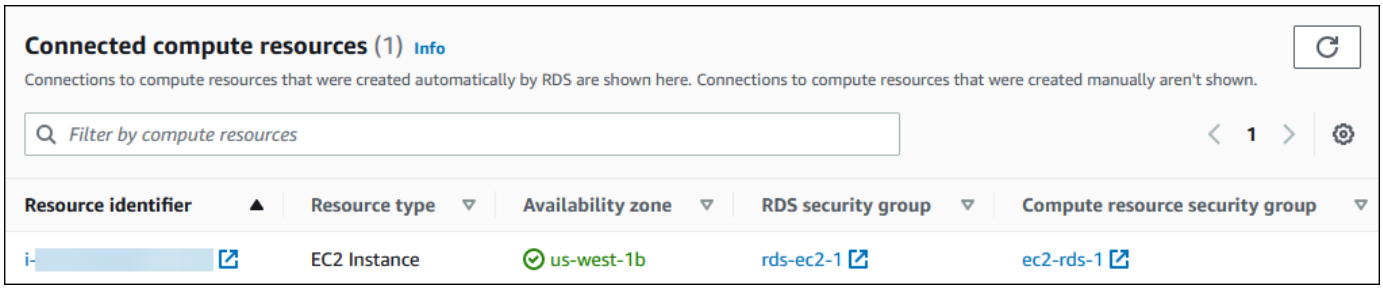
Sumber daya komputasi yang tercantum tidak menyertakan sumber daya yang dihubungkan secara manual dengan basis data. Misalnya, Anda dapat mengizinkan sumber daya komputasi untuk mengakses basis data secara manual dengan menambahkan aturan ke grup keamanan VPC yang terkait dengan basis data.

Agar sumber daya komputasi tercantum, syarat-syarat berikut harus dipenuhi:

- Nama grup keamanan yang terkait dengan sumber daya komputasi cocok dengan pola `ec2-rds-`*n* (dengan *n* berupa angka).
- Grup keamanan yang terkait dengan sumber daya komputasi memiliki aturan keluar dengan rentang port diatur ke port yang digunakan klaster basis data.
- Grup keamanan yang terkait dengan sumber daya komputasi memiliki aturan keluar dengan sumber yang diatur ke grup keamanan yang terkait dengan klaster basis data.
- Nama grup keamanan yang terkait dengan klaster basis data cocok dengan pola `rds-ec2-`*n* (dengan *n* berupa angka).
- Grup keamanan yang terkait dengan klaster basis data memiliki aturan masuk dengan rentang port yang diatur ke port yang digunakan klaster basis data.
- Grup keamanan yang terkait dengan klaster basis data memiliki aturan masuk dengan sumber yang diatur ke grup keamanan yang terkait dengan sumber daya komputasi.

Untuk melihat sumber daya komputasi yang menghubungi klaster basis data Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, lalu pilih nama klaster basis data.
3. Pada tab Konektivitas dan keamanan, lihat sumber daya komputasi di Sumber daya komputasi terhubung.



Menghubungi instans basis data yang menjalankan mesin basis data tertentu

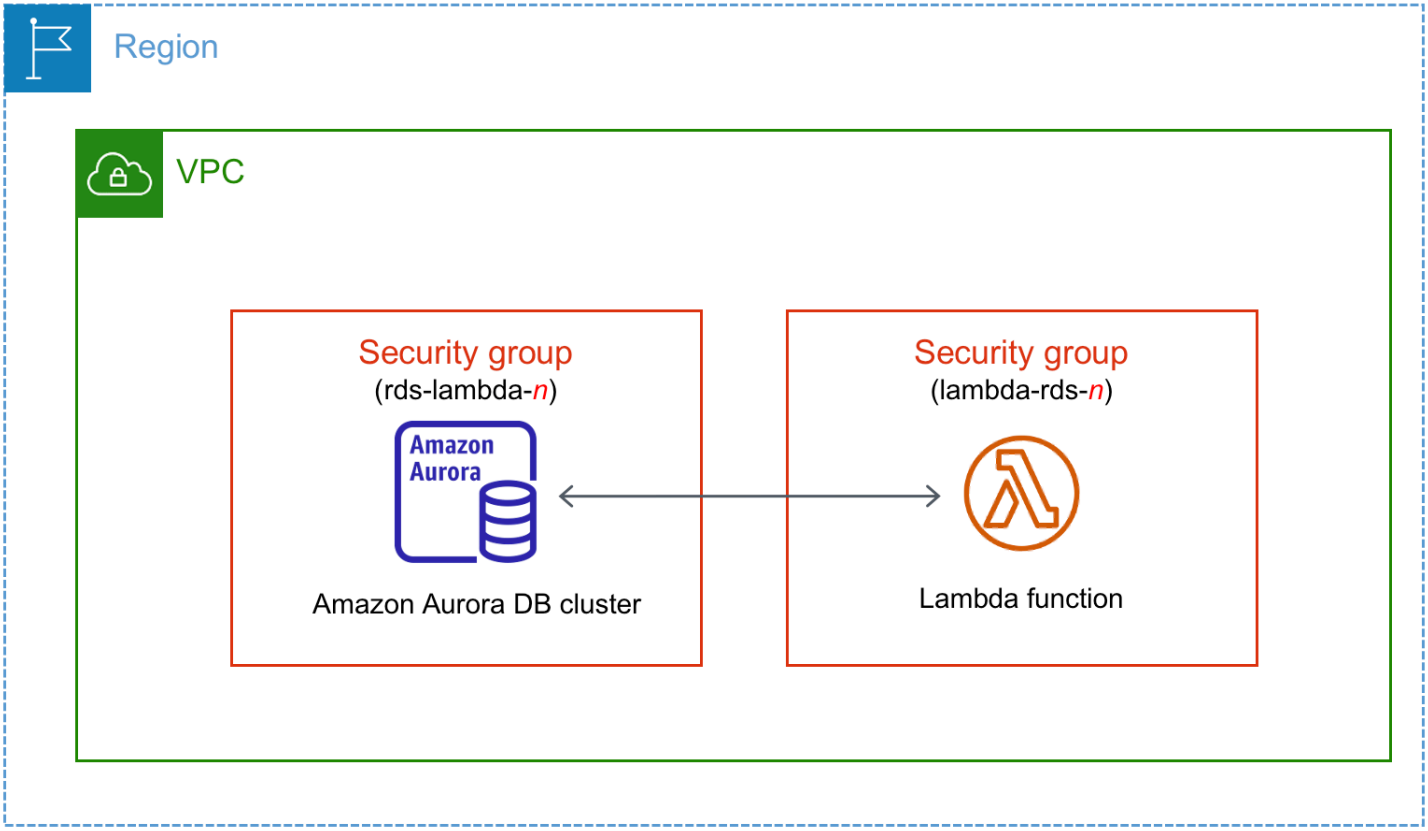
Untuk informasi tentang cara menghubungi instans basis data yang menjalankan mesin basis data tertentu, ikuti petunjuk untuk mesin basis data Anda:

- Menghubungkan ke klaster DB Amazon Aurora MySQL
- Menghubungkan ke klaster DB Amazon Aurora PostgreSQL.

Menghubungkan secara otomatis fungsi Lambda dan klaster DB Aurora

Anda dapat menggunakan konsol Amazon RDS untuk menyederhanakan penyiapan koneksi antara fungsi Lambda dan klaster DB Aurora. Biasanya, klaster DB Anda berada di subnet privat dalam VPC. Fungsi Lambda dapat digunakan oleh aplikasi untuk mengakses klaster DB privat Anda.

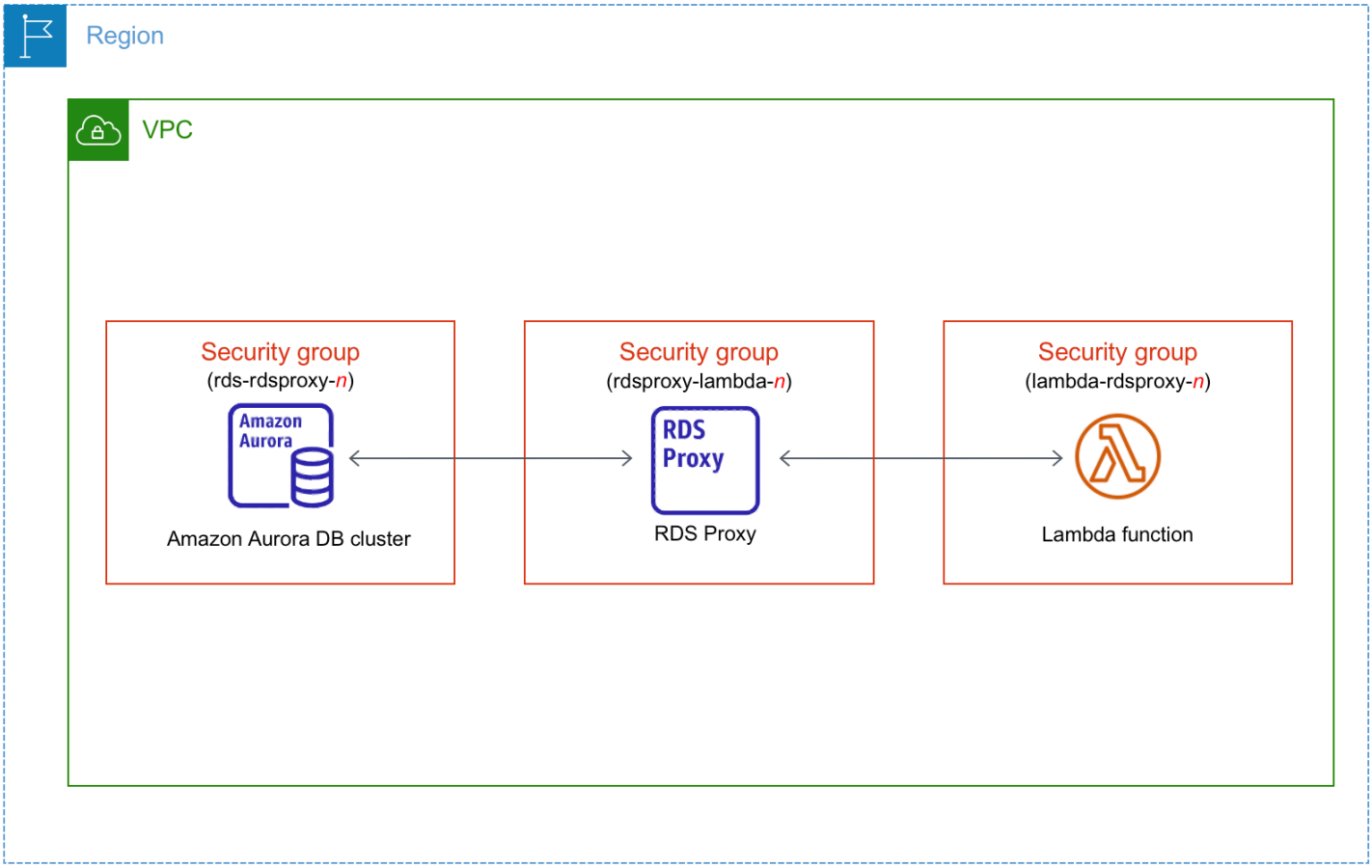
Gambar berikut menunjukkan koneksi langsung antara klaster DB dan fungsi Lambda.



Anda dapat menyiapkan koneksi antara fungsi Lambda dan klaster DB Anda melalui Proksi RDS untuk meningkatkan kinerja dan ketangguhan basis data Anda. Biasanya, fungsi Lambda membuat koneksi basis data yang singkat tetapi sering yang mendapat manfaat dari penghimpunan koneksi yang ditawarkan Proksi RDS. Anda dapat memanfaatkan autentikasi AWS Identity and Access Management (IAM) apa pun yang Anda miliki untuk fungsi Lambda, alih-alih mengelola kredensial basis data dalam kode aplikasi Lambda. Untuk informasi selengkapnya, lihat Menggunakan Proksi Amazon RDS untuk Aurora.

Saat Anda menggunakan konsol untuk terhubung dengan proksi yang ada, Amazon RDS memperbarui grup keamanan proksi untuk mengizinkan koneksi dari klaster DB dan fungsi Lambda Anda.

Anda juga dapat membuat proksi baru dari halaman konsol yang sama. Saat Anda membuat proksi di konsol, untuk mengakses klaster DB, Anda harus memasukkan kredensial basis data Anda atau memilih rahasia AWS Secrets Manager.



Topik

- Ikhtisar konektivitas otomatis dengan fungsi Lambda
- Menghubungkan secara otomatis fungsi Lambda dan klaster DB Aurora
- Melihat sumber daya komputasi terhubung

Ikhtisar konektivitas otomatis dengan fungsi Lambda

Berikut adalah persyaratan untuk menghubungkan fungsi Lambda dengan klaster DB Aurora:

- Fungsi Lambda harus ada di VPC yang sama dengan klaster DB.
- Saat ini, klaster DB tidak dapat berupa klaster DB Aurora Serverless atau bagian dari basis data global Aurora.
- Pengguna yang menyiapkan konektivitas harus memiliki izin untuk melakukan operasi-operasi Amazon RDS, Amazon EC2, Lambda, Secrets Manager, dan IAM berikut:
  - Amazon RDS

    - rds:CreateDBProxies
    - rds:DescribeDBClusters
    - rds:DescribeDBProxies
    - rds:ModifyDBCluster
    - rds:ModifyDBProxy
    - rds:RegisterProxyTargets
- Amazon EC2
    - ec2:AuthorizeSecurityGroupEgress
    - ec2:AuthorizeSecurityGroupIngress
    - ec2:CreateSecurityGroup
    - ec2:DeleteSecurityGroup
    - ec2:DescribeSecurityGroups
    - ec2:RevokeSecurityGroupEgress
    - ec2:RevokeSecurityGroupIngress
- Lambda
    - lambda:CreateFunctions
    - lambda:ListFunctions
    - lambda:UpdateFunctionConfiguration
- Secrets Manager
    - secretsmanager:CreateSecret
    - secretsmanager:DescribeSecret
- IAM
    - iam:AttachPolicy
    - iam:CreateRole
    - iam:CreatePolicy
- AWS KMS
    - kms:describeKey

> **Note**
>
> Jika klaster DB dan fungsi Lambda berada di Zona Ketersediaan yang berbeda, akun Anda mungkin dikenakan biaya lintas Zona Ketersediaan.

Saat Anda menyiapkan koneksi antara fungsi Lambda dan klaster DB Aurora, Amazon RDS mengonfigurasikan grup keamanan VPC untuk fungsi Anda dan untuk klaster DB Anda. Jika Anda menggunakan Proksi RDS, maka Amazon RDS juga mengonfigurasikan grup keamanan VPC untuk proksi tersebut. Amazon RDS bertindak sesuai dengan konfigurasi grup keamanan saat ini yang terkait dengan klaster DB, fungsi Lambda, dan proksi, seperti dijelaskan dalam tabel berikut.

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan Lambda saat ini	Konfigurasi grup keamanan proksi saat ini	Tindakan RDS
Ada satu atau beberapa grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika proksi sudah terhubung dengan klaster DB Anda, RDS memeriksa apakah `TargetHealth` proksi terkait adalah `AVAILABLE` . Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini memiliki hanya satu	Ada satu atau beberapa grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n* (dengan *n* berupa angka). Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini hanya memiliki satu aturan keluar dengan grup keamanan VPC klaster DB atau proksi sebagai tujuannya.	Ada satu atau beberapa grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan pola `rdsproxy-lambda-` *n* (dengan *n* berupa angka). Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini memiliki aturan masuk dan aturan keluar dengan grup keamanan VPC fungsi Lambda dan klaster DB.	Amazon RDS tidak mengambil tindakan. Koneksi sudah dikonfigurasikan secara otomatis antara fungsi Lambda, proksi (opsional), dan klaster DB. Karena koneksi sudah ada antara fungsi, proksi, dan basis data, grup keamanan tidak diubah.

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan Lambda saat ini	Konfigurasi grup keamanan proksi saat ini	Tindakan RDS
aturan masuk dengan grup keamanan VPC dari fungsi Lambda atau proksi sebagai sumbernya.			

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan Lambda saat ini	Konfigurasi grup keamanan proksi saat ini	Tindakan RDS
Salah satu syarat berikut berlaku: • Tidak ada grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE`. • Ada satu atau beberapa grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE`. Namun, tidak satu pun grup keamanan ini dapat digunakan untuk koneksi dengan fungsi Lambda.	Salah satu syarat berikut berlaku: • Tidak ada grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*. • Ada satu atau beberapa grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*. Namun, Amazon RDS tidak dapat menggunakan satu pun grup keamanan ini untuk koneksi dengan klaster DB. Amazon RDS tidak dapat menggunakan grup keamanan yang	Salah satu syarat berikut berlaku: • Tidak ada grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan pola `rdsproxy-lambda-` *n*. • Ada satu atau beberapa grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan `rdsproxy-lambda-` *n*. Namun, Amazon RDS tidak dapat menggunakan satu pun grup keamanan ini untuk koneksi dengan klaster DB atau fungsi Lambda. Amazon RDS tidak dapat menggunak an grup keamanan	RDS action: create new security groups

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan Lambda saat ini	Konfigurasi grup keamanan proksi saat ini	Tindakan RDS
Amazon RDS tidak dapat menggunakan grup keamanan yang tidak memiliki satu aturan masuk dengan grup keamanan VPC proksi atau fungsi Lambda sebagai sumbernya. Amazon RDS juga tidak dapat menggunak an grup keamanan yang telah diubah. Contoh-contoh perubahan meliputi penambahan aturan atau pengubahan port aturan yang ada.	tidak memiliki satu aturan keluar dengan grup keamanan VPC klaster DB atau proksi sebagai tujuannya. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah diubah.	yang tidak memiliki aturan masuk dan keluar dengan grup keamanan VPC klaster DB dan fungsi Lambda. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah diubah.	

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan Lambda saat ini	Konfigurasi grup keamanan proksi saat ini	Tindakan RDS
Ada satu atau beberapa grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE` . Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini memiliki hanya satu aturan masuk dengan grup keamanan VPC dari fungsi Lambda atau proksi sebagai sumbernya.	Ada satu atau beberapa grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*. Namun, Amazon RDS tidak dapat menggunakan satu pun grup keamanan ini untuk koneksi dengan klaster DB. Amazon RDS tidak dapat menggunakan grup keamanan yang tidak memiliki satu aturan keluar dengan grup keamanan VPC klaster DB atau proksi sebagai tujuannya. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah diubah.	Ada satu atau beberapa grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan pola `rdsproxy-lambda-` *n*. Namun, Amazon RDS tidak dapat menggunakan satu pun grup keamanan ini untuk koneksi dengan klaster DB atau fungsi Lambda. Amazon RDS tidak dapat menggunakan grup keamanan yang tidak memiliki aturan masuk dan keluar dengan grup keamanan VPC klaster DB dan fungsi Lambda. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah diubah.	RDS action: create new security groups

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan Lambda saat ini	Konfigurasi grup keamanan proksi saat ini	Tindakan RDS
Ada satu atau beberapa grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE` . Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini memiliki hanya satu aturan masuk dengan grup keamanan VPC dari fungsi Lambda atau proksi sebagai sumbernya.	Ada grup keamanan Lambda yang valid untuk koneksi, tetapi tidak dikaitkan dengan fungsi Lambda. Grup keamanan ini memiliki nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*. Grup tersebut belum diubah. Grup hanya memiliki satu aturan keluar dengan grup keamanan VPC klaster DB atau proksi sebagai tujuannya.	Ada grup keamanan proksi yang valid untuk koneksi, tetapi tidak dikaitkan dengan proksi. Grup keamanan ini memiliki nama yang cocok dengan pola `rdsproxy-lambda-` *n*. Grup tersebut belum diubah. Grup ini memiliki aturan masuk dan aturan keluar dengan grup keamanan VPC klaster DB dan fungsi Lambda.	RDS action: associate Lambda security group

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan Lambda saat ini	Konfigurasi grup keamanan proksi saat ini	Tindakan RDS
Salah satu syarat berikut berlaku: • Tidak ada grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE`. • Ada satu atau beberapa grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE`. Namun, Amazon RDS tidak dapat menggunakan satu pun grup keamanan ini untuk koneksi dengan proksi atau fungsi Lambda.	Ada satu atau beberapa grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*. Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini hanya memiliki satu aturan keluar dengan grup keamanan VPC klaster DB atau proksi sebagai tujuannya.	Ada satu atau beberapa grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan pola `rdsproxy-lambda-` *n*. Grup keamanan yang cocok dengan pola belum diubah. Grup keamanan ini memiliki aturan masuk dan aturan keluar dengan grup keamanan VPC fungsi Lambda dan klaster DB.	RDS action: create new security groups

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan Lambda saat ini	Konfigurasi grup keamanan proksi saat ini	Tindakan RDS
Amazon RDS tidak dapat menggunakan grup keamanan yang tidak memiliki satu aturan masuk dengan grup keamanan VPC proksi atau fungsi Lambda sebagai sumbernya. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah diubah.			

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan Lambda saat ini	Konfigurasi grup keamanan proksi saat ini	Tindakan RDS
Salah satu syarat berikut berlaku: • Tidak ada grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE`. • Ada satu atau beberapa grup keamanan yang terkait dengan klaster DB dengan nama yang cocok dengan pola `rds-lambda-` *n* atau jika `TargetHealth` proksi terkait adalah `AVAILABLE`. Namun, Amazon RDS tidak dapat menggunakan satu pun grup keamanan ini untuk koneksi dengan proksi atau fungsi Lambda.	Salah satu syarat berikut berlaku: • Tidak ada grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*. • Ada satu atau beberapa grup keamanan yang terkait dengan fungsi Lambda dengan nama yang cocok dengan pola `lambda-rds-` *n* atau `lambda-rdsproxy-` *n*. Namun, Amazon RDS tidak dapat menggunakan satu pun grup keamanan ini untuk koneksi dengan klaster DB. Amazon RDS tidak dapat menggunakan grup keamanan yang	Salah satu syarat berikut berlaku: • Tidak ada grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan pola `rdsproxy-lambda-` *n*. • Ada satu atau beberapa grup keamanan yang terkait dengan proksi dengan nama yang cocok dengan `rdsproxy-lambda-` *n*. Namun, Amazon RDS tidak dapat menggunakan satu pun grup keamanan ini untuk koneksi dengan klaster DB atau fungsi Lambda. Amazon RDS tidak dapat menggunak an grup keamanan	RDS action: create new security groups

Konfigurasi grup keamanan RDS saat ini	Konfigurasi grup keamanan Lambda saat ini	Konfigurasi grup keamanan proksi saat ini	Tindakan RDS
Amazon RDS tidak dapat menggunakan grup keamanan yang tidak memiliki satu aturan masuk dengan grup keamanan VPC proksi atau fungsi Lambda sebagai sumbernya. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah diubah.	tidak memiliki satu aturan keluar dengan grup keamanan VPC klaster DB atau proksi sebagai sumbernya. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah diubah.	yang tidak memiliki aturan masuk dan keluar dengan grup keamanan VPC klaster DB dan fungsi Lambda. Amazon RDS juga tidak dapat menggunakan grup keamanan yang telah diubah.	

Tindakan RDS: membuat grup keamanan baru

Amazon RDS melakukan tindakan-tindakan berikut:

- Membuat grup keamanan baru yang cocok dengan pola `rds-lambda-`*n* atau `rds-rdsproxy-`*n* (jika Anda memilih untuk menggunakan Proksi RDS). Grup keamanan ini memiliki aturan masuk dengan grup keamanan VPC fungsi Lambda atau proksi sebagai sumbernya. Grup keamanan ini dikaitkan dengan klaster DB dan memungkinkan fungsi atau proksi mengakses klaster DB.
- Membuat grup keamanan baru yang cocok dengan pola `lambda-rds-`*n* atau `lambda-rdsproxy-`*n*. Grup keamanan ini memiliki aturan keluar dengan grup keamanan VPC klaster DB atau proksi sebagai tujuannya. Grup keamanan ini dikaitkan dengan fungsi Lambda dan memungkinkan fungsi mengirim lalu lintas ke klaster DB atau mengirim lalu lintas melalui proksi.
- Membuat grup keamanan baru yang cocok dengan pola `rdsproxy-lambda-`*n*. Grup keamanan ini memiliki aturan masuk dan aturan keluar dengan grup keamanan VPC fungsi Lambda dan klaster DB.

Tindakan RDS: mengaitkan grup keamanan Lambda

Amazon RDS mengaitkan grup keamanan Lambda yang valid dan sudah ada dengan fungsi Lambda. Grup keamanan ini memungkinkan fungsi mengirim lalu lintas ke klaster DB atau mengirim lalu lintas melalui proksi.

Menghubungkan secara otomatis fungsi Lambda dan klaster DB Aurora

Anda dapat menggunakan konsol Amazon RDS untuk menghubungkan secara otomatis fungsi Lambda dengan Anda. Ini menyederhanakan proses penyiapan koneksi di antara sumber daya-sumber daya ini.

Anda juga dapat menggunakan Proksi RDS untuk menyertakan proksi dalam koneksi Anda. Fungsi Lambda membuat koneksi basis data yang singkat tetapi sering yang mendapat manfaat dari penghimpunan koneksi yang ditawarkan Proksi RDS. Anda juga dapat menggunakan autentikasi IAM apa pun yang Anda siapkan untuk fungsi Lambda, alih-alih mengelola kredensial basis data dalam kode aplikasi Lambda.

Anda dapat menghubungkan klaster DB yang ada dengan fungsi Lambda baru dan lama dengan menggunakan halaman Siapkan koneksi Lambda. Proses penyiapan menyiapkan secara otomatis grup keamanan yang Anda perlukan.

Sebelum menyiapkan koneksi antara fungsi Lambda dan klaster DB, pastikan bahwa:

- Fungsi Lambda dan klaster DB Anda berada di VPC yang sama.
- Anda memiliki izin-izin yang tepat untuk akun pengguna Anda. Lihat informasi lebih lanjut tentang persyaratan di Ikhtisar konektivitas otomatis dengan fungsi Lambda.

Jika Anda mengubah grup keamanan setelah mengonfigurasikan konektivitas, perubahan itu dapat memengaruhi koneksi antara fungsi Lambda dan klaster DB.

> **Note**
>
> Anda dapat menyiapkan secara otomatis koneksi antara klaster DB dan fungsi Lambda hanya di AWS Management Console. Untuk menghubungkan fungsi Lambda, semua instans di klaster DB harus dalam keadaan Tersedia.

Untuk menghubungkan secara otomatis fungsi Lambda dan klaster DB

<result>

Setelah memastikan penyiapan, Amazon RDS memulai proses menghubungkan fungsi Lambda, Proksi RDS (jika Anda menggunakan proksi), dan klaster DB. Konsol menampilkan kotak dialog Detail koneksi, yang mencantumkan perubahan grup keamanan yang memungkinkan koneksi di antara sumber daya-sumber daya Anda.
</result>

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, lalu pilih klaster DB yang ingin Anda hubungkan dengan fungsi Lambda.
3. Untuk Tindakan, pilih Siapkan koneksi Lambda.
4. Pada halaman Siapkan koneksi Lambda, di bawah Pilih fungsi Lambda, lakukan salah satu hal berikut:
   - Jika Anda memiliki fungsi Lambda yang ada di VPC yang sama dengan klaster DB Anda, pilih Pilih fungsi yang ada, lalu pilih fungsi itu.
   - Jika Anda tidak memiliki fungsi Lambda di VPC yang sama, pilih Buat fungsi baru, lalu masukkan Nama fungsi. Runtime default diatur ke Nodejs.18. Anda dapat mengubah setelan untuk fungsi Lambda baru di konsol Lambda setelah menyelesaikan penyiapan koneksi.
5. (Opsional) Di bawah Proksi RDS, pilih Hubungkan lewat Proksi RDS, lalu lakukan salah satu hal berikut:
   - Jika Anda sudah memiliki proksi yang ingin Anda gunakan, pilih Pilih proksi yang ada, lalu pilih proksi itu.
   - Jika Anda tidak memiliki proksi, dan Anda ingin Amazon RDS membuatnya secara otomatis untuk Anda, pilih Buat proksi baru. Lalu, untuk Kredensial basis data, lakukan salah satu langkah berikut:
     a. Pilih Nama pengguna dan kata sandi basis data, lalu masukkan Nama pengguna dan Kata sandi untuk klaster DB Anda.
     b. Pilih Rahasia Secrets Manager. Kemudian, untuk Pilih rahasia, pilih rahasia AWS Secrets Manager. Jika Anda tidak memiliki rahasia Secrets Manager, pilih Buat rahasia Secrets Manager baru untuk membuat rahasia baru. Setelah Anda membuat rahasia, untuk Pilih rahasia, pilih rahasia baru.

Setelah Anda membuat proksi baru, pilih Pilih proksi yang ada, lalu pilih proksi. Perhatikan bahwa mungkin perlu beberapa waktu sebelum proksi Anda tersedia untuk koneksi.

6. (Opsional) Perluas Ringkasan koneksi dan periksa pembaruan yang disorot untuk sumber daya Anda.
7. Pilih Siapkan.

Melihat sumber daya komputasi terhubung

Anda dapat menggunakan AWS Management Console untuk melihat fungsi Lambda yang terhubung dengan klaster DB Anda. Sumber daya yang ditampilkan meliputi koneksi sumber daya komputasi yang disiapkan secara otomatis oleh Amazon RDS.

Sumber daya komputasi tercantum yang tidak menyertakan sumber daya yang dihubungkan secara manual dengan klaster DB. Misalnya, Anda dapat mengizinkan sumber daya komputasi untuk mengakses klaster DB Anda secara manual dengan menambahkan aturan ke grup keamanan VPC yang terkait dengan basis data.

Agar konsol menampilkan suatu fungsi Lambda, kondisi-kondisi berikut harus terpenuhi:

- Nama grup keamanan yang terkait dengan sumber daya komputasi cocok dengan pola `lambda-rds-`*n* atau `lambda-rdsproxy-`*n* (dengan *n* berupa angka).
- Grup keamanan yang terkait dengan sumber daya komputasi memiliki aturan keluar dengan rentang port yang diatur ke port klaster DB atau proksi terkait. Tujuan untuk aturan keluar harus diatur ke grup keamanan yang terkait dengan klaster DB atau proksi terkait.
- Jika konfigurasi menyertakan proksi, nama grup keamanan yang dilampirkan pada proksi yang terkait dengan basis data Anda cocok dengan pola `rdsproxy-lambda-`*n* (dengan *n* berupa angka).
- Grup keamanan yang terkait dengan fungsi memiliki aturan keluar dengan port yang diatur ke port yang digunakan oleh klaster DB atau proksi terkait. Tujuan harus diatur ke grup keamanan yang terkait dengan klaster DB atau proksi terkait.

Untuk melihat sumber daya komputasi yang dihubungkan secara otomatis dengan klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

2. Di panel navigasi, pilih Basis Data, lalu pilih klaster DB.
3. Pada tab Konektivitas & keamanan, lihat sumber daya komputasi di bawah Sumber daya komputasi terhubung.

Memodifikasi klaster DB Amazon Aurora

Anda dapat mengubah pengaturan klaster DB untuk menyelesaikan tugas seperti mengubah periode retensi cadangan atau port basis datanya. Anda juga dapat memodifikasi instans DB dalam klaster DB untuk menyelesaikan tugas seperti mengubah kelas instans DB atau mengaktifkan Wawasan Performa untuknya. Topik ini memandu Anda dalam memodifikasi klaster DB Aurora dan instans DB-nya, serta menjelaskan masing-masing pengaturannya.

Kami menyarankan Anda menguji setiap perubahan pada instans klaster DB uji atau instans DB sebelum memodifikasi instans DB atau klaster DB produksi, sehingga Anda sepenuhnya memahami dampak dari setiap perubahan. Hal ini sangat penting ketika meng-upgrade versi basis data.

Topik

- Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API
- Memodifikasi instans DB dalam klaster DB
- Mengubah kata sandi untuk pengguna master database
- Pengaturan untuk Amazon Aurora
- Pengaturan yang tidak berlaku untuk klaster DB Amazon Aurora
- Pengaturan yang tidak berlaku untuk instans DB Amazon Aurora

Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API

Anda dapat memodifikasi cluster DB menggunakan AWS Management Console, AWS CLI, atau RDS API.

> **Note**
>
> Sebagian besar modifikasi dapat diterapkan segera atau selama periode pemeliharaan terjadwal berikutnya. Beberapa modifikasi, seperti mengaktifkan perlindungan penghapusan, akan diterapkan segera, terlepas dari kapan Anda memilih untuk menerapkannya.
> Mengubah kata sandi utama di AWS Management Console selalu diterapkan segera. Namun, saat menggunakan AWS CLI atau RDS API, Anda dapat memilih apakah akan segera menerapkan perubahan ini atau selama jendela pemeliharaan terjadwal berikutnya.
> Jika Anda menggunakan titik akhir SSL dan mengubah pengidentifikasi klaster DB, hentikan dan mulai ulang klaster DB untuk memperbarui titik akhir SSL. Untuk informasi selengkapnya, lihat Menghentikan dan memulai klaster DB Amazon Aurora.

Konsol

Untuk memodifikasi klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, lalu pilih klaster DB yang ingin dimodifikasi.
3. Pilih Modifikasi. Halaman Ubah klaster basis data akan muncul.
4. Ubah semua setelan yang Anda inginkan. Lihat informasi tentang setiap setelan di Pengaturan untuk Amazon Aurora.

   > **Note**
   >
   > Dalam AWS Management Console, beberapa perubahan tingkat instance hanya berlaku untuk instans DB saat ini, sementara yang lain berlaku untuk seluruh cluster DB. Untuk informasi tentang apakah pengaturan berlaku pada instans DB atau klaster DB, lihat cakupan untuk pengaturan tersebut dalam Pengaturan untuk Amazon Aurora. Untuk mengubah setelan yang memodifikasi seluruh cluster DB pada tingkat instans di AWS Management Console, ikuti instruksi diMemodifikasi instans DB dalam klaster DB.

5. Jika semua perubahan sudah sesuai dengan keinginan Anda, pilih Lanjutkan dan periksa ringkasan modifikasi.
6. Untuk menerapkan perubahan dengan segera, pilih Terapkan segera.
7. Di halaman konfirmasi, tinjau perubahan Anda. Jika sudah benar, pilih Modifikasi klaster untuk menyimpan perubahan Anda.

   Atau, pilih Kembali untuk mengedit perubahan, atau pilih Batal untuk membatalkan perubahan.

AWS CLI

Untuk memodifikasi cluster DB menggunakan AWS CLI, panggil modify-db-clusterperintah. Tentukan pengidentifikasi klaster DB, dan nilai untuk pengaturan yang ingin Anda modifikasi. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk Amazon Aurora.

> **Note**
>
> Beberapa pengaturan hanya berlaku untuk instans DB. Untuk mengubah pengaturan tersebut, ikuti petunjuk dalam Memodifikasi instans DB dalam klaster DB.

Example

Perintah berikut memodifikasi `mydbcluster` dengan mengatur periode retensi pencadangan menjadi 1 minggu (7 hari).

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --backup-retention-period 7
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --backup-retention-period 7
```

API RDS

Untuk memodifikasi klaster DB menggunakan API Amazon RDS, panggil operasi ModifyDBCluster. Tentukan pengidentifikasi klaster DB, dan nilai untuk pengaturan yang ingin Anda modifikasi. Untuk informasi tentang setiap parameter, lihat Pengaturan untuk Amazon Aurora.

> **Note**
>
> Beberapa pengaturan hanya berlaku untuk instans DB. Untuk mengubah pengaturan tersebut, ikuti petunjuk dalam Memodifikasi instans DB dalam klaster DB.

Memodifikasi instans DB dalam klaster DB

Anda dapat memodifikasi instans DB di cluster DB menggunakan AWS CLI,, atau RDS API. AWS Management Console

Saat Anda memodifikasi instans DB, Anda dapat langsung menerapkan perubahan. Untuk segera menerapkan perubahan, Anda memilih opsi Terapkan Segera di AWS Management Console, Anda menggunakan `--apply-immediately` parameter saat memanggil AWS CLI, atau Anda mengatur `ApplyImmediately` parameter `true` saat menggunakan Amazon RDS API.

Jika Anda tidak memilih untuk segera menerapkan perubahan, perubahan akan ditunda hingga periode pemeliharaan berikutnya. Selama periode pemeliharaan berikutnya, perubahan yang ditunda ini akan diterapkan. Jika Anda memilih untuk segera menerapkan perubahan, perubahan baru dan setiap perubahan yang sebelumnya ditunda akan diterapkan.

Untuk melihat modifikasi yang tertunda untuk jendela pemeliharaan berikutnya, gunakan describe-db-clusters AWS CLI perintah dan periksa `PendingModifiedValues` bidangnya.

> **Important**
>
> Jika salah satu dari modifikasi yang ditunda memerlukan waktu henti, memilih Terapkan segera dapat menyebabkan waktu henti yang tidak terduga untuk instans DB. Tidak ada waktu henti untuk instans DB lain dalam klaster DB.
> Modifikasi yang Anda tunda tidak tercantum dalam output dari perintah CLI `describe-pending-maintenance-actions`. Tindakan pemeliharaan hanya mencakup upgrade sistem yang Anda jadwalkan untuk periode pemeliharaan berikutnya.

Konsol

Untuk memodifikasi instans DB dalam klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih instans DB yang ingin Anda modifikasi.
3. Untuk Tindakan, pilih Modifikasi. Halaman Modifikasi instans DB muncul.
4. Ubah pengaturan yang Anda inginkan. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk Amazon Aurora.

> **Note**
>
> Beberapa pengaturan berlaku untuk keseluruhan klaster DB dan harus diubah pada tingkat klaster. Untuk mengubah pengaturan tersebut, ikuti petunjuk di Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.
> Dalam AWS Management Console, beberapa perubahan tingkat instance hanya berlaku untuk instans DB saat ini, sementara yang lain berlaku untuk seluruh cluster DB. Untuk informasi tentang apakah pengaturan berlaku pada instans DB atau klaster DB, lihat cakupan untuk pengaturan tersebut dalam Pengaturan untuk Amazon Aurora.

5. Jika semua perubahan sudah sesuai dengan keinginan Anda, pilih Lanjutkan dan periksa ringkasan modifikasi.
6. Untuk menerapkan perubahan dengan segera, pilih Terapkan segera.
7. Di halaman konfirmasi, tinjau perubahan Anda. Jika sudah benar, pilih Modifikasi instans DB untuk menyimpan perubahan Anda.

   Alternatifnya, pilih Kembali untuk mengedit perubahan, atau pilih Batal untuk membatalkan perubahan Anda.

AWS CLI

Untuk memodifikasi instans DB di cluster DB dengan menggunakan AWS CLI, panggil modify-db-instanceperintah. Tentukan pengidentifikasi instans DB, dan nilai untuk pengaturan yang ingin Anda modifikasi. Untuk informasi tentang setiap parameter, lihat Pengaturan untuk Amazon Aurora.

> **Note**
>
> Beberapa pengaturan berlaku untuk seluruh klaster DB. Untuk mengubah pengaturan tersebut, ikuti petunjuk dalam Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.

Example

Kode berikut memodifikasi `mydbinstance` dengan menetapkan kelas instans DB untuk `db.r4.xlarge`. Perubahan akan diterapkan selama periode pemeliharaan berikutnya dengan

menggunakan `--no-apply-immediately`. Gunakan `--apply-immediately` untuk segera menerapkan perubahan.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --db-instance-class db.r4.xlarge \
    --no-apply-immediately
```

Untuk Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --db-instance-class db.r4.xlarge ^
    --no-apply-immediately
```

RDS API

Untuk memodifikasi instans DB menggunakan Amazon RDS API, panggil operasi ModifyDBInstance. Tentukan pengidentifikasi instans DB, dan nilai untuk pengaturan yang ingin Anda modifikasi. Untuk informasi tentang setiap parameter, lihat Pengaturan untuk Amazon Aurora.

> **Note**
>
> Beberapa pengaturan berlaku untuk seluruh klaster DB. Untuk mengubah pengaturan tersebut, ikuti petunjuk dalam Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.

Mengubah kata sandi untuk pengguna master database

Anda dapat menggunakan AWS Management Console atau AWS CLI untuk mengubah kata sandi pengguna utama.

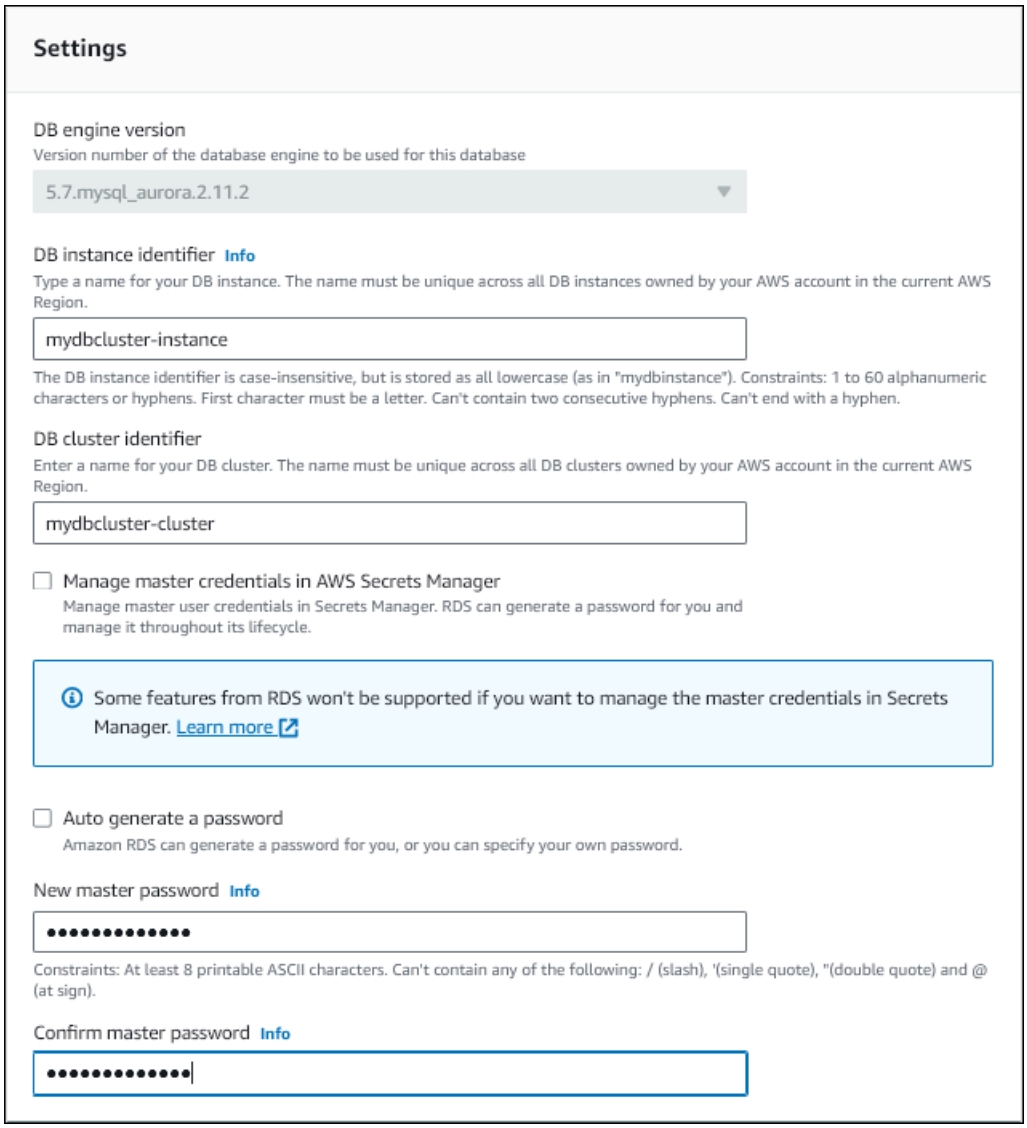
Konsol

Anda memodifikasi instance DB penulis untuk mengubah kata sandi pengguna master menggunakan file AWS Management Console.

Untuk mengubah kata sandi pengguna utama

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih instans DB yang ingin Anda modifikasi.
3. Untuk Tindakan, pilih Modifikasi.

   Halaman Modifikasi instans DB muncul.
4. Masukkan kata sandi master baru.
5. Untuk Konfirmasi kata sandi utama, masukkan kata sandi baru yang sama.



6. Pilih Lanjutkan dan periksa ringkasan modifikasi.

> **Note**
>
> Perubahan kata sandi selalu diterapkan segera.

7. Di halaman konfirmasi, pilih Modifikasi instans DB.

CLI

Anda memanggil modify-db-clusterperintah untuk mengubah kata sandi pengguna master menggunakan file AWS CLI. Tentukan pengidentifikasi cluster DB dan kata sandi baru, seperti yang ditunjukkan pada contoh berikut.

Anda tidak perlu menentukan`--apply-immediately|--no-apply-immediately`, karena perubahan kata sandi selalu diterapkan segera.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --master-user-password mynewpassword
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --master-user-password mynewpassword
```

Pengaturan untuk Amazon Aurora

Tabel berikut berisi detail tentang pengaturan mana yang dapat Anda modifikasi, metode untuk memodifikasi pengaturan, dan cakupan pengaturan. Cakupan menentukan apakah pengaturan berlaku untuk keseluruhan klaster DB atau apakah pengaturan hanya dapat diatur untuk instans DB tertentu.

> **Note**
>
> Pengaturan tambahan tersedia jika Anda memodifikasi klaster DB Aurora Serverless v1 atau Aurora Serverless v2. Untuk informasi tentang pengaturan ini, lihat Memodifikasi klaster DB Aurora Serverless v1 dan Mengelola cluster Aurora Serverless v2 DB.

Beberapa pengaturan tidak tersedia untuk Aurora Serverless v1 dan Aurora Serverless v2 karena batasannya. Lihat informasi yang lebih lengkap di Batasan Aurora Serverless v1 dan Persyaratan dan batasan untuk Aurora Serverless v2.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Peningkatan versi minor otomatis Hal ini menentukan apakah Anda ingin instans DB menerima upgrade versi mesin minor pilihan secara otomatis ketika upgrade tersebut tersedia. Upgrade hanya diinstal selama periode pemeliharaan terjadwal Anda. Untuk informasi selengkapnya tentang pembaruan mesin, lihat Pembaruan Amazon Aurora PostgreSQL dan Pembaruan mesin basis data untuk Amazon Aurora MySQL. Untuk informasi selengkapnya tentang pengaturan Upgrade versi minor otomatis	Note Pengaturan ini diaktifkan secara default. Untuk setiap klaster baru, pilih nilai yang sesuai untuk pengaturan ini berdasarkan tingkat kepentingannya, masa penggunaan yang diharapkan, dan jumlah pengujian verifikasi yang Anda lakukan setelah setiap upgrade. Saat Anda mengubah pengaturan ini,	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini. Pemadaman terjadi selama periode pemeliharaan mendatang saat Aurora menerapkan upgrade otomatis.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
untuk Aurora MySQL, lihat Mengaktifkan peningkatan otomatis di antara versi minor Aurora MySQL.	lakukan modifikasi ini untuk setiap instans DB di klaster Aurora Anda. Jika ada instans DB di kluster Anda yang menonaktifkan pengaturan ini, klaster tersebut tidak dimutakhirkan secara otomatis. Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB. Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--auto-minor-version-upgrade|--no-auto-minor-version-upgrade` opsi. Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `AutoMinorVersionUpgrade` .		

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Periode retensi cadangan Jumlah hari retensi cadangan otomatis. Nilai minimum adalah 1. Untuk informasi selengkapnya, lihat Cadangan.	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--backup-retention-period` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `BackupRetentionPeriod` .	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Jendela cadangan (Waktu mulai) Rentang waktu saat pencadangan otomatis basis data Anda terjadi. Jendela cadangan adalah waktu mulai dalam Waktu Universal (UTC) Terkoordinasi, dan durasi dalam jam. Cadangan Aurora bersifat terus-menerus dan bertahap, tetapi periode pencadangan digunakan untuk membuat cadangan sistem harian yang dipertahankan dalam periode retensi cadangan. Anda dapat menyalin cadangan ini untuk mempertahankannya di luar periode retensi. Jendela pemeliharaan dan jendela cadangan untuk klaster DB tidak dapat tumpang tindih.	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--preferred-backup-window` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `PreferredBackupWindow` .	Seluruh klaster DB.	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Untuk informasi selengkapnya, lihat Jendela cadangan.			
Pengaturan kapasitas Properti penskalaan klaster DB Aurora Serverless v1. Anda hanya dapat memodifikasi properti penskalaan untuk klaster DB dalam mode mesin DB `serverless` . Untuk informasi tentang Aurora Serverless v1, lihat Menggunakan Amazon Aurora Serverless v1.	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--scaling-configuration` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `ScalingConfiguration` .	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini. Perubahan terjadi dengan segera. Pengaturan ini mengabaikan pengaturan "Terapkan segera".

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Otoritas sertifikat Otoritas sertifikat (CA) untuk sertifikat server yang digunakan oleh instans DB.	Menggunakan AWS Management Console,Memodifikasi instans DB dalam klaster DB. Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--ca-certificate-identifier` opsi. Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `CACertificateIdentifier` .	Hanya instans DB yang ditentukan	Pemadaman hanya terjadi jika mesin DB tidak mendukung rotasi tanpa pengaktifan ulang. Anda dapat menggunakan `describe-db-engine-versions` AWS CLI perintah untuk menentukan apakah mesin DB mendukung rotasi tanpa restart.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Konfigurasi penyimpanan klaster Jenis penyimpanan untuk klaster DB: Aurora I/O-Optimized atau Aurora Standard. Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--storage-type` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `StorageType` .	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Salin tanda ke snapshot Pilih untuk menentukan agar tanda/tag yang ditentukan untuk klaster DB ini disalin ke snapshot DB yang dibuat dari klaster DB ini. Untuk informasi selengkapnya, lihat Memberi tag pada sumber daya Amazon RDS.	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--no-copy-tags-to-snapshot` opsi `--copy-tags-to-snapshot` or. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `CopyTagsToSnapshot` .	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Data API Anda dapat mengakses Aurora Serverless v1 dengan aplikasi berbasis layanan web, termasuk dan. AWS Lambda AWS AppSync Pengaturan ini hanya berlaku untuk klaster DB Aurora Serverless v1. Untuk informasi selengkapnya, lihat Menggunakan RDS Data API.	Menggunakan AWS Management Console,Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster`dan atur `--enable-http-endpoint` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `EnableHttpEndpoint` .	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Autentikasi basis data Autentikasi basis data yang ingin Anda gunakan. Untuk MySQL: • Pilih Autentikasi kata sandi untuk mengautentikasi pengguna basis data dengan kata sandi basis data saja. • Pilih Kata sandi dan autentikasi basis data IAM untuk mengautentikasi pengguna basis data dengan kata sandi basis data dan kredensial pengguna melalui pengguna dan peran IAM. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM. Untuk PostgreSQL: • Pilih Autentikasi basis data IAM	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan modify-db-clusterdan atur opsi berikut: • Untuk autentikasi IAM, atur opsi `--enable-iam-database-authentication|--no-enable-iam-database-authentication` . • Untuk autentikasi Kerberos, atur opsi `--domain` dan `--domain-iam-role-name` . Menggunakan API RDS, panggil ModifyDBCluster dan tetapkan parameter berikut:	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
untuk mengauten tikasi pengguna basis data dengan kata sandi basis data dan kredensia l pengguna melalui pengguna dan peran. Untuk informasi selengkap nya, lihat Autentika si basis data IAM. • Pilih Autentikasi Kerberos untuk mengautentikasi kata sandi basis data dan kredensia l pengguna menggunak an autentika si Kerberos. Untuk informasi selengkapnya, lihat Menggunak an autentikasi Kerberos dengan Aurora PostgreSQ L.	• Untuk autentikasi IAM, atur parameter `EnableIAM DatabaseA uthentica tion` . • Untuk autentika si Kerberos, atur parameter `Domain` dan `DomainIAM RoleName` .		

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Port basis data Port yang ingin Anda gunakan untuk mengakses klaster DB.	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--port` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `Port`.	Seluruh klaster DB	Pemadaman terjadi selama perubahan ini. Semua instans DB dalam klaster DB langsung di-boot ulang.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Pengidentifikasi klaster DB Pengidentifikasi klaster basis data. Nilai ini disimpan sebagai string huruf kecil. Saat Anda mengubah pengidentifikasi klaster DB, titik akhir klaster DB berubah. Titik akhir instans DB di klaster DB tidak berubah.	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--new-db-cluster-identifier` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `NewDBClusterIdentifier`.	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Grup parameter klaster DB Grup parameter klaster basis data yang Anda inginkan terkait dengan klaster basis data. Untuk informasi selengkapnya, lihat Bekerja dengan grup parameter.	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--db-cluster-parameter-group-name` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `DBClusterParameterGroupName` .	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini. Ketika Anda mengubah grup parameter, perubahan pada beberapa parameter akan diterapkan ke instans DB dalam klaster DB segera tanpa boot ulang. Perubahan pada parameter lain diterapkan hanya setelah instans DB dalam klaster DB di-boot ulang.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Kelas instans DB Kelas instans DB yang ingin Anda gunakan. Untuk informasi selengkapnya, lihat Kelas instans DB Aurora.	Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB. Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--db-instance-class` opsi. Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `DBInstanceClass` .	Hanya instans DB yang ditentukan	Pemadaman terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Pengidentifikasi instans DB Pengidentifikasi instans DB. Nilai ini disimpan sebagai string huruf kecil.	Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB. Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--new-db-instance-identifier` opsi. Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `NewDBInstanceIdentifier` .	Hanya instans DB yang ditentukan	Waktu henti terjadi selama perubahan ini kecuali jika versi mesin DB Anda mendukung pengunggahan dinamis SSL. Untuk menentukan apakah versi Anda memerlukan restart, jalankan AWS CLI perintah berikut: `aws rds describe-db-engine-versions \ --default-only \ --engine your-db-engine \ --query 'DBEngineVersions[*].SupportsCertificateRotationWithoutRestart'`

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Grup parameter DB Grup parameter DB terkait instans DB yang Anda inginkan. Untuk informasi selengkapnya, lihat Bekerja dengan grup parameter.	Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB. Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--db-parameter-group-name` opsi. Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `DBParameterGroupName` .	Hanya instans DB yang ditentukan	Pemadaman tidak akan terjadi selama perubahan ini. Ketika Anda mengaitkan grup parameter DB baru dengan instans DB, parameter statis dan dinamis yang dimodifikasi diterapkan hanya setelah instans DB di boot ulang. Namun, jika Anda memodifikasi parameter dinamis dalam grup parameter DB setelah mengaitkannya dengan instans DB, perubahan ini akan langsung diterapkan tanpa reboot. Untuk informasi lebih lanjut, lihat Bekerja dengan grup parameter dan Mem-boot ulang klaster DB Amazon Aurora atau instans DB Amazon Aurora.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Perlindungan penghapusan Aktifkan perlindungan penghapusan untuk mencegah klaster basis data dari terhapus. Untuk informasi selengkapnya, lihat Perlindungan penghapusan untuk klaster Aurora.	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--deletion-protection|--no-deletion-protection` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `DeletionProtection` .	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Versi mesin Versi mesin DB yang ingin Anda gunakan. Sebelum meng-upgrade klaster DB produksi Anda, sebaiknya Anda menguji proses upgrade pada klaster DB uji untuk memverifikasi durasinya dan untuk memvalidasi aplikasi Anda.	Menggunakan AWS Management Console,Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster`dan atur `--engine-version` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `EngineVersion` .	Seluruh klaster DB	Pemadaman terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Pemantauan yang ditingkatkan Aktifkan Pemantauan yang ditingkatkan untuk memungkinkan pengumpulan metrik secara waktu nyata untuk sistem operasi yang dijalankan oleh instans DB Anda. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan.	Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB. Menggunakan AWS CLI, menjalankan `modify-db-instance` dan mengatur `--monitoring-role-arn` dan `--monitoring-interval` pilihan. Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `MonitoringRoleArn` dan `MonitoringInterval` .	Hanya instans DB yang ditentukan	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Ekspor log Pilih jenis log untuk dipublikasikan ke Amazon CloudWatch Logs. Untuk informasi selengkapnya, lihat File log basis data Aurora MySQL.	Menggunakan AWS Management Console,Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster`dan atur `--cloudwatch-logs-export-configuration` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `CloudwatchLogsExportConfiguration` .	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Jendela pemeliharaan Rentang waktu selama pemeliharaan sistem berlangsung. Pemeliharaan sistem mencakup peningkatan, jika ada. Jendela pemeliharaan adalah waktu mulai dalam Waktu Terkoordinasi Universal (UTC), dan durasi dalam jam. Jika Anda mengatur jendela ke waktu saat ini, harus ada minimal 30 menit antara waktu saat ini dan akhir jendela untuk memastikan setiap perubahan yang tertunda diterapkan. Anda dapat mengatur jendela pemeliharaan secara terpisah untuk klaster DB dan untuk setiap instans DB dalam klaster DB. Jika cakupan modifikasi adalah seluruh klaster DB, modifikasi dilakukan	Untuk mengubah jendela pemeliharaan untuk cluster DB menggunakan AWS Management Console,Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Untuk mengubah jendela pemeliharaan untuk instans DB menggunakan AWS Management Console,Memodifikasi instans DB dalam klaster DB. Untuk mengubah jendela pemeliharaan untuk cluster DB menggunakan AWS CLI, jalankan `modify-db-cluster`dan atur `--preferred-maintenance-window` opsi. Untuk mengubah jendela pemeliharaan untuk instans DB menggunakan	Seluruh klaster DB atau satu instans DB	Jika ada satu atau beberapa tindakan tertunda yang menyebabkan pemadaman, dan periode pemeliharaan diubah untuk menyertakan waktu saat ini, tindakan tertunda tersebut akan segera diterapkan, dan pemadaman pun terjadi.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
selama jendela pemeliharaan klaster DB. Jika cakupan modifikasi adalah instans DB, modifikasi dilakukan selama jendela pemeliharaan instans DB tersebut. Jendela pemeliharaan dan jendela cadangan untuk klaster DB tidak dapat tumpang tindih. Untuk informasi selengkapnya, lihat Periode pemeliharaan Amazon RDS.	AWS CLI, jalankan `modify-db-instance` dan atur `--preferred-maintenance-window` opsi. Untuk mengubah jendela pemeliharaan untuk klaster DB menggunakan API RDS, panggil `ModifyDBCluster` dan atur parameter `PreferredMaintenanceWindow` . Untuk mengubah jendela pemeliharaan untuk instans DB menggunakan API RDS, panggil `ModifyDBInstance` dan atur parameter `PreferredMaintenanceWindow` .		

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Kelola kredensi master di AWS Secrets Manager Pilih Kelola kredensial master di AWS Secrets Manager untuk mengelola kata sandi pengguna master dalam sebuah rahasia di Secrets Manager. Atau, pilih kunci KMS yang akan digunakan untuk melindungi rahasia. Pilih dari kunci KMS di akun Anda, atau masukkan kunci dari akun yang berbeda. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. Jika Aurora sudah mengelola kata sandi pengguna master untuk klaster DB, Anda dapat merotasi kata sandi pengguna	Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB. Menggunakan AWS CLI, menjalankan `modify-db-cluster` dan mengatur `--manage-master-user-password | --no-manage-master-user-password` dan `--master-user-secret-kms-key-id` pilihan. Untuk segera merotasi kata sandi pengguna master, atur opsi `--rotate-master-user-password` . Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `ManageMasterUserPassword` dan `MasterUse`	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
master dengan memilih Putar rahasia segera. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager.	`rSecretKmsKeyId` . Untuk segera merotasi kata sandi pengguna master, atur parameter `RotateMasterUserPassword` ke `true`.		

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Jenis jaringan Protokol alamat IP yang didukung oleh klaster DB. IPv4 untuk menentukan bahwa sumber daya dapat berkomunikasi dengan klaster DB hanya melalui protokol alamat IPv4. Mode tumpukan ganda untuk menentukan bahwa sumber daya dapat berkomunikasi dengan klaster DB melalui IPv4, IPv6, atau keduanya. Gunakan mode tumpukan ganda jika Anda memiliki sumber daya yang harus berkomunikasi dengan klaster DB Anda melalui protokol alamat IPv6. Untuk menggunakan mode tumpukan ganda, pastikan ada setidaknya dua subnet yang	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--network-type` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `NetworkType` .	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
mencakup dua Zona Ketersediaan yang mendukung protokol jaringan IPv4 dan IPv6. Selain itu, pastikan Anda mengaitkan blok CIDR IPv6 dengan subnet di grup subnet DB yang Anda tentukan. Untuk informasi selengkapnya, lihat Penentuan alamat IP Amazon Aurora.			

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Kata sandi master baru Kata sandi untuk pengguna master Anda. • Untuk Aurora MySQL, kata sandi harus berisi 8–41 karakter ASCII yang dapat dicetak. • Untuk Aurora PostgreSQL, kata sandi harus berisi 8–99 karakter ASCII yang dapat dicetak. • Kata sandi tidak dapat berisi `/`, `"`, `@`, atau spasi.	Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--master-user-password` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `MasterUserPassword` .	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Wawasan Performa Opsi untuk mengaktifkan Wawasan Performa, alat yang memantau beban instans DB Anda sehingga Anda dapat menganalisis dan memecahkan masalah performa basis data Anda. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.	Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB. Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--enable-performance-insights|--no-enable-performance-insights` opsi. Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `EnablePerformanceInsights` .	Hanya instans DB yang ditentukan	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Wawasan Performa AWS KMS key AWS KMS key Pengidentifikasi untuk enkripsi data Performance Insights. Pengidentifikasi kunci KMS adalah Amazon Resource Name (ARN), pengident ifikasi kunci, atau alias kunci untuk kunci KMS. Untuk informasi selengkapnya, lihat Mengaktifkan dan menonaktifkan Wawasan Performa.	Menggunakan AWS Management Console,Memodifik asi instans DB dalam klaster DB. Menggunakan AWS CLI, jalankan `modify-db- instance` dan atur `--performance- insights-kms- key-id` opsi. Menggunakan API RDS, panggil `ModifyDBI nstance` dan tetapkan parameter `Performan ceInsight sKMSKeyId` .	Hanya instans DB yang ditentukan	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Periode retensi Wawasan Performa Jumlah waktu (dalam hari) untuk mempertahankan data Wawasan Performa. Pengaturan retensi di tingkat gratis adalah Default (7 hari). Untuk mempertahankan data kinerja Anda lebih lama, tetapkan 1–24 bulan. Untuk informasi selengkapnya tentang periode retensi, lihat Harga dan retensi data untuk Wawasan Performa. Untuk informasi selengkapnya, lihat Mengaktifkan dan menonaktifkan Wawasan Performa.	Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB. Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--performance-insights-retention-period` opsi. Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `PerformanceInsightsRetentionPeriod` .	Hanya instans DB yang ditentukan	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Tingkat promosi Nilai yang menentukan urutan Replika Aurora dipromosikan ke instans primer dalam klaster DB, setelah kegagalan instans primer yang ada. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora.	Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB. Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--promotion-tier` opsi. Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `PromotionTier` .	Hanya instans DB yang ditentukan	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Akses publik Dapat diakses publik untuk memberikan alamat IP publik ke instans DB, yang berarti dapat diakses di luar VPC. Agar dapat diakses oleh publik, instans DB juga harus berada di subnet publik di VPC. Tidak dapat diakses publik agar instans DB hanya dapat diakses dari dalam VPC. Untuk informasi selengkapnya, lihat Menyembunyikan instans DB dalam VPC dari internet. Untuk terhubung ke instans DB dari luar Amazon VPC-nya, instans DB harus dapat diakses secara publik, akses harus diberikan menggunakan aturan masuk grup keamanan instans DB, dan	Menggunakan AWS Management Console, Memodifikasi instans DB dalam klaster DB. Menggunakan AWS CLI, jalankan `modify-db-instance` dan atur `--publicly-accessible|--no-publicly-accessible` opsi. Menggunakan API RDS, panggil `ModifyDBInstance` dan tetapkan parameter `PubliclyAccessible` .	Hanya instans DB yang ditentukan	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
persyaratan lain harus dipenuhi. Untuk mengetahui informasi nya, lihat Tidak dapat terhubung ke instans DB Amazon RDS. Jika instans DB Anda tidak dapat diakses publik, Anda juga dapat menggunak an koneksi VPN AWS Site-to-Site AWS Direct Connect atau koneksi untuk mengaksesnya dari jaringan pribadi. Untuk informasi selengkapnya, lihat Privasi lalu lintas antarjaringan.			

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Pengaturan kapasitas Serverless v2 Kapasitas basis data klaster DB Aurora Serverless v2, yang diukur dalam Unit Kapasitas Aurora (ACU). Untuk informasi selengkapnya, lihat Mengatur rentang kapasitas Aurora Serverless v2 untuk sebuah klaster.	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--serverless-v2-scaling-configuration` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `ServerlessV2ScalingConfiguration`.	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini. Perubahan terjadi dengan segera. Pengaturan ini mengabaikan pengaturan "Terapkan segera".

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Grup keamanan Grup keamanan yang ingin Anda kaitkan dengan klaster DB. Untuk informasi selengkapnya, lihat Mengontrol akses dengan grup keamanan.	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--vpc-security-group-ids` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `VpcSecurityGroupIds` .	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan dan deskripsi	Metode	Cakupan	Catatan waktu henti
Jendela Backtrack Target Jumlah waktu yang Anda inginkan untuk melakukan backtrack klaster DB Anda, dalam hitungan detik. Pengaturan ini hanya tersedia untuk Aurora MySQL dan hanya jika klaster DB dibuat dengan Backtrack diaktifkan.	Menggunakan AWS Management Console, Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Menggunakan AWS CLI, jalankan `modify-db-cluster` dan atur `--backtrack-window` opsi. Menggunakan API RDS, panggil `ModifyDBCluster` dan tetapkan parameter `BacktrackWindow` .	Seluruh klaster DB	Pemadaman tidak akan terjadi selama perubahan ini.

Pengaturan yang tidak berlaku untuk klaster DB Amazon Aurora

Pengaturan berikut dalam AWS CLI perintah `modify-db-cluster` dan operasi RDS API `ModifyDBCluster` tidak berlaku untuk cluster Amazon Aurora DB.

> **Note**
>
> Anda tidak dapat menggunakan AWS Management Console untuk memodifikasi pengaturan ini untuk cluster Aurora DB.

AWS CLI pengaturan	Pengaturan API RDS
`--allocated-storage`	`AllocatedStorage`
`--auto-minor-version-upgrade | --no-auto-minor-version-upgrade`	`AutoMinorVersionUpgrade`
`--db-cluster-instance-class`	`DBClusterInstanceClass`
`--enable-performance-insights | --no-enable-performance-insights`	`EnablePerformanceInsights`
`--iops`	`Iops`
`--monitoring-interval`	`MonitoringInterval`
`--monitoring-role-arn`	`MonitoringRoleArn`
`--option-group-name`	`OptionGroupName`
`--performance-insights-kms-key-id`	`PerformanceInsightsKMSKeyId`
`--performance-insights-retention-period`	`PerformanceInsightsRetentionPeriod`

Pengaturan yang tidak berlaku untuk instans DB Amazon Aurora

Pengaturan berikut dalam AWS CLI perintah `modify-db-instance`dan operasi RDS API `ModifyDBInstance`tidak berlaku untuk instans Amazon Aurora DB.

> **Note**
>
> Anda tidak dapat menggunakan AWS Management Console untuk memodifikasi pengaturan ini untuk instans Aurora DB.

AWS CLI pengaturan	Setelan API RDS
`--allocated-storage`	`AllocatedStorage`
`--allow-major-version-upgrade|--no-allow-major-version-upgrade`	`AllowMajorVersionUpgrade`
`--copy-tags-to-snapshot|--no-copy-tags-to-snapshot`	`CopyTagsToSnapshot`
`--domain`	`Domain`
`--db-security-groups`	`DBSecurityGroups`
`--db-subnet-group-name`	`DBSubnetGroupName`
`--domain-iam-role-name`	`DomainIAMRoleName`
`--multi-az|--no-multi-az`	`MultiAZ`
`--iops`	`Iops`
`--license-model`	`LicenseModel`
`--network-type`	`NetworkType`
`--option-group-name`	`OptionGroupName`
`--processor-features`	`ProcessorFeatures`
`--storage-type`	`StorageType`
`--tde-credential-arn`	`TdeCredentialArn`
`--tde-credential-password`	`TdeCredentialPassword`
`--use-default-processor-features|--no-use-default-processor-features`	`UseDefaultProcessorFeatures`

Menambahkan Replika Aurora ke klaster DB

Klaster DB Aurora dengan replikasi memiliki satu instans DB primer dan hingga 15 Replika Aurora. Instans DB primer mendukung operasi baca dan tulis, dan melakukan semua modifikasi data pada volume klaster. Replika Aurora terhubung ke volume penyimpanan yang sama dengan instans DB primer, tetapi hanya mendukung operasi baca. Anda menggunakan Replika Aurora untuk mengalihkan beban kerja baca dari instans DB primer. Untuk informasi selengkapnya, lihat Replika Aurora.

Replika Amazon Aurora memiliki batasan berikut ini:

- Anda tidak dapat membuat Replika Aurora untuk klaster DB Aurora Serverless v1. Aurora Serverless v1 memiliki instans DB tunggal yang menaikkan dan menurunkan skalanya secara otomatis untuk mendukung semua operasi baca dan tulis basis data.

  Namun, Anda dapat menambahkan instans pembaca ke klaster DB Aurora Serverless v2. Untuk informasi selengkapnya, lihat Menambahkan pembaca Aurora Serverless v2.

Kami menyarankan agar Anda mendistribusikan instans primer dan Replika Aurora dari klaster DB Anda ke beberapa Zona Ketersediaan untuk meningkatkan ketersediaan klaster DB Anda. Untuk informasi selengkapnya, lihat Ketersediaan wilayah.

Untuk menghapus Replika Aurora dari klaster DB Aurora, hapus Replika Aurora dengan mengikuti petunjuk dalam Menghapus instans dari klaster DB Aurora.

> **Note**
>
> Amazon Aurora juga mendukung replikasi dengan basis data eksternal, atau instans DB RDS. Instans DB RDS harus berada di Wilayah AWS yang sama seperti Amazon Aurora. Untuk informasi selengkapnya, lihat Replikasi dengan Amazon Aurora.

Anda dapat menambahkan Replika Aurora ke klaster DB menggunakan AWS Management Console, AWS CLI, atau API RDS.

Konsol

Untuk menambahkan Replika Aurora ke klaster DB

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih klaster DB tempat Anda ingin menambahkan instans DB baru.
3. Pastikan klaster dan instans primer berada dalam status Tersedia. Jika klaster DB atau instans primer berada dalam status transisi seperti Membuat, Anda tidak dapat menambahkan replika.

   Jika cluster tidak memiliki instance utama, buat satu menggunakan create-db-instanceAWS CLIperintah. Situasi ini dapat muncul jika Anda menggunakan CLI untuk memulihkan snapshot klaster DB lalu melihat klaster ini di AWS Management Console.
4. Untuk Tindakan, pilih Tambahkan pembaca.

   Halaman Tambahkan pembaca muncul.
5. Di halaman Tambahkan pembaca, tentukan opsi untuk Replika Aurora Anda. Tabel berikut menunjukkan pengaturan untuk Replika Aurora.

Untuk opsi ini	Lakukan hal berikut
Zona ketersediaan	Tentukan apakah Anda ingin menentukan Zona Ketersediaan tertentu. Daftar ini hanya mencakup Zona Ketersediaan yang dipetakan ke grup subnet DB yang Anda pilih saat Anda membuat klaster DB. Untuk informasi selengkapnya tentang Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan.
Dapat diakses publik	Pilih `Yes` untuk memberikan alamat IP publik pada Replika Aurora; jika tidak, pilih `No`. Untuk informasi selengkapnya tentang menyembunyikan Replika Aurora dari akses publik, lihat Menyembunyikan instans DB dalam VPC dari internet.
Enkripsi	Pilih `Enable encryption` untuk mengaktifkan enkripsi saat diam untuk Replika Aurora ini. Untuk

Untuk opsi ini	Lakukan hal berikut
	informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.
Kelas instans DB	Pilih kelas instans DB yang menentukan persyaratan pemrosesan dan memori untuk Replika Aurora. Untuk informasi selengkapnya tentang opsi kelas instans DB, lihat Kelas instans DB Aurora.
Sumber replika Aurora	Pilih pengidentifikasi instans primer yang akan dibuatkan Replika Aurora.
Pengidentifikasi instans DB	Ketik nama instans yang unik untuk akun Anda di Wilayah AWS yang Anda pilih. Anda dapat memilih untuk menambahkan beberapa detail ke nama ini, seperti menyertakan Wilayah AWS dan mesin DB yang Anda pilih, misalnya **`aurora-read-instance1`** .
Prioritas	Pilih prioritas failover untuk instans. Jika Anda tidak memilih nilai, nilai default-nya adalah tier-1. Prioritas ini akan menentukan urutan promosi Aurora Replika saat melakukan pemulihan dari kegagalan instans primer. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora.
Port basis data	Port untuk Replika Aurora sama dengan port untuk klaster DB.
Grup parameter DB	Pilih grup parameter. Aurora memiliki grup parameter default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter Anda sendiri. Untuk informasi selengkapnya tentang grup parameter, lihat Bekerja dengan grup parameter.

Untuk opsi ini	Lakukan hal berikut
Wawasan Performa	Kotak centang Aktifkan Wawasan Performa dipilih secara default. Nilai ini tidak diwarisi dari instans penulis. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.
Pemantauan yang ditingkatkan	Pilih Aktifkan pemantauan yang ditingkatkan untuk mengaktifkan metrik pengumpulan secara waktu nyata untuk sistem operasi tempat klaster DB Anda berjalan. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan.
Peran Pemantauan	Hanya tersedia jika Pemantauan yang Ditingkatkan diatur ke Aktifkan pemantauan yang ditingkatkan. Pilih peran IAM yang Anda buat untuk mengizinkan Amazon RDS berkomunikasi dengan CloudWatch Log Amazon untuk Anda, atau pilih Default agar RDS membuat peran untuk nama Anda. `rds-monitoring-role` Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan.
Granularitas	Hanya tersedia jika Pemantauan yang Ditingkatkan diatur ke Aktifkan pemantauan yang ditingkatkan. Atur interval, dalam detik, di antara waktu pengumpulan metrik untuk klaster DB Anda.

Untuk opsi ini	Lakukan hal berikut
Peningkatan versi minor otomatis	Pilih Aktifkan peningkatan versi minor otomatis jika Anda ingin klaster DB Aurora Anda menerima upgrade versi Mesin DB minor secara otomatis saat tersedia. Pengaturan Peningkatan versi minor otomatis berlaku untuk klaster DB Aurora PostgreSQL dan Aurora MySQL. Untuk klaster Aurora MySQL 2.x, pengaturan ini meng-upgrade klaster ke versi maksimum 2.07.2. Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora PostgreSQL, lihat Pembaruan Amazon Aurora PostgreSQL. Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora MySQL, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL.

6. Pilih Tambahkan pembaca untuk membuat Replika Aurora.

AWS CLI

Untuk membuat Replika Aurora di cluster DB Anda, jalankan perintah. create-db-instanceAWS CLI Sertakan nama klaster DB sebagai opsi `--db-cluster-identifier`. Anda dapat menentukan Zona Ketersediaan secara opsional untuk Replika Aurora menggunakan parameter `--availability-zone` sebagaimana ditunjukkan dalam contoh berikut.

Misalnya, perintah berikut membuat Replika Aurora baru yang kompatibel dengan MySQL 5.7 bernama `sample-instance-us-west-2a`.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
    --availability-zone us-west-2a
```

Untuk Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large ^
    --availability-zone us-west-2a
```

Perintah berikut membuat Replika Aurora baru yang kompatibel dengan MySQL 5.7 bernama `sample-instance-us-west-2a`.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class
 db.r5.large \
    --availability-zone us-west-2a
```

Untuk Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora --db-instance-class
 db.r5.large ^
    --availability-zone us-west-2a
```

Perintah berikut membuat Replika Aurora baru yang kompatibel dengan PostgreSQL bernama `sample-instance-us-west-2a`.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a \
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large \
    --availability-zone us-west-2a
```

Untuk Windows:

```
aws rds create-db-instance --db-instance-identifier sample-instance-us-west-2a ^
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-
class db.r5.large ^
    --availability-zone us-west-2a
```

API RDS

Untuk membuat Replika Aurora dalam klaster DB Anda, panggil operasi CreateDBInstance. Sertakan nama klaster DB sebagai parameter `DBClusterIdentifier`. Anda dapat menentukan Zona Ketersediaan secara opsional untuk Replika Aurora menggunakan parameter `AvailabilityZone`.

Mengelola performa dan penskalaan untuk klaster DB Aurora

Anda dapat menggunakan opsi berikut untuk mengelola performa dan penskalaan untuk klaster DB dan instans DB Aurora:

Topik

- Penskalaan penyimpanan
- Penskalaan instans
- Penskalaan baca
- Mengelola koneksi
- Mengelola rencana eksekusi kueri

Penskalaan penyimpanan

Penyimpanan Aurora secara otomatis diskalakan dengan data dalam volume klaster Anda. Seiring pertumbuhan data Anda, penyimpanan volume klaster Anda dapat diperluas hingga maksimum 128 tebibyte (TiB) atau 64 TiB. Ukuran maksimum ini bergantung pada versi mesin DB. Untuk mempelajari data yang termasuk dalam volume klaster, lihat Penyimpanan dan keandalan Amazon Aurora. Untuk detail tentang ukuran maksimum untuk versi tertentu, lihat Batas ukuran Amazon Aurora.

Ukuran volume klaster Anda dievaluasi setiap jam untuk menentukan biaya penyimpanan Anda. Untuk informasi harga, lihat halaman Harga Aurora.

Meskipun volume klaster Aurora dapat menaikkan skala ukurannya menjadi banyak tebibyte, Anda hanya dikenai biaya untuk ruang yang Anda gunakan dalam volume. Mekanisme untuk menentukan ruang penyimpanan yang ditagih bergantung pada versi klaster Aurora Anda.

- Ketika data Aurora dihapus dari volume klaster, keseluruhan ruang yang ditagih berkurang sebesar jumlah yang sebanding. Perilaku pengubahan ukuran dinamis ini terjadi saat ruang tabel dasar dihapus atau disusun ulang agar membutuhkan lebih sedikit ruang. Dengan demikian, Anda dapat mengurangi biaya penyimpanan dengan menghapus tabel dan basis data yang tidak lagi Anda butuhkan. Pengubahan ukuran dinamis berlaku untuk versi Aurora tertentu. Berikut adalah versi Aurora yang memungkinkan volume klaster berubah ukurannya secara dinamis saat Anda menghapus data:

Aurora MySQL	• Versi 3 (kompatibel dengan MySQL 8.0): semua versi didukung • Versi 2 (kompatibel dengan MySQL 5.7): 2.11 dan lebih tinggi
Aurora PostgreSQL	Versi yang Didukung
Aurora Serverless v2	Versi yang Didukung
Aurora Serverless v1	Versi yang Didukung

- Dalam versi Aurora yang lebih rendah daripada yang ada di daftar sebelumnya, volume cluster dapat menggunakan kembali ruang yang dibebaskan saat Anda menghapus data, tetapi volume itu sendiri tidak pernah berkurang ukurannya.
- Fitur ini di-deploy secara bertahap ke Wilayah AWS tempat Aurora tersedia. Tergantung Wilayah tempat klaster Anda berada, fitur ini mungkin belum tersedia.

Pengubahan ukuran dinamis berlaku untuk operasi yang secara fisik menghapus atau mengubah ukuran ruang tabel dalam volume klaster. Dengan demikian, hal ini berlaku untuk pernyataan SQL seperti `DROP TABLE`, `DROP DATABASE`, `TRUNCATE TABLE`, dan `ALTER TABLE ... DROP PARTITION`. Hal ini tidak berlaku untuk penghapusan baris menggunakan pernyataan `DELETE`. Jika Anda menghapus sejumlah besar baris dari tabel, Anda dapat menjalankan pernyataan `OPTIMIZE TABLE` Aurora MySQL atau menggunakan ekstensi `pg_repack` Aurora PostgreSQL setelahnya untuk menyusun ulang tabel dan mengubah ukuran volume klaster secara dinamis.

> **Note**
>
> Untuk Aurora MySQL, parameter `innodb_file_per_table` akan memengaruhi cara penyimpanan tabel disusun. Jika tabelnya adalah bagian dari ruang tabel sistem, menghapus tabel tidak akan mengurangi ukuran ruang tabel sistem. Jadi, pastikan untuk mengatur `innodb_file_per_table` ke 1 untuk klaster DB Aurora MySQL agar mengambil keuntungan penuh dari pengubahan ukuran dinamis.
> Untuk Aurora MySQL versi 2.11 dan yang lebih tinggi, ruang meja sementara InnoDB dijatuhkan dan dibuat ulang saat restart. Hal ini akan melepaskan ruang yang ditempati oleh ruang tabel sementara ke sistem, lalu volume klaster akan diubah ukurannya. Untuk

memanfaatkan sepenuhnya fitur pengubahan ukuran dinamis, kami sarankan Anda meningkatkan cluster DB Anda ke Aurora MySQL versi 2.11 atau lebih tinggi.
Fitur pengubahan ukuran dinamis tidak segera merebut kembali ruang saat tabel di ruang tabel dijatuhkan, tetapi secara bertahap dengan kecepatan sekitar 10 TB per hari. Spasi di tablespace sistem tidak direklamasi, karena tablespace sistem tidak pernah dihapus. Ruang kosong yang tidak diklaim ulang di ruang tabel akan digunakan kembali saat operasi membutuhkan ruang di ruang tabel tersebut. Fitur pengubahan ukuran dinamis dapat mengklaim kembali ruang penyimpanan hanya ketika klasternya dalam keadaan tersedia.

Anda dapat memeriksa berapa banyak ruang penyimpanan yang digunakan klaster dengan memantau metrik `VolumeBytesUsed` pada CloudWatch. Untuk informasi selengkapnya tentang penagihan penyimpanan, lihat Cara penyimpanan data Aurora ditagih.

- Di AWS Management Console, Anda dapat melihat angka ini dalam bagan dengan melihat tab `Monitoring` pada halaman detail untuk klaster.
- Dengan AWS CLI, Anda dapat menjalankan perintah yang serupa dengan contoh Linux berikut. Ganti contoh ini dengan nilai Anda sendiri untuk waktu mulai dan berakhir serta nama klaster.

```
aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '6 hours ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" \
  --statistics Average Maximum Minimum \
  --dimensions Name=DBClusterIdentifier,Value=my_cluster_identifier
```

Perintah tersebut menghasilkan output seperti yang berikut ini.

```
{
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T21:25:00+00:00",
            "Average": 182871982080.0,
            "Minimum": 182871982080.0,
            "Maximum": 182871982080.0,
            "Unit": "Bytes"
        }
    ]
}
```

Contoh-contoh berikut menunjukkan bagaimana Anda dapat melacak penggunaan penyimpanan untuk klaster Aurora dari waktu ke waktu menggunakan perintah AWS CLI di sistem Linux. Parameter `--start-time` dan `--end-time` menentukan interval waktu keseluruhan sebagai satu hari. Parameter `--period` meminta pengukuran pada interval satu jam. Tidak masuk akal untuk memilih nilai `--period` yang kecil karena metrik dikumpulkan dalam interval dan tidak terus-menerus. Selain itu, operasi penyimpanan Aurora terkadang berlanjut selama beberapa waktu di latar belakang setelah pernyataan SQL yang relevan selesai.

Contoh pertama menghasilkan output dalam format default JSON. Titik data dihasilkan dalam urutan arbitrer, tidak diurutkan berdasarkan stempel waktu. Anda dapat mengimpor data JSON ini ke alat bagan untuk melakukan pengurutan dan visualisasi.

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id
{
    "Label": "VolumeBytesUsed",
    "Datapoints": [
        {
            "Timestamp": "2020-08-04T19:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-05T00:40:00+00:00",
            "Maximum": 198573719552.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-05T05:40:00+00:00",
            "Maximum": 206827454464.0,
            "Unit": "Bytes"
        },
        {
            "Timestamp": "2020-08-04T17:40:00+00:00",
            "Maximum": 182872522752.0,
            "Unit": "Bytes"
        },
... output omitted ...
```

Contoh ini menghasilkan data yang sama seperti sebelumnya. Parameter `--output` merepresentasikan data dalam format teks biasa yang ringkas. Perintah `aws cloudwatch` mengirimkan output ke perintah `sort`. Parameter `-k` dari perintah `sort` mengurutkan output berdasarkan bidang ketiga, yang merupakan stempel waktu dalam format Waktu Universal Terkoordinasi (UTC).

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '1 day ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_cluster_id \
  --output text | sort -k 3
VolumeBytesUsed
DATAPOINTS  182872522752.0  2020-08-04T17:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T18:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T19:41:00+00:00 Bytes
DATAPOINTS  182872522752.0  2020-08-04T20:41:00+00:00 Bytes
DATAPOINTS  187667791872.0  2020-08-04T21:41:00+00:00 Bytes
DATAPOINTS  190981029888.0  2020-08-04T22:41:00+00:00 Bytes
DATAPOINTS  195587244032.0  2020-08-04T23:41:00+00:00 Bytes
DATAPOINTS  201048915968.0  2020-08-05T00:41:00+00:00 Bytes
DATAPOINTS  205368492032.0  2020-08-05T01:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T02:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T03:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T04:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T05:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T06:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T07:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T08:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T09:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T10:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T11:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T12:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T13:41:00+00:00 Bytes
DATAPOINTS  206827454464.0  2020-08-05T14:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T15:41:00+00:00 Bytes
DATAPOINTS  206833664000.0  2020-08-05T16:41:00+00:00 Bytes
```

Output yang diurutkan menunjukkan berapa banyak penyimpanan yang digunakan pada awal dan akhir periode pemantauan. Anda juga dapat menemukan titik selama periode tersebut saat Aurora mengalokasikan lebih banyak penyimpanan untuk klaster. Contoh berikut menggunakan perintah Linux untuk memformat ulang nilai `VolumeBytesUsed` awal dan akhir sebagai gigabyte (GB) dan gibibyte (GiB). Gigabyte merepresentasikan satuan yang diukur dalam pangkat 10 dan

umumnya digunakan dalam pembahasan penyimpanan untuk hard drive rotasional. Gibibyte merepresentasikan satuan yang diukur dalam pangkat 2. Pengukuran dan batas penyimpanan Aurora biasanya dinyatakan dalam satuan pangkat 2, seperti gibibyte dan tebibyte.

```
$ GiB=$((1024*1024*1024))
$ GB=$((1000*1000*1000))
$ echo "Start: $((182872522752/$GiB)) GiB, End: $((206833664000/$GiB)) GiB"
Start: 170 GiB, End: 192 GiB
$ echo "Start: $((182872522752/$GB)) GB, End: $((206833664000/$GB)) GB"
Start: 182 GB, End: 206 GB
```

Metrik `VolumeBytesUsed` memberi tahu Anda berapa banyak penyimpanan dalam klaster tersebut yang menimbulkan biaya. Jadi, jika memungkinkan sebaiknya minimalkan angka ini. Namun, metrik ini tidak mencakup beberapa penyimpanan yang digunakan Aurora secara internal di klaster dan tidak dikenai biaya. Jika klaster Anda mendekati batas penyimpanan dan mungkin kehabisan ruang, sebaiknya pantau metrik `AuroraVolumeBytesLeftTotal` dan coba maksimalkan angka tersebut. Contoh berikut menjalankan perhitungan yang sama seperti yang sebelumnya, tetapi untuk `AuroraVolumeBytesLeftTotal`, bukan `VolumeBytesUsed`.

```
$ aws cloudwatch get-metric-statistics --metric-name "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Maximum --dimensions
 Name=DBClusterIdentifier,Value=my_old_cluster_id \
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS      140530528288768.0       2023-02-23T19:25:00+00:00       Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((69797067915264 / $TB)) TB remaining for this cluster"
69 TB remaining for this cluster
$ echo "$((69797067915264 / $TiB)) TiB remaining for this cluster"
63 TiB remaining for this cluster
```

Untuk klaster yang menjalankan Aurora MySQL versi 2.09 atau yang lebih tinggi, atau Aurora PostgreSQL, ukuran kosong yang dilaporkan meningkat sebesar `VolumeBytesUsed` saat data ditambahkan dan berkurang saat data dihapus. Contoh berikut menunjukkan cara melakukannya. Laporan ini menunjukkan ukuran penyimpanan maksimum dan minimum untuk klaster dengan interval 15 menit saat tabel dengan data sementara dibuat dan dihapus. Laporan ini mencantumkan nilai maksimum sebelum nilai minimum. Jadi, untuk memahami bagaimana penggunaan penyimpanan berubah dalam interval 15 menit, tafsirkan angkanya dari kanan ke kiri.

```
$ aws cloudwatch get-metric-statistics --metric-name "VolumeBytesUsed" \
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
  --namespace "AWS/RDS" --statistics Maximum Minimum --dimensions
 Name=DBClusterIdentifier,Value=my_new_cluster_id
  --output text | sort -k 4
VolumeBytesUsed
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T20:49:00+00:00 Bytes
DATAPOINTS 14545305600.0 14545305600.0 2020-08-05T21:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 14545305600.0 2020-08-05T21:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:19:00+00:00 Bytes
DATAPOINTS 22022176768.0 22022176768.0 2020-08-05T22:49:00+00:00 Bytes
DATAPOINTS 22022176768.0 15614263296.0 2020-08-05T23:19:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-05T23:49:00+00:00 Bytes
DATAPOINTS 15614263296.0 15614263296.0 2020-08-06T00:19:00+00:00 Bytes
```

Contoh berikut menggunakan klaster yang menjalankan Aurora MySQL versi 2.09 atau yang lebih tinggi, atau Aurora PostgreSQL, untuk menunjukkan bahwa ukuran kosong yang dilaporkan oleh `AuroraVolumeBytesLeftTotal` mencerminkan batas ukuran 128 TiB.

```
$ aws cloudwatch get-metric-statistics --region us-east-1 --metric-name
 "AuroraVolumeBytesLeftTotal" \
  --start-time "$(date -d '4 hours ago')" --end-time "$(date -d 'now')" --period 1800 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBClusterIdentifier,Value=pq-57 \
  --output text | sort -k 3
AuroraVolumeBytesLeftTotal
DATAPOINTS 140515818864640.0 2020-08-05T20:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:26:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-05T21:56:00+00:00 Count
DATAPOINTS 140514866757632.0 2020-08-05T22:26:00+00:00 Count
DATAPOINTS 140511020580864.0 2020-08-05T22:56:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:26:00+00:00 Count
DATAPOINTS 140503168843776.0 2020-08-05T23:56:00+00:00 Count
DATAPOINTS 140515818864640.0 2020-08-06T00:26:00+00:00 Count
$ TiB=$((1024*1024*1024*1024))
$ TB=$((1000*1000*1000*1000))
$ echo "$((140515818864640 / $TB)) TB remaining for this cluster"
140 TB remaining for this cluster
$ echo "$((140515818864640 / $TiB)) TiB remaining for this cluster"
127 TiB remaining for this cluster
```

Penskalaan instans

Anda dapat menskalakan klaster DB Aurora sesuai kebutuhan dengan memodifikasi kelas instans DB untuk setiap instans DB dalam klaster DB ini. Aurora mendukung beberapa kelas instans DB yang dioptimalkan untuk Aurora, bergantung pada kompatibilitas mesin basis data.

Mesin basis data	Penskalaan instans
Amazon Aurora MySQL	Lihat Penskalaan instans DB Aurora MySQL
Amazon Aurora PostgreSQL	Lihat Penskalaan instans DB Aurora PostgreSQL

Penskalaan baca

Anda dapat melakukan penskalaan baca untuk klaster DB Aurora dengan membuat hingga 15 Replika Aurora dalam klaster DB. Setiap Replika Aurora menghasilkan data yang sama dari volume klaster dengan sedikit lag replika—biasanya kurang dari 100 milidetik setelah instans primer menulis pembaruan. Saat lalu lintas baca meningkat, Anda dapat membuat Replika Aurora tambahan dan terhubung langsung ke replika ini untuk mendistribusikan beban baca untuk klaster DB Anda. Replika Aurora tidak harus berasal dari kelas instans DB yang sama dengan instans primer.

Untuk informasi tentang menambahkan Replika Aurora ke klaster DB, lihat Menambahkan Replika Aurora ke klaster DB.

Mengelola koneksi

Jumlah maksimum koneksi yang diizinkan untuk instans DB Aurora ditentukan oleh parameter `max_connections` dalam grup parameter tingkat instans untuk instans DB. Nilai default parameter tersebut bervariasi bergantung pada kelas instans DB yang digunakan untuk instans DB dan kompatibilitas mesin basis data.

Mesin basis data	Nilai default max_connections
Amazon Aurora MySQL	Lihat Koneksi maksimum ke instans DB Aurora MySQL
Amazon Aurora PostgreSQL	Lihat Koneksi maksimum ke instans DB Aurora PostgreSQL

> **Tip**
>
> Jika aplikasi Anda sering membuka dan menutup koneksi, atau membiarkan sejumlah besar koneksi yang lama aktif tetap terbuka, kami menyarankan agar Anda menggunakan Proksi Amazon RDS. Proksi RDS adalah proksi basis data yang sepenuhnya terkelola dan memiliki ketersediaan tinggi yang menggunakan kumpulan koneksi untuk berbagi koneksi basis data dengan aman dan efisien. Untuk mempelajari tentang RDS, lihat Menggunakan Proksi Amazon RDS untuk Aurora.

Mengelola rencana eksekusi kueri

Jika Anda menggunakan manajemen rencana kueri untuk Aurora PostgreSQL, Anda dapat mengontrol rencana mana yang dijalankan pengoptimisasi. Untuk informasi selengkapnya, lihat Mengelola rencana eksekusi kueri untuk Aurora PostgreSQL.

Mengkloning volume untuk klaster DB Amazon Aurora

Dengan menggunakan kloning Aurora, Anda dapat membuat klaster baru yang memiliki halaman data yang sama dengan aslinya, tetapi merupakan volume terpisah dan independen. Prosesnya dirancang agar cepat dan hemat biaya. Klaster baru dengan volume data terkaitnya dikenal sebagai Klon. Membuat klon lebih cepat dan lebih hemat ruang daripada menyalin data secara fisik menggunakan teknik lain, seperti memulihkan snapshot.

Topik

- Gambaran umum kloning Aurora
- Batasan kloning Aurora
- Cara kerja kloning Aurora
- Membuat klon Amazon Aurora
- Kloning lintas akun dengan AWS RAM dan Amazon Aurora

Gambaran umum kloning Aurora

Aurora menggunakan copy-on-write protokol untuk membuat klon. Mekanisme ini menggunakan ruang tambahan minimal untuk membuat klon awal. Ketika klon pertama kali dibuat, Aurora mempertahankan satu salinan data yang digunakan oleh klaster DB Aurora sumber dan klaster DB Aurora yang baru (klon). Penyimpanan tambahan dialokasikan hanya ketika perubahan dibuat pada data (pada volume penyimpanan Aurora) oleh klaster DB Aurora sumber atau klon klaster DB Aurora. Untuk mempelajari lebih lanjut tentang copy-on-write protokol, lihatCara kerja kloning Aurora.

Kloning Aurora sangat berguna untuk menyiapkan lingkungan pengujian dengan cepat menggunakan data produksi Anda, tanpa risiko kerusakan data. Anda dapat menggunakan klon untuk berbagai jenis aplikasi, seperti berikut:

- Bereksperimen dengan potensi perubahan (misalnya, perubahan skema dan perubahan grup parameter) untuk menilai semua dampak.
- Menjalankan operasi sarat beban kerja, seperti mengekspor data atau menjalankan kueri analitis.
- Membuat salinan dari klaster DB produksi Anda untuk pengembangan, pengujian, atau tujuan lainnya.

Anda dapat membuat lebih dari satu klon dari klaster DB Aurora yang sama. Anda juga dapat membuat beberapa klon dari klon lain.

Setelah membuat klon Aurora, Anda dapat mengonfigurasi instans DB Aurora secara berbeda dari klaster DB Aurora sumber. Misalnya, Anda mungkin tidak memerlukan klon untuk tujuan pengembangan guna memenuhi persyaratan ketersediaan tinggi yang sama dengan klaster DB Aurora produksi. Dalam hal ini, Anda dapat mengonfigurasi klon dengan satu instans DB Aurora dan bukan beberapa instans DB yang digunakan oleh klaster DB Aurora.

Saat Anda membuat klon menggunakan konfigurasi penyebaran yang berbeda dari sumbernya, klon dibuat menggunakan versi minor terbaru dari mesin Aurora DB sumber.

Jika Anda membuat klon dari klaster Aurora DB Anda, klon yang dibuat di akun AWS—akun yang sama yang memiliki klaster DB Aurora sumber. Namun, Anda juga dapat berbagi Aurora Serverless v2 dan menyediakan cluster dan klon Aurora DB dengan akun lain. AWS Untuk informasi selengkapnya, lihat Kloning lintas akun dengan AWS RAM dan Amazon Aurora.

Setelah selesai menggunakan klon untuk pengujian, pengembangan, atau tujuan lainnya, Anda dapat menghapusnya.

Batasan kloning Aurora

Kloning Aurora saat ini memiliki batasan berikut:

- Anda dapat membuat klon sebanyak yang Anda inginkan, hingga jumlah maksimum klaster DB yang diizinkan di Wilayah AWS.

  Anda dapat membuat klon menggunakan copy-on-write protokol atau protokol salinan lengkap. Protokol salinan lengkap bertindak seperti point-in-time pemulihan.
- Anda tidak dapat membuat klon di Wilayah AWS yang berbeda dari klaster DB Aurora.
- Anda tidak dapat membuat klon dari klaster DB Aurora tanpa fitur kueri paralel ke klaster yang menggunakan kueri paralel. Untuk memasukkan data ke dalam klaster yang menggunakan kueri paralel, buat snapshot dari klaster asli dan pulihkan ke klaster yang menggunakan fitur opsi kueri pararel.
- Anda tidak dapat membuat klon dari klaster DB Aurora yang tidak memiliki instans DB. Anda hanya dapat mengkloning klaster DB Aurora yang memiliki setidaknya satu instans DB.
- Anda dapat membuat klon di cloud privat virtual (VPC) yang berbeda dari klaster DB Aurora. Jika Anda melakukannya, subnet VPC harus dipetakan ke Zona Ketersediaan yang sama.
- Anda dapat membuat klon terprovisi Aurora dari klaster Aurora DB terprovisi.
- Klaster dengan instans Aurora Serverless v2 mengikuti aturan yang sama dengan klaster terprovisi.

- Untuk Aurora Serverless v1:
  - Anda dapat membuat klon yang disediakan dari cluster DB. Aurora Serverless v1
  - Anda dapat membuat Aurora Serverless v1 klon dari cluster DB Aurora Serverless v1 atau yang disediakan.
  - Anda tidak dapat membuat klon dari Aurora Serverless v1 klaster Aurora DB yang tidak terenkripsi dan disediakan.
  - Kloning lintas akun saat ini tidak mendukung kloning klaster DB Aurora Serverless v1. Untuk informasi selengkapnya, lihat Batasan kloning lintas akun.
  - Klaster DB Aurora Serverless v1 hasil klon memiliki perilaku dan batasan yang sama dengan klaster DB Aurora Serverless v1 lainnya. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Serverless v1.
  - Klaster DB Aurora Serverless v1 selalu terenkripsi. Ketika Anda mengkloning sebuah klaster DB Aurora Serverless v1 ke klaster DB Aurora, klaster DB Aurora DB terprovisi akan dienkripsi. Anda dapat memilih kunci enkripsi, tetapi tidak dapat menonaktifkan enkripsi. Untuk mengkloning dari cluster Aurora DB yang disediakan ke cluster, Anda harus mulai dengan cluster Aurora Serverless v1 Aurora DB yang disediakan terenkripsi.

Cara kerja kloning Aurora

Kloning Aurora beroperasi pada lapisan penyimpanan klaster DB Aurora. Ini menggunakan copy-on-writeprotokol yang cepat dan hemat ruang dalam hal media tahan lama yang mendasarinya yang mendukung volume penyimpanan Aurora. Anda dapat mempelajari selengkapnya tentang volume klaster Aurora dalam Gambaran umum penyimpanan Amazon Aurora.

Topik

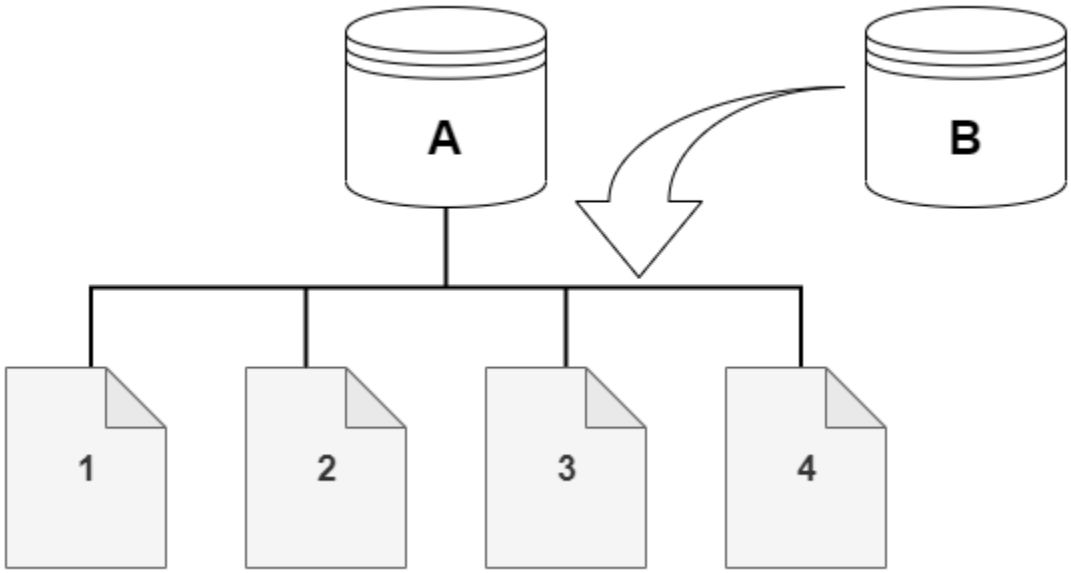
- Memahami copy-on-write protokol
- Menghapus volume klaster sumber

Memahami copy-on-write protokol

Sebuah klaster DB Aurora menyimpan data dalam halaman di volume penyimpanan Aurora yang mendasarinya.

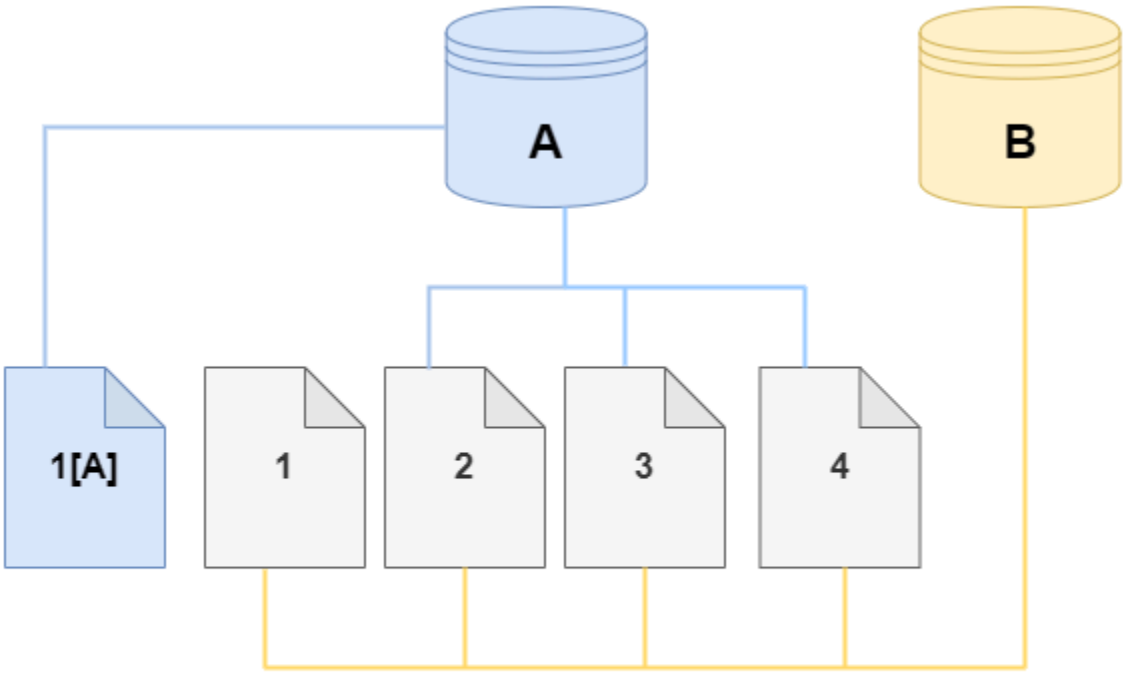
Misalnya, dalam diagram berikut, Anda dapat menemukan klaster DB Aurora (A) yang memiliki empat halaman data, 1, 2, 3, dan 4. Bayangkan bahwa sebuah klon, B, dibuat dari klaster DB Aurora. Saat

klon dibuat, tidak ada data yang disalin. Sebagai gantinya, klon tersebut menunjuk ke kumpulan halaman yang sama sebagai sumber klaster DB Aurora.

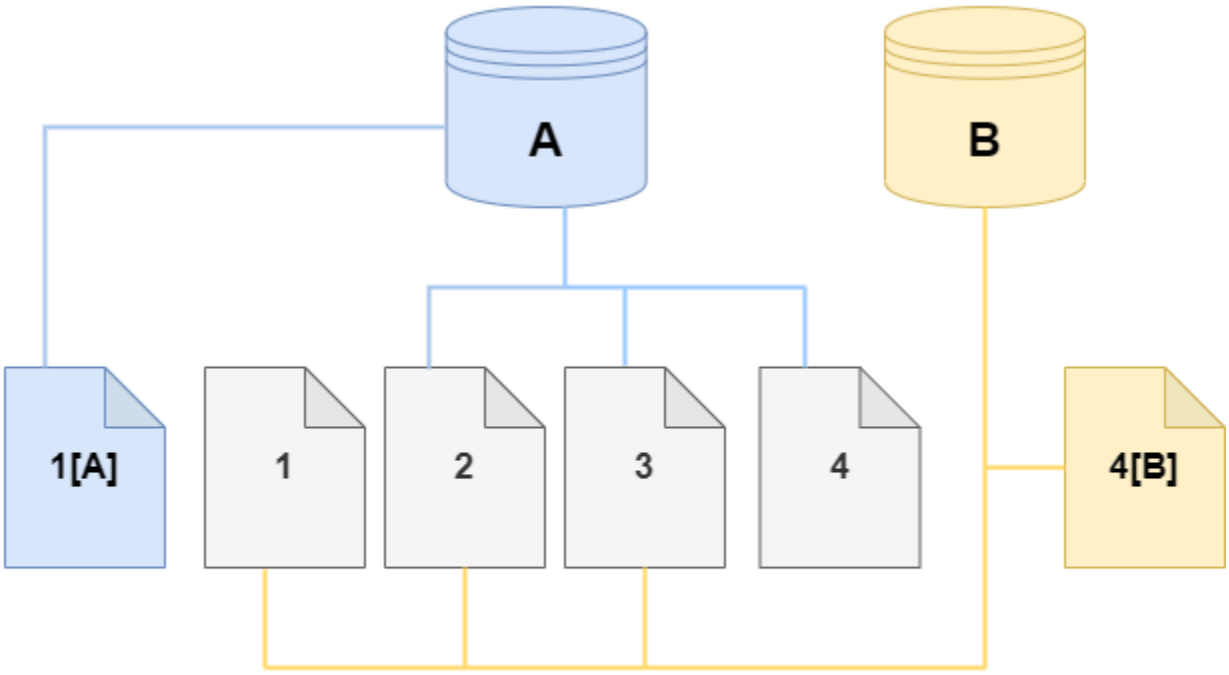


Saat klon dibuat, tidak ada penyimpanan tambahan yang biasanya diperlukan. copy-on-write Protokol menggunakan segmen yang sama pada media penyimpanan fisik sebagai segmen sumber. Penyimpanan tambahan hanya diperlukan jika kapasitas segmen sumber tidak cukup untuk seluruh segmen klon. Jika demikian, segmen sumber disalin ke perangkat fisik lain.

Dalam diagram berikut, Anda dapat menemukan contoh copy-on-write protokol yang sedang beraksi menggunakan cluster A yang sama dan klonnya, B, seperti yang ditunjukkan sebelumnya. Katakanlah bahwa Anda membuat perubahan pada klaster DB Aurora (A) yang menghasilkan perubahan pada data yang disimpan di halaman 1. Alih-alih menulis ke halaman 1 asli, Aurora membuat halaman baru 1[A]. Volume klaster DB Aurora untuk klaster (A) sekarang menunjuk ke halaman 1[A], 2, 3, dan 4, sedangkan klon (B) masih merujuk ke halaman asli.



Pada klon, perubahan dibuat pada halaman 4 di volume penyimpanan. Alih-alih menulis ke halaman 4 asli, Aurora membuat halaman baru 4[B]. Klon sekarang menunjuk ke halaman 1, 2, 3, dan halaman 4[B], sementara klaster (A) terus menunjuk ke 1[A], 2, 3, dan 4.



Saat lebih banyak perubahan terjadi seiring waktu di volume klaster DB Aurora sumber dan klon, diperlukan lebih banyak penyimpanan untuk menangkap dan menyimpan perubahan tersebut.

Menghapus volume klaster sumber

Saat Anda menghapus volume klaster sumber yang memiliki satu atau beberapa klon yang berkaitan dengannya, klon ini tidak akan terpengaruh. Klon terus menunjuk ke halaman yang sebelumnya dimiliki oleh volume klaster sumber.

Membuat klon Amazon Aurora

Anda dapat membuat klon di akun AWS yang sama dengan klaster DB Aurora sumber. Untuk melakukannya, Anda dapat menggunakan AWS Management Console atau AWS CLI dan prosedur berikut.

Untuk mengizinkan akun AWS lain membuat klon atau untuk berbagi klon dengan akun AWS lain, gunakan prosedur dalam Kloning lintas akun dengan AWS RAM dan Amazon Aurora.

Konsol

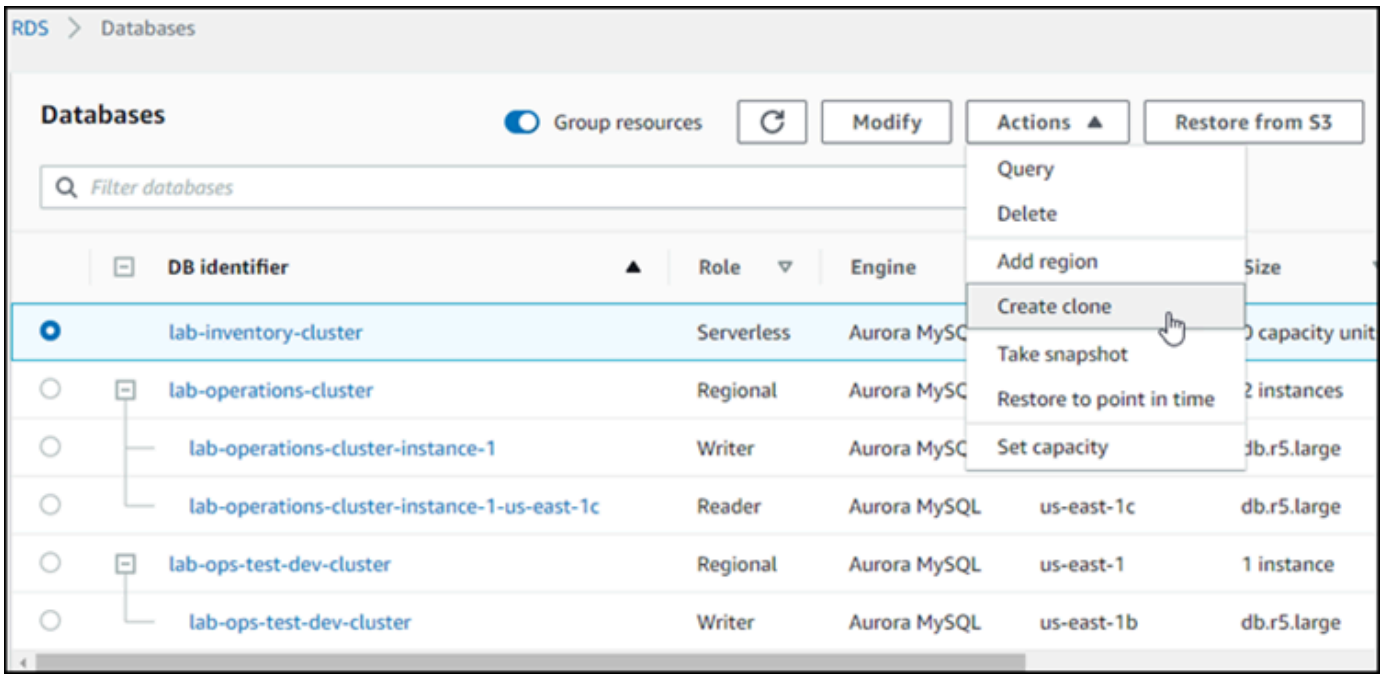
Prosedur berikut menjelaskan cara mengkloning klaster DB Aurora menggunakan AWS Management Console.

Pembuatan klon menggunakan AWS Management Console akan menghasilkan klaster DB Aurora dengan satu instans DB Aurora.

Petunjuk ini berlaku untuk klaster DB yang dimiliki oleh akun AWS yang sama yang membuat klon. Jika klaster DB dimiliki oleh akun AWS berbeda, lihat Kloning lintas akun dengan AWS RAM dan Amazon Aurora.

Untuk membuat klon klaster DB yang dimiliki oleh akun AWS Anda menggunakan AWS Management Console

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB Aurora Anda dari daftar, dan untuk Tindakan pilih Buat klon.



   Halaman Buat klon terbuka, yang memungkinkan Anda mengonfigurasi Pengaturan, Konektivitas, dan opsi lain untuk klon klaster DB Aurora.
4. Untuk Pengidentifikasi instans DB, masukkan nama yang ingin Anda berikan ke klaster DB Aurora Anda.

5. Untuk cluster Aurora Serverless v1 DB, pilih Provisioned atau Serverless untuk tipe Kapasitas.

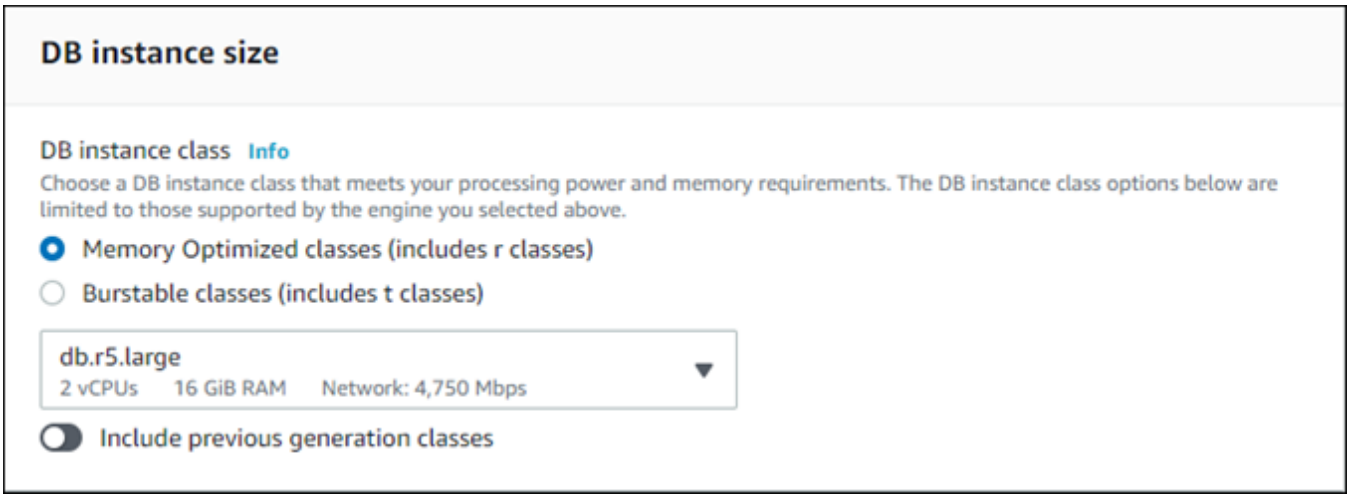
   Anda dapat memilih Nirserver hanya jika klaster DB Aurora sumber adalah klaster DB Aurora Serverless v1 atau klaster DB Aurora terprovisi yang dienkripsi.

6. Untuk Aurora Serverless v2 atau klaster DB yang disediakan, pilih salah satu Aurora I/O-Optimizedatau Aurora Standarduntuk konfigurasi penyimpanan Cluster.

   Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.

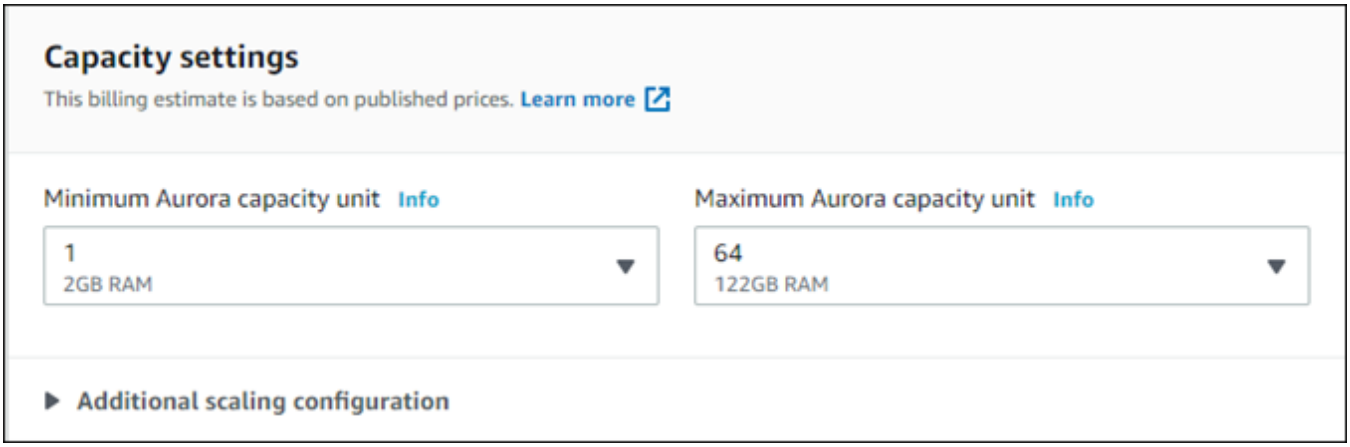
7. Pilih ukuran instans DB atau kapasitas klaster DB:

   - Untuk klon yang disediakan, pilih kelas instans DB.



   Anda dapat menerima pengaturan yang disediakan, atau Anda dapat menggunakan kelas instans DB yang berbeda untuk klon Anda.

   - Untuk Aurora Serverless v1 atau Aurora Serverless v2 klon, pilih pengaturan Kapasitas.



Anda dapat menerima pengaturan yang disediakan, atau Anda dapat mengubahnya untuk klon Anda.

8. Pilih pengaturan lain sesuai kebutuhan untuk klon Anda. Untuk mempelajari selengkapnya tentang klaster DB Aurora dan pengaturan instans, lihat Membuat klaster DB Amazon Aurora.
9. Pilih Buat klon.

Ketika dibuat, klon akan tercantum dengan klaster DB Aurora Anda lainnya di bagian Basis data pada konsol dan menampilkan statusnya saat ini. Klon Anda siap digunakan ketika statusnya Tersedia.

AWS CLI

Agar dapat menggunakan AWS CLI untuk kloning klaster DB Aurora Anda, diperlukan beberapa langkah.

Perintah `restore-db-cluster-to-point-in-time` AWS CLI yang Anda gunakan akan menghasilkan klaster DB Aurora kosong dengan 0 instans DB Aurora. Artinya, perintah tersebut hanya memulihkan klaster DB Aurora, bukan instans DB untuk klaster tersebut. Anda melakukannya secara terpisah setelah klon tersedia. Dua langkah dalam proses ini adalah sebagai berikut:

1. Buat klon dengan menggunakan perintah restore-db-cluster-to- point-in-time CLI. Parameter yang Anda gunakan dengan perintah ini mengontrol jenis kapasitas dan detail lain dari klaster DB Aurora kosong (klon) yang dibuat.
2. Buat instance Aurora DB untuk klon dengan menggunakan perintah CLI create-db-instanceuntuk membuat ulang instans Aurora DB di cluster Aurora DB yang dipulihkan.

Topik

- Membuat klon
- Memeriksa status dan mendapatkan detail klon
- Membuat instans DB Aurora untuk klon Anda
- Parameter yang digunakan untuk kloning

Membuat klon

Parameter tertentu yang Anda teruskan ke perintah CLI `restore-db-cluster-to-point-in-time` akan bervariasi. Apa yang Anda teruskan tergantung pada jenis mode mesin klaster DB sumber—Serverless atau Terprovisi—dan jenis klon yang ingin Anda buat.

Untuk membuat klon dari mode mesin yang sama sebagai klaster DB Aurora sumber

- Gunakan perintah CLI `restore-db-cluster-to-point-in-time` dan tentukan nilai untuk parameter berikut:
  - `--db-cluster-identifier` – Pilih nama yang bermakna untuk klon anda. Anda memberi nama klon saat Anda menggunakan perintah restore-db-cluster-to- point-in-time CLI. Anda kemudian meneruskan nama klon dalam perintah create-db-instanceCLI.
  - `--restore-type` – Gunakan `copy-on-write` untuk membuat klon dari klaster DB sumber. Tanpa parameter ini, `restore-db-cluster-to-point-in-time` akan memulihkan klaster DB Aurora dan bukan membuat klon.
  - `--source-db-cluster-identifier` – Gunakan nama klaster DB Aurora sumber yang ingin Anda kloning.
  - `--use-latest-restorable-time`— Nilai ini menunjuk ke data volume terbaru yang dapat dipulihkan untuk cluster DB sumber. Gunakan untuk membuat klon.

Contoh berikut membuat klon bernama `my-clone` dari klaster bernama `my-source-cluster`.

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

Untuk Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

Perintah tersebut menghasilkan objek JSON yang berisi detail klon. Periksa untuk memastikan bahwa klaster DB yang Anda kloning tersedia sebelum mencoba membuat instans DB untuk klon Anda. Untuk informasi selengkapnya, lihat Memeriksa status dan mendapatkan detail klon.

Untuk membuat klon dengan mode mesin yang berbeda dari sumber Aurora DB cluster

- Gunakan perintah CLI `restore-db-cluster-to-point-in-time` dan tentukan nilai untuk parameter berikut:
  - `--db-cluster-identifier` – Pilih nama yang bermakna untuk klon anda. Anda memberi nama klon saat Anda menggunakan perintah restore-db-cluster-to- point-in-time CLI. Anda kemudian meneruskan nama klon dalam perintah create-db-instanceCLI.
  - `--source-db-cluster-identifier` – Gunakan nama klaster DB Aurora sumber yang ingin Anda kloning.
  - `--restore-type` – Gunakan `copy-on-write` untuk membuat klon dari klaster DB sumber. Tanpa parameter ini, `restore-db-cluster-to-point-in-time` akan memulihkan klaster DB Aurora dan bukan membuat klon.
  - `--use-latest-restorable-time`— Nilai ini menunjuk ke data volume terbaru yang dapat dipulihkan untuk cluster DB sumber. Gunakan untuk membuat klon.
  - `--engine-mode`— (Opsional) Gunakan parameter ini hanya untuk membuat klon yang dari jenis yang berbeda dari sumber Aurora DB cluster. Pilih nilai yang akan diteruskan dengan `--engine-mode` sebagai berikut:
    - Gunakan `provisioned` untuk membuat klon klaster DB Aurora terprovisi dari klaster DB Aurora Serverless.
    - Gunakan `serverless` untuk membuat klon klaster DB Aurora Serverless v1 dari klaster DB Aurora terprovisi. Saat Anda menentukan mode `serverless` mesin, Anda juga dapat memilih`--scaling-configuration`.
  - `--scaling-configuration`— (Opsional) Gunakan dengan `--engine-mode serverless` untuk mengkonfigurasi kapasitas minimum dan maksimum untuk Aurora Serverless v1 klon. Jika Anda tidak menggunakan parameter ini, Aurora membuat klon menggunakan nilai kapasitas default untuk mesin DB.
  - `--serverless-v2-scaling-configuration`— (Opsional) Gunakan parameter ini untuk mengonfigurasi kapasitas minimum dan maksimum untuk Aurora Serverless v2 klon. Jika Anda tidak menggunakan parameter ini, Aurora membuat klon menggunakan nilai kapasitas default untuk mesin DB.

Contoh berikut membuat Aurora Serverless v1 klon bernama`my-clone`, dari cluster Aurora DB yang disediakan bernama. `my-source-cluster` Klaster DB Aurora terprovisi akan dienkripsi.

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier my-source-cluster \
    --db-cluster-identifier my-clone \
    --engine-mode serverless \
    --scaling-configuration MinCapacity=8,MaxCapacity=64 \
    --restore-type copy-on-write \
    --use-latest-restorable-time
```

Untuk Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier my-source-cluster ^
    --db-cluster-identifier my-clone ^
    --engine-mode serverless ^
    --scaling-configuration MinCapacity=8,MaxCapacity=64 ^
    --restore-type copy-on-write ^
    --use-latest-restorable-time
```

Perintah ini menghasilkan objek JSON yang berisi detail klon yang Anda butuhkan untuk membuat instans DB. Anda tidak dapat melakukannya sampai klon (klaster DB Aurora kosong) memiliki status Tersedia.

> **Note**
>
> Perintah restore-db-cluster-to- point-in-time AWS CLI hanya mengembalikan cluster DB, bukan instance DB untuk cluster DB itu. Anda harus memanggil create-db-instanceperintah untuk membuat instance DB untuk cluster DB yang dipulihkan, menentukan pengidentifikasi cluster DB yang dipulihkan di. `--db-cluster-identifier` Anda dapat membuat instans DB hanya setelah perintah `restore-db-cluster-to-point-in-time` selesai dan klaster DB tersedia.

Misalnya, anggap Anda memiliki klaster bernama `tpch100g` yang ingin Anda kloning. Contoh Linux berikut membuat klaster yang dikloning bernama `tpch100g-clone` dan instans primer bernama `tpch100g-clone-instance` untuk klaster baru. Anda tidak perlu menyediakan beberapa parameter, seperti `--master-username` dan `--master-user-password`. Aurora secara otomatis menentukannya dari klaster asli. Anda perlu menentukan mesin DB yang akan digunakan. Dengan demikian, uji coba klaster baru untuk menentukan nilai yang tepat yang akan digunakan untuk parameter `--engine`.

```
$ aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier tpch100g \
  --db-cluster-identifier tpch100g-clone \
  --restore-type copy-on-write \
  --use-latest-restorable-time

$ aws rds describe-db-clusters \
  --db-cluster-identifier tpch100g-clone \
    --query '*[].[Engine]' \
    --output text
aurora-mysql

$ aws rds create-db-instance \
  --db-instance-identifier tpch100g-clone-instance \
  --db-cluster-identifier tpch100g-clone \
  --db-instance-class db.r5.4xlarge \
  --engine aurora-mysql
```

Memeriksa status dan mendapatkan detail klon

Anda dapat menggunakan perintah berikut untuk memeriksa status klaster DB kosong yang baru dibuat.

```
$ aws rds describe-db-clusters --db-cluster-identifier my-clone --query '*[].[Status]'
 --output text
```

Atau Anda dapat memperoleh status dan nilai-nilai lain yang Anda butuhkan untuk membuat instans DB untuk klon Anda dengan menggunakan kueri AWS CLI berikut.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-clusters --db-cluster-identifier my-clone \
  --query '*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}'
```

Untuk Windows:

```
aws rds describe-db-clusters --db-cluster-identifier my-clone ^
  --query "*[].
{Status:Status,Engine:Engine,EngineVersion:EngineVersion,EngineMode:EngineMode}"
```

Kueri ini menghasilkan output seperti yang berikut ini:

```
[
  {
        "Status": "available",
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.04.1",
        "EngineMode": "provisioned"
    }
]
```

Membuat instans DB Aurora untuk klon Anda

Gunakan perintah create-db-instanceCLI untuk membuat instance DB untuk klon Anda Aurora Serverless v2 atau yang disediakan. Anda tidak membuat instance DB untuk Aurora Serverless v1 klon.

Instans DB mewarisi `--master-username` dan `--master-user-password` properti dari cluster DB sumber.

Contoh berikut membuat instance DB untuk klon yang disediakan.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
    --db-instance-identifier my-new-db \
    --db-cluster-identifier my-clone \
    --db-instance-class db.r5.4xlarge \
    --engine aurora-mysql
```

Untuk Windows:

```
aws rds create-db-instance ^
    --db-instance-identifier my-new-db ^
    --db-cluster-identifier my-clone ^
    --db-instance-class db.r5.4xlarge ^
    --engine aurora-mysql
```

Parameter yang digunakan untuk kloning

Tabel berikut merangkum berbagai parameter yang digunakan dengan `restore-db-cluster-to-point-in-time` untuk mengkloning klaster DB Aurora.

Parameter	Deskripsi
`--source-db-cluster-identifier`	Pilih parameter ini untuk menggunakan nama klaster DB Aurora sumber yang ingin Anda kloning.
`--db-cluster-identifier`	Pilih nama yang berarti untuk klon Anda saat Anda membuatnya dengan `restore-db-cluster-to-point-in-time` perintah. Kemudian, Anda meneruskan nama ini ke perintah `create-db-instance` .
`--restore-type`	Tentukan `copy-on-write` sebagai `--restore-type` untuk membuat klon klaster DB sumber dan bukan memulihkan klaster DB Aurora sumber.
`--use-latest-restorable-time`	Nilai ini menunjuk ke data volume terbaru yang dapat dipulihkan untuk cluster DB sumber. Gunakan untuk membuat klon.
`--engine-mode`	Gunakan parameter ini untuk membuat klon yang memiliki tipe berbeda dari cluster Aurora DB sumber, dengan salah satu nilai berikut: • Gunakan `provisioned` untuk membuat provisioned atau Aurora Serverless v2 clone dari cluster DB. Aurora Serverless v1 • Gunakan `serverless` untuk membuat Aurora Serverless v1 klon dari cluster yang disediakan atau Aurora Serverless v2 DB. Saat Anda menentukan mode `serverless` mesin, Anda juga dapat memilih`--scaling-configuration` .
`--scaling-configuration`	Gunakan parameter ini untuk mengonfigurasi kapasitas minimum dan maksimum untuk Aurora Serverless v1 klon. Jika Anda tidak menentukan parameter ini, Aurora membuat klon menggunakan nilai kapasitas default untuk mesin DB.
`--serverless-v2-scaling-configuration`	Gunakan parameter ini untuk mengonfigurasi kapasitas minimum dan maksimum untuk Aurora Serverless v2 klon. Jika Anda tidak menentukan parameter ini, Aurora membuat klon menggunakan nilai kapasitas default untuk mesin DB.

Kloning lintas akun dengan AWS RAM dan Amazon Aurora

Dengan menggunakan AWS Resource Access Manager (AWS RAM) dengan Amazon Aurora, Anda dapat berbagi klaster DB Aurora dan klon milik akun AWS Anda dengan akun AWS atau organisasi lain. Kloning lintas akun lebih cepat daripada membuat dan memulihkan snapshot basis data. Anda dapat membuat klon dari salah satu klaster DB Aurora Anda lalu membagikan klon tersebut. Atau Anda dapat berbagi klaster DB Aurora Anda dengan akun AWS lain dan membiarkan pemegang akun membuat klon. Pendekatan yang Anda pilih tergantung pada kasus penggunaan Anda.

Misalnya, Anda mungkin perlu berbagi klon basis data keuangan Anda secara rutin dengan tim audit internal organisasi Anda. Dalam hal ini, tim audit Anda memiliki akun AWS untuk aplikasi yang digunakannya. Anda dapat memberikan izin pada akun AWS tim audit untuk mengakses klaster DB Aurora Anda dan mengkloningnya sesuai kebutuhan.

Di sisi lain, jika vendor luar mengaudit data keuangan Anda, Anda mungkin lebih memilih untuk membuat klon sendiri. Anda kemudian memberi vendor luar tersebut akses ke klon saja.

Anda juga dapat menggunakan kloning lintas akun untuk mendukung banyak kasus penggunaan yang sama untuk kloning dalam akun AWS yang sama, seperti pengembangan dan pengujian. Misalnya, organisasi Anda mungkin menggunakan akun AWS berbeda untuk produksi, pengembangan, pengujian, dan sebagainya. Untuk informasi selengkapnya, lihat Gambaran umum kloning Aurora.

Dengan demikian, Anda sebaiknya berbagi klon dengan akun AWS lain atau mengizinkan akun AWS untuk membuat klon dari klaster DB Aurora Anda. Dalam kedua kasus ini, mulailah dengan menggunakan AWS RAM untuk membuat objek yang akan dibagikan. Untuk informasi lengkap tentang berbagi sumber daya AWS antar-akun AWS, lihat Panduan Pengguna AWS RAM.

Membuat klon lintas akun memerlukan tindakan dari akun AWS yang memiliki klaster asli, dan akun AWS yang membuat klon. Pertama, pemilik klaster asli memodifikasi klaster untuk mengizinkan satu atau beberapa akun lain mengkloningnya. Jika salah satu akun ada di organisasi AWS berbeda, AWS akan membuat undangan berbagi. Akun lain tersebut harus menerima undangan sebelum melanjutkan. Kemudian, setiap akun terotorisasi dapat mengkloning klaster. Selama proses ini, klaster diidentifikasi berdasarkan Amazon Resource Name (ARN) yang unik.

Seperti kloning dalam akun AWS yang sama, ruang penyimpanan tambahan hanya digunakan jika perubahan dibuat ke data oleh sumber atau klon. Biaya untuk penyimpanan kemudian diterapkan pada waktu itu. Jika klaster sumber dihapus, biaya penyimpanan didistribusikan secara merata di antara klon yang tersisa.

Topik

- Batasan kloning lintas akun
- Mengizinkan akun AWS lain mengkloning klaster Anda
- Mengkloning klaster yang dimiliki oleh akun AWS lain

Batasan kloning lintas akun

Kloning lintas akun Aurora memiliki batasan sebagai berikut:

- Anda tidak dapat mengkloning sebuah klaster Aurora Serverless v1 ke seluruh akun AWS.
- Anda tidak dapat melihat atau menerima undangan ke sumber daya yang dibagikan menggunakan AWS Management Console. Gunakan AWS CLI, API Amazon RDS, atau AWS RAM untuk melihat dan menerima undangan ke sumber daya yang dibagikan.
- Anda dapat membuat hanya satu klon baru dari klon yang telah dibagikan dengan akun AWS Anda.
- Anda tidak dapat berbagi sumber daya (klon atau klaster DB Aurora) yang telah dibagikan dengan akun AWS Anda.
- Anda dapat membuat maksimum 15 klon lintas akun dari setiap klaster DB Aurora.
- Masing-masing dari 15 klon lintas akun harus dimiliki oleh akun AWS yang berbeda. Artinya, Anda hanya dapat membuat satu klon lintas akun dari sebuah klaster dalam akun AWS apa pun.
- Setelah Anda mengkloning klaster, klaster asli dan klonnya dianggap sama untuk tujuan memberlakukan batasan pada klon lintas akun. Anda tidak dapat membuat klon lintas akun dari klaster asli dan klaster yang dikloning dalam akun AWS yang sama. Jumlah total klon lintas akun untuk klaster asli dan klonnya tidak boleh melebihi 15.
- Anda tidak dapat berbagi klaster DB Aurora dengan akun AWS kecuali jika klaster tersebut berada dalam status `ACTIVE`.
- Anda tidak dapat mengubah nama klaster DB Aurora yang telah dibagikan dengan akun AWS lain.
- Anda tidak dapat membuat klon lintas akun dari klaster yang dienkripsikan dengan kunci RDS default.
- Anda tidak dapat membuat klon yang tidak dienkripsi dalam satu akun AWS dari klaster DB Aurora terenkripsi yang telah dibagikan oleh akun AWS lain. Pemilik klaster harus memberikan izin untuk mengakses AWS KMS key klaster sumber. Namun, Anda dapat menggunakan kunci yang berbeda saat Anda membuat klon.

Mengizinkan akun AWS lain mengkloning klaster Anda

Untuk mengizinkan akun AWS lain mengkloning klaster yang Anda miliki, gunakan AWS RAM untuk mengatur izin berbagi. Tindakan tersebut juga akan mengirimkan undangan ke akun-akun lain yang ada di organisasi AWS berbeda.

Untuk prosedur berbagi sumber daya yang Anda miliki di konsol AWS RAM, lihat Berbagi sumber daya yang dimiliki oleh Anda dalam Panduan Pengguna AWS RAM.

Topik

- Memberikan izin ke akun AWS lain untuk mengkloning klaster Anda
- Memeriksa apakah klaster milik Anda dibagikan dengan akun AWS lainnya

Memberikan izin ke akun AWS lain untuk mengkloning klaster Anda

Jika klaster yang Anda bagikan dienkripsi, Anda juga berbagi AWS KMS key untuk klaster tersebut. Anda dapat mengizinkan pengguna atau peran AWS Identity and Access Management (IAM) dalam satu akun AWS untuk menggunakan kunci KMS di akun yang berbeda.

Untuk melakukannya, Anda pertama-tama menambahkan akun eksternal (pengguna root) ke kebijakan kunci untuk kunci KMS melalui AWS KMS. Anda tidak menambahkan pengguna atau peran IAM individual ke kebijakan kunci, hanya akun eksternal yang memilikinya. Anda hanya dapat berbagi kunci KMS yang Anda buat, bukan kunci layanan RDS default. Untuk informasi tentang kontrol akses untuk kunci KMS, lihat Autentikasi dan kontrol akses untuk AWS KMS.

Konsol

Untuk memberikan izin untuk mengkloning klaster Anda

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB yang ingin Anda bagikan untuk melihat halaman Detail, lalu pilih tab Konektivitas & keamanan.
4. Di bagian Berbagi klaster DB dengan akun AWS lain, masukkan ID akun numerik untuk akun AWS yang ingin Anda izinkan untuk mengkloning klaster ini. Untuk ID akun dalam organisasi yang sama, Anda dapat mulai mengetik di kotak, kemudian memilih dari menu.

> **Important**
>
> Dalam beberapa kasus, Anda mungkin ingin akun yang tidak berada dalam organisasi AWS yang sama dengan akun Anda untuk mengkloning klaster. Dalam kasus ini, demi alasan keamanan, konsol tidak melaporkan siapa yang memiliki ID akun tersebut atau apakah akun tersebut ada.
> Berhati-hatilah saat memasukkan nomor akun yang tidak berada dalam organisasi AWS yang sama dengan akun AWS Anda. Segera verifikasi bahwa Anda berbagi kepada akun yang dimaksud.

5. Di halaman konfirmasi, verifikasi bahwa ID akun yang Anda tentukan sudah benar. Masukkan `share` di kotak konfirmasi untuk mengonfirmasi.

   Di halaman Detail, akan muncul entri yang menunjukkan ID akun AWS yang ditentukan di Akun yang dibagikan klaster DB ini. Kolom Status awalnya menunjukkan status Tertunda.

6. Hubungi pemilik akun AWS lain, atau masuk ke akun tersebut jika Anda memiliki keduanya. Instruksikan pemilik akun lain untuk menerima undangan berbagi dan mengkloning klaster DB, sebagaimana dijelaskan sebagai berikut.

AWS CLI

Untuk memberikan izin untuk mengkloning klaster Anda

1. Kumpulkan informasi untuk parameter yang diperlukan. Anda memerlukan ARN untuk klaster dan ID numerik untuk akun AWS lainnya.
2. Jalankan perintah CLI AWS RAM `create-resource-share`.

   Untuk Linux, macOS, atau Unix:

```
aws ram create-resource-share --name descriptive_name \
  --region region \
  --resource-arns cluster_arn \
  --principals other_account_ids
```

   Untuk Windows:

```
aws ram create-resource-share --name descriptive_name ^
```

```
  --region region ^
  --resource-arns cluster_arn ^
  --principals other_account_ids
```

Untuk menyertakan banyak ID akun untuk parameter `--principals`, pisahkan ID dari satu sama lainnya dengan spasi. Untuk menentukan apakah akun AWS yang diizinkan bisa berada di luar organisasi Anda, termasuk parameter `--allow-external-principals` atau `--no-allow-external-principals` untuk `create-resource-share`.

API AWS RAM

Untuk memberikan izin untuk mengkloning klaster Anda

1. Kumpulkan informasi untuk parameter yang diperlukan. Anda memerlukan ARN untuk klaster dan ID numerik untuk akun AWS lainnya.
2. Panggil operasi AWS RAM API CreateResourceShare, dan tentukan nilai berikut:
   - Tentukan ID akun untuk satu atau beberapa akun AWS sebagai parameter `principals`.
   - Tentukan ARN untuk satu atau beberapa klaster DB Aurora sebagai parameter `resourceArns`.
   - Tentukan apakah ID akun yang diizinkan bisa berada di luar organisasi AWS Anda atau tidak dengan menyertakan nilai Boolean untuk parameter `allowExternalPrincipals`.

Membuat ulang klaster yang menggunakan kunci RDS default

Jika klaster terenkripsi yang ingin Anda bagikan menggunakan kunci RDS default, pastikan untuk membuat ulang klaster tersebut. Untuk melakukannya, buat snapshot manual dari klaster DB Anda, gunakan AWS KMS key, lalu pulihkan klaster tersebut ke klaster baru. Kemudian, bagikan klaster baru ini. Untuk melakukan proses ini, lakukan langkah-langkah berikut.

Untuk membuat ulang klaster terenkripsi yang menggunakan kunci RDS default

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Snapshot dari panel navigasi.
3. Pilih snapshot Anda.
4. Untuk Tindakan, pilih Salin Snapshot, lalu pilih Aktifkan enkripsi.

5. Untuk AWS KMS key, pilih kunci enkripsi baru yang ingin Anda gunakan.
6. Pulihkan snapshot yang telah disalin. Untuk melakukannya, ikuti prosedur dalam Memulihkan dari snapshot klaster DB. Instans DB baru akan menggunakan kunci enkripsi baru Anda.
7. (Opsional) Hapus klaster DB lama jika Anda tidak lagi membutuhkannya. Untuk melakukannya, ikuti prosedur dalam Menghapus snapshot klaster DB. Sebelum Anda melakukannya, konfirmasi bahwa klaster baru Anda memiliki semua data yang diperlukan dan bahwa aplikasi Anda dapat mengaksesnya dengan sukses.

Memeriksa apakah klaster milik Anda dibagikan dengan akun AWS lainnya

Anda dapat memeriksa apakah pengguna lain memiliki izin untuk berbagi klaster. Tindakan tersebut dapat membantu Anda memahami apakah klaster hampir mencapai batas jumlah maksimum klon lintas akun.

Untuk prosedur berbagi sumber daya menggunakan konsol AWS RAM, lihat Berbagi sumber daya yang dimiliki oleh Anda dalam Panduan Pengguna AWS RAM.

AWS CLI

Untuk mencari tahu apakah klaster milik Anda dibagikan kepada akun AWS lainnya

- Panggil perintah CLI AWS RAM `list-principals`, menggunakan ID akun Anda sebagai pemilik sumber daya dan ARN klaster Anda sebagai ARN sumber daya. Anda dapat melihat semua pembagian dengan perintah berikut. Hasilnya akan menunjukkan akun AWS mana yang diizinkan untuk mengkloning klaster.

```
aws ram list-principals \
    --resource-arns your_cluster_arn \
    --principals your_aws_id
```

API AWS RAM

Untuk mencari tahu apakah klaster milik Anda dibagikan kepada akun AWS lainnya

- Panggil operasi API AWS RAM ListPrincipals. Gunakan ID akun Anda sebagai pemilik sumber daya dan ARN klaster sebagai ARN sumber daya.

Mengkloning klaster yang dimiliki oleh akun AWS lain

Untuk mengkloning klaster yang dimiliki oleh akun AWS lain, gunakan AWS RAM untuk mendapatkan izin membuat klon. Setelah Anda mendapatkan izin yang diperlukan, gunakan prosedur standar untuk mengkloning klaster Aurora.

Anda juga dapat memeriksa apakah klaster yang Anda miliki adalah klon dari klaster yang dimiliki oleh akun AWS yang berbeda.

Untuk prosedur dalam menangani sumber daya yang dimiliki orang lain di konsol AWS RAM, lihat Mengakses sumber daya yang dibagikan kepada Anda dalam Panduan Pengguna AWS RAM.

Topik

- Melihat undangan untuk mengkloning klaster yang dimiliki oleh akun AWS lainnya
- Menerima undangan untuk berbagi klaster yang dimiliki oleh akun AWS lain
- Mengkloning klaster Aurora yang dimiliki oleh akun AWS lain
- Memeriksa apakah klaster DB merupakan klon lintas akun

Melihat undangan untuk mengkloning klaster yang dimiliki oleh akun AWS lainnya

Untuk menggunakan undangan untuk mengkloning klaster yang dimiliki oleh akun AWS di organisasi AWS lain, gunakan AWS CLI, konsol AWS RAM, atau API AWS RAM. Saat ini, Anda tidak dapat melakukan prosedur ini menggunakan konsol Amazon RDS.

Untuk prosedur penggunaan undangan di konsol AWS RAM, lihat Mengakses sumber daya yang dibagikan kepada Anda dalam Panduan Pengguna AWS RAM.

AWS CLI

Untuk melihat undangan untuk mengkloning klaster yang dimiliki oleh akun AWS lainnya

1. Jalankan perintah CLI AWS RAM `get-resource-share-invitations`.

   ```
   aws ram get-resource-share-invitations --region region_name
   ```

   Hasil dari perintah sebelumnya menunjukkan semua undangan untuk klaster klon, termasuk semua yang telah Anda terima atau ditolak.

2. (Opsional) Filter daftar sehingga Anda hanya melihat undangan yang memerlukan tindakan dari Anda. Untuk melakukannya, tambahkan parameter `--query 'resourceShareInvitations[?status==`PENDING`]'`.

API AWS RAM

Untuk melihat undangan untuk mengkloning klaster yang dimiliki oleh akun AWS lainnya

1. Panggil operasi API AWS RAM `GetResourceShareInvitations`. Operasi ini akan menampilkan semua undangan tersebut, termasuk semua yang telah Anda terima atau tolak.
2. (Opsional) Temukan hanya undangan yang memerlukan tindakan dari Anda dengan memeriksa `status` dengan nilai `PENDING` pada bidang `resourceShareAssociations` yang dihasilkan.

Menerima undangan untuk berbagi klaster yang dimiliki oleh akun AWS lain

Anda dapat menerima undangan untuk berbagi klaster yang dimiliki oleh akun AWS lain yang ada di organisasi AWS berbeda. Untuk Menangani undangan ini, gunakan AWS CLI, AWS RAM, dan API RDS, atau konsol AWS RAM. Saat ini, Anda tidak dapat melakukan prosedur ini menggunakan konsol RDS.

Untuk prosedur penggunaan undangan di konsol AWS RAM, lihat Mengakses sumber daya yang dibagikan kepada Anda dalam Panduan Pengguna AWS RAM.

AWS CLI

Untuk menerima undangan untuk berbagi klaster dari akun AWS lain

1. Temukan ARN undangan dengan menjalankan perintah CLI AWS RAM `get-resource-share-invitations`, seperti yang ditunjukkan sebelumnya.
2. Terima undangan dengan memanggil perintah CLI AWS RAM `accept-resource-share-invitation`, seperti yang ditunjukkan berikut ini.

   Untuk Linux, macOS, atau Unix:

   ```
   aws ram accept-resource-share-invitation \
     --resource-share-invitation-arn invitation_arn \
     --region region
   ```

   Untuk Windows:

```
aws ram accept-resource-share-invitation ^
  --resource-share-invitation-arn invitation_arn ^
  --region region
```

AWS RAM dan API RDS

Untuk menerima undangan untuk membagikan klaster orang lain

1. Temukan ARN undangan dengan memanggil operasi API AWS RAM `GetResourceShareInvitations`, seperti yang ditunjukkan sebelumnya.
2. Lewatkan ARN tersebut sebagai parameter `resourceShareInvitationArn` untuk operasi API RDS AcceptResourceShareInvitation.

Mengkloning klaster Aurora yang dimiliki oleh akun AWS lain

Setelah Anda menerima undangan dari akun AWS yang memiliki klaster DB, seperti yang ditunjukkan sebelumnya, Anda dapat mengkloning klaster tersebut.

Konsol

Untuk mengkloning klaster Aurora yang dimiliki oleh akun AWS lain

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.

   Di bagian atas daftar basis data, Anda akan melihat satu atau beberapa item dengan nilai Peran yang menampilkan `Shared from account #account_id`. Untuk alasan keamanan, Anda hanya dapat melihat informasi terbatas tentang klaster asli. Properti yang dapat Anda lihat antara lain adalah mesin dan versi basis data, yang harus sama dalam kloning klaster Anda.
3. Pilih klaster yang akan dikloning.
4. Untuk Tindakan, pilih Buat klon.
5. Ikuti prosedur dalam Konsol untuk menyelesaikan penyiapan klaster yang dikloning.
6. Sesuai kebutuhan, aktifkan enkripsi untuk klaster yang dikloning. Jika klaster tersebut dienkripsi, Anda harus mengaktifkan enkripsi untuk klaster yang dikloning. Akun AWS yang berbagi klaster dengan Anda juga harus berbagi kunci KMS yang digunakan untuk mengenkripsi klaster. Anda

dapat menggunakan kunci KMS yang sama untuk mengenkripsi klon, atau kunci KMS Anda sendiri. Anda tidak dapat membuat klon lintas akun untuk klaster yang dienkripsi dengan kunci KMS default.

Akun yang memiliki kunci enkripsi harus memberikan izin penggunaan kunci tersebut ke akun tujuan melalui kebijakan kunci. Proses ini serupa dengan cara berbagi snapshot terenkripsi, dengan menggunakan kebijakan kunci yang memberikan izin ke akun tujuan untuk menggunakan kunci.

AWS CLI

Untuk mengkloning klaster Aurora yang dimiliki oleh akun AWS lain

1. Terima undangan dari akun AWS yang memiliki klaster DB, seperti yang ditunjukkan sebelumnya.
2. Kloning klaster dengan menentukan ARN lengkap klaster sumber dalam parameter `source-db-cluster-identifier` pada perintah CLI RDS `restore-db-cluster-to-point-in-time`, sebagaimana ditunjukkan berikut ini.

   Jika ARN yang diteruskan sebagai `source-db-cluster-identifier` belum dibagikan, kesalahan yang sama akan ditampilkan seolah-olah klaster tersebut tidak ada.

   Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier=arn:aws:rds:arn_details \
  --db-cluster-identifier=new_cluster_id \
  --restore-type=copy-on-write \
  --use-latest-restorable-time
```

   Untuk Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
  --source-db-cluster-identifier=arn:aws:rds:arn_details ^
  --db-cluster-identifier=new_cluster_id ^
  --restore-type=copy-on-write ^
  --use-latest-restorable-time
```

3. Jika klaster yang Anda kloning terenkripsi, enkripsi klaster yang dikloning dengan menyertakan parameter `kms-key-id`. Nilai `kms-key-id` ini dapat sama dengan yang digunakan untuk mengenkripsi klaster DB asli, atau kunci KMS Anda sendiri. Akun Anda harus memiliki izin untuk menggunakan kunci enkripsi tersebut.

   Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-to-point-in-time \
  --source-db-cluster-identifier=arn:aws:rds:arn_details \
  --db-cluster-identifier=new_cluster_id \
  --restore-type=copy-on-write \
  --use-latest-restorable-time \
  --kms-key-id=arn:aws:kms:arn_details
```

   Untuk Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
  --source-db-cluster-identifier=arn:aws:rds:arn_details ^
  --db-cluster-identifier=new_cluster_id ^
  --restore-type=copy-on-write ^
  --use-latest-restorable-time ^
  --kms-key-id=arn:aws:kms:arn_details
```

   Akun yang memiliki kunci enkripsi harus memberikan izin penggunaan kunci tersebut ke akun tujuan melalui kebijakan kunci. Proses ini serupa dengan cara berbagi snapshot terenkripsi, dengan menggunakan kebijakan kunci yang memberikan izin ke akun tujuan untuk menggunakan kunci. Contoh kebijakan kunci adalah sebagai berikut.

```
{
  "Id": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
```

```
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
  ]
}
```

Note

Perintah restore-db-cluster-to- point-in-time AWS CLI hanya mengembalikan cluster DB, bukan instance DB untuk cluster DB itu. Untuk membuat instance DB untuk cluster DB yang dipulihkan, panggil perintah. create-db-instance Tentukan pengidentifikasi klaster DB yang dipulihkan di `--db-cluster-identifier`.

Anda dapat membuat instans DB hanya setelah perintah `restore-db-cluster-to-point-in-time` selesai dan klaster DB tersedia.

API RDS

Untuk mengkloning klaster Aurora yang dimiliki oleh akun AWS lain

1. Terima undangan dari akun AWS yang memiliki klaster DB, seperti yang ditunjukkan sebelumnya.
2. Kloning klaster dengan menentukan ARN lengkah klaster sumber dalam parameter `SourceDBClusterIdentifier` pada operasi API RDS `RestoreDBClusterToPointInTime`.

   Jika ARN yang diteruskan sebagai `SourceDBClusterIdentifier` belum dibagikan, kesalahan yang sama akan ditampilkan seolah-olah klaster tersebut tidak ada.
3. Jika klaster yang Anda kloning terenkripsi, sertakan parameter `KmsKeyId` untuk mengenkripsi klaster yang dikloning. Nilai `kms-key-id` ini dapat sama dengan yang digunakan untuk mengenkripsi klaster DB asli, atau kunci KMS Anda sendiri. Akun Anda harus memiliki izin untuk menggunakan kunci enkripsi tersebut.

   Saat mengkloning volume, akun tujuan harus memiliki izin untuk menggunakan kunci enkripsi yang digunakan untuk mengenkripsi klaster sumber. Aurora mengenkripsi klaster baru yang dikloning dengan kunci enkripsi yang ditentukan dalam `KmsKeyId`.

   Akun yang memiliki kunci enkripsi harus memberikan izin penggunaan kunci tersebut ke akun tujuan melalui kebijakan kunci. Proses ini serupa dengan cara berbagi snapshot terenkripsi, dengan menggunakan kebijakan kunci yang memberikan izin ke akun tujuan untuk menggunakan kunci. Contoh kebijakan kunci adalah sebagai berikut.

```
{
  "Id": "key-policy-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:Encrypt",
```

```
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::account_id:user/KeyUser",
        "arn:aws:iam::account_id:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
    }
  ]
}
```

> **Note**
>
> Operasi RestoreDB ClusterToPointInTime RDS API hanya mengembalikan cluster DB, bukan instance DB untuk cluster DB tersebut. Untuk membuat instans DB untuk klaster DB yang dipulihkan, invokasi operasi API RDS CreateDBInstance. Tentukan pengidentifikasi klaster DB yang dipulihkan di `DBClusterIdentifier`. Anda dapat membuat instans DB hanya setelah operasi `RestoreDBClusterToPointInTime` selesai dan klaster DB tersedia.

Memeriksa apakah klaster DB merupakan klon lintas akun

Objek `DBClusters` mengidentifikasi apakah setiap klaster merupakan klon lintas akun. Anda dapat melihat klaster yang izin kloning Anda miliki dengan menggunakan opsi `include-shared` saat Anda menjalankan perintah CLI RDS `describe-db-clusters`. Namun, Anda tidak dapat melihat sebagian besar detail konfigurasi untuk klaster tersebut.

AWS CLI

Untuk memeriksa apakah klaster DB merupakan klon lintas akun

- Panggil perintah CLI RDS `describe-db-clusters`.

  Contoh berikut menunjukkan bagaimana klaster DB klon lintas akun yang sebenarnya atau potensial muncul di output `describe-db-clusters`. Untuk klaster yang sudah ada yang dimiliki oleh akun AWS Anda, bidang `CrossAccountClone` akan menunjukkan apakah klaster tersebut adalah klon dari klaster DB yang dimiliki akun AWS lain.

  Dalam beberapa kasus, sebuah entri mungkin memiliki nomor akun AWS yang berbeda dengan nomor di bidang `DBClusterArn`. Dalam hal ini, entri tersebut merepresentasikan klaster yang dimiliki oleh akun AWS yang berbeda dan dapat Anda kloning. Entri tersebut memiliki beberapa bidang selain `DBClusterArn`. Saat membuat klaster yang diklon, tentukan nilai `StorageEncrypted`, `Engine`, dan `EngineVersion` yang sama seperti dalam klaster asli.

```
$aws rds describe-db-clusters --include-shared --region us-east-1
{
  "DBClusters": [
      {
          "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": false,
...
      },
      {
          "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": true,
...
      },
      {
          "StorageEncrypted": false,
          "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0
  ]
```

```
}
```

API RDS

Untuk memeriksa apakah klaster DB merupakan klon lintas akun

- Panggil operasi API RDS DescribeDBClusters.

  Untuk klaster yang sudah ada yang dimiliki oleh akun AWS Anda, bidang `CrossAccountClone` akan menunjukkan apakah klaster tersebut adalah klon dari klaster DB yang dimiliki akun AWS lain. Entri dengan nomor akun AWS berbeda dalam bidang `DBClusterArn` merepresentasikan klaster yang dapat Anda kloning dan yang dimiliki oleh akun AWS lainnya. Entri tersebut memiliki beberapa bidang selain `DBClusterArn`. Saat membuat klaster yang diklon, tentukan nilai `StorageEncrypted`, `Engine`, dan `EngineVersion` yang sama seperti dalam klaster asli.

  Contoh berikut menunjukkan nilai yang dihasilkan yang menunjukkan klaster yang dikloning yang sebenarnya dan potensial.

```
{
  "DBClusters": [
      {
          "EarliestRestorableTime": "2023-02-01T21:17:54.106Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": false,
...
      },
      {
          "EarliestRestorableTime": "2023-02-09T16:01:07.398Z",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0",
          "CrossAccountClone": true,
...
      },
      {
          "StorageEncrypted": false,
          "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-
abcdefgh",
          "Engine": "aurora-mysql",
          "EngineVersion": "8.0.mysql_aurora.3.02.0"
      }
```

```
    ]
}
```

Mengintegrasikan Aurora dengan layanan AWS lainnya

Integrasikan Amazon Aurora dengan layanan AWS lain sehingga Anda dapat memperluas klaster DB Aurora Anda untuk menggunakan kemampuan tambahan di AWS Cloud.

Topik

- Mengintegrasikan layanan AWS dengan Amazon Aurora MySQL
- Mengintegrasikan layanan AWS dengan Amazon Aurora PostgreSQL
- Menggunakan Amazon Aurora Auto Scaling dengan Replika Aurora

Mengintegrasikan layanan AWS dengan Amazon Aurora MySQL

Amazon Aurora MySQL berintegrasi dengan layanan AWS lain sehingga Anda dapat memperluas klaster DB Aurora MySQL Anda untuk menggunakan kemampuan tambahan di AWS Cloud. Klaster DB Aurora MySQL Anda dapat menggunakan layanan AWS untuk melakukan hal berikut:

- Secara sinkron atau asinkron menginvokasi fungsi AWS Lambda menggunakan fungsi native `lambda_sync` atau `lambda_async`. Atau, secara asinkron menginvokasi fungsi AWS Lambda menggunakan prosedur `mysql.lambda_async`.
- Memuat data dari file teks atau XML yang disimpan dalam bucket Amazon S3 ke dalam klaster DB Anda menggunakan perintah `LOAD DATA FROM S3` atau `LOAD XML FROM S3`.
- Menyimpan data ke file teks yang disimpan dalam bucket Amazon S3 dari klaster DB Anda menggunakan perintah `SELECT INTO OUTFILE S3`.
- Secara otomatis menambahkan atau menghapus Replika Aurora dengan Application Auto Scaling. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Auto Scaling dengan Replika Aurora.

Untuk informasi selengkapnya tentang mengintegrasikan Aurora MySQL dengan layanan AWS lainnya, lihat Mengintegrasikan Amazon Aurora MySQL dengan layanan AWS lainnya.

Mengintegrasikan layanan AWS dengan Amazon Aurora PostgreSQL

Amazon Aurora PostgreSQL berintegrasi dengan layanan AWS lain sehingga Anda dapat memperluas klaster DB Aurora PostgreSQL Anda untuk menggunakan kemampuan tambahan di AWS Cloud. Klaster DB Aurora PostgreSQL Anda dapat menggunakan layanan AWS untuk melakukan hal berikut:

- Mengumpulkan, melihat, dan menilai performa dengan cepat pada beban kerja basis data relasional Anda menggunakan Wawasan Performa.
- Secara otomatis menambahkan atau menghapus Replika Aurora dengan Aurora Auto Scaling. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Auto Scaling dengan Replika Aurora.

Untuk informasi selengkapnya tentang mengintegrasikan Aurora PostgreSQL dengan layanan AWS lainnya, lihat Mengintegrasikan Amazon Aurora PostgreSQL dengan layanan AWS lainnya.

Menggunakan Amazon Aurora Auto Scaling dengan Replika Aurora

Untuk memenuhi persyaratan konektivitas dan beban kerja Anda, Aurora Auto Scaling secara dinamis menyesuaikan jumlah Replika Aurora (instans DB pembaca) terprovisi untuk klaster DB Aurora. Aurora Auto Scaling tersedia untuk Aurora MySQL dan Aurora PostgreSQL. Aurora Auto Scaling memungkinkan klaster DB Aurora Anda menangani peningkatan mendadak dalam konektivitas atau beban kerja. Ketika konektivitas atau beban kerja berkurang, Aurora Auto Scaling menghapus Replika Aurora yang tidak perlu sehingga Anda tidak membayar untuk instans DB terprovisi yang tidak digunakan.

Anda menentukan dan menerapkan kebijakan penskalaan untuk klaster DB Aurora. Kebijakan penskalaan menentukan jumlah minimum dan maksimum Replika Aurora yang dapat dikelola oleh Aurora Auto Scaling. Berdasarkan kebijakan tersebut, Auto Scaling Aurora menyesuaikan jumlah Replika Aurora naik atau turun sebagai respons terhadap beban kerja aktual, yang ditentukan dengan menggunakan metrik Amazon dan nilai target. CloudWatch

Anda dapat menggunakan AWS Management Console untuk menerapkan kebijakan penskalaan berdasarkan metrik standar. Alternatifnya, Anda dapat menggunakan AWS CLI atau API Aurora Auto Scaling untuk menerapkan kebijakan penskalaan berdasarkan metrik standar atau kustom.

Topik

- Sebelum Anda memulai
- Kebijakan Aurora Auto Scaling
- Menambahkan kebijakan penskalaan untuk klaster DB Aurora
- Mengedit kebijakan penskalaan
- Menghapus kebijakan penskalaan
- ID dan pemberian tag instans DB

- Aurora Auto Scaling dan Wawasan Performa

Sebelum Anda memulai

Sebelum Anda dapat menggunakan Aurora Auto Scaling dengan klaster DB Aurora, Anda terlebih dahulu harus membuat sebuah klaster DB Aurora dengan instans DB utama (penulis). Untuk informasi selengkapnya tentang cara membuat klaster DB Aurora, lihat Membuat klaster DB Amazon Aurora.

Aurora Auto Scaling hanya menskalakan klaster DB jika klaster DB berada dalam status tersedia.

Saat Aurora Auto Scaling menambahkan Replika Aurora baru, Replika Aurora yang baru adalah kelas instans DB yang sama dengan yang digunakan oleh instans primer. Untuk informasi selengkapnya tentang kelas instans DB, lihat Kelas instans DB Aurora. Selain itu, tingkat promosi untuk Replika Aurora yang baru akan ditetapkan sesuai prioritas terakhir, yaitu 15 secara default. Ini berarti bahwa selama failover, replika dengan prioritas lebih baik, seperti yang dibuat secara manual, akan dipromosikan terlebih dahulu. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora.

Aurora Auto Scaling hanya menghapus Replika Aurora yang dibuatnya.

Untuk memanfaatkan Aurora Auto Scaling, aplikasi Anda harus mendukung koneksi ke Replika Aurora baru. Untuk melakukannya, sebaiknya gunakan titik akhir pembaca Aurora. Untuk Aurora MySQL, Anda dapat menggunakan driver seperti AWS JDBC Driver for MySQL. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster DB Amazon Aurora.

> **Note**
>
> Basis data global Aurora saat ini tidak mendukung Aurora Auto Scaling untuk klaster DB sekunder.

Kebijakan Aurora Auto Scaling

Aurora Auto Scaling menggunakan kebijakan penskalaan untuk menyesuaikan jumlah Replika Aurora dalam klaster DB Aurora. Aurora Auto Scaling memiliki komponen berikut:

- Peran terkait layanan

- Metrik target
- Kapasitas minimum dan maksimum
- Periode pendinginan

Topik

- Peran terkait layanan
- Metrik target
- Kapasitas minimum dan maksimum
- Periode pendinginan
- Mengaktifkan atau menonaktifkan aktivitas penskalaan ke dalam

Peran terkait layanan

Aurora Auto Scaling menggunakan peran yang berkaitan dengan layanan `AWSServiceRoleForApplicationAutoScaling_RDSCluster`. Untuk informasi selengkapnya, lihat Peran yang ditautkan dengan layanan untuk Application Auto Scaling dalam Panduan Pengguna Application Auto Scaling.

Metrik target

Dalam jenis kebijakan ini, metrik standar atau kustom dan nilai target untuk metrik tersebut ditentukan dalam konfigurasi kebijakan penskalaan pelacakan target. Auto Scaling Aurora membuat dan CloudWatch mengelola alarm yang memicu kebijakan penskalaan dan menghitung penyesuaian penskalaan berdasarkan metrik dan nilai target. Kebijakan penskalaan menambah atau menghapus Replika Aurora yang diperlukan untuk menjaga metrik pada, atau mendekati, nilai target yang ditentukan. Selain menjaga metrik tetap dekat dengan nilai target, kebijakan penskalaan pelacakan target juga disesuaikan menurut fluktuasi metrik karena pola beban kerja yang berubah. Kebijakan tersebut juga meminimalkan fluktuasi cepat dalam jumlah Replika Aurora yang tersedia untuk klaster DB Anda.

Misalnya, ambil kebijakan penskalaan yang menggunakan metrik penggunaan CPU rata-rata standar. Kebijakan tersebut dapat menjaga pemanfaatan CPU pada, atau mendekati, persentase penggunaan tertentu, seperti 40 persen.

> **Note**
>
> Untuk setiap klaster DB Aurora, Anda hanya dapat membuat satu kebijakan Penskalaan Otomatis untuk setiap metrik target.

Kapasitas minimum dan maksimum

Anda dapat menentukan jumlah maksimum Replika Aurora yang akan dikelola oleh Application Auto Scaling. Nilai ini harus diatur ke 0–15, dan harus sama dengan atau lebih besar dari nilai yang ditentukan untuk jumlah minimum Replika Aurora.

Anda dapat menentukan jumlah minimum Replika Aurora yang akan dikelola oleh Application Auto Scaling. Nilai ini harus diatur ke 0–15, dan harus sama dengan atau kurang dari nilai yang ditentukan untuk jumlah maksimum Replika Aurora.

> **Note**
>
> Kapasitas minimum dan maksimum ditetapkan untuk klaster DB Aurora. Nilai yang ditentukan berlaku untuk semua kebijakan yang terkait dengan klaster DB Aurora.

Periode pendinginan

Anda dapat menyesuaikan daya respons kebijakan penskalaan pelacakan target dengan menambahkan periode pendinginan yang memengaruhi penskalaan klaster DB Aurora Anda ke dalam dan ke luar. Periode pendinginan memblokir permintaan penskalaan ke dalam atau ke luar berikutnya hingga periode ini berakhir. Pemblokiran ini memperlambat penghapusan Replika Aurora dalam klaster DB Aurora Anda untuk permintaan penskalaan ke dalam, dan pembuatan Replika Aurora untuk permintaan penskalaan ke luar.

Anda dapat menentukan periode pendinginan berikut:

- Aktivitas penskalaan ke dalam mengurangi jumlah Replika Aurora dalam klaster DB Aurora Anda. Periode pendinginan penskalaan ke dalam menentukan jumlah waktu, dalam detik, setelah aktivitas penskalaan ke dalam selesai sebelum aktivitas penskalaan ke dalam lainnya dapat dimulai.

- Aktivitas penskalaan ke luar menambah jumlah Replika Aurora dalam klaster DB Aurora Anda. Periode pendinginan penskalaan ke luar menentukan jumlah waktu, dalam detik, setelah aktivitas penskalaan ke luar selesai sebelum aktivitas penskalaan ke luar lainnya dapat dimulai.

> **Note**
>
> Periode pendinginan penskalaan ke luar diabaikan jika permintaan penskalaan ke luar berikutnya ditujukan untuk jumlah Replika Aurora yang lebih besar daripada permintaan pertama.

Jika Anda tidak mengatur periode pendinginan penskalaan masuk atau ke luar, nilai default untuk masing-masing adalah 300 detik.

Mengaktifkan atau menonaktifkan aktivitas penskalaan ke dalam

Anda dapat mengaktifkan atau menonaktifkan aktivitas penskalaan ke dalam untuk sebuah kebijakan. Mengaktifkan aktivitas penskalaan ke dalam memungkinkan kebijakan penskalaan untuk menghapus Replika Aurora. Saat aktivitas penskalaan ke dalam diaktifkan, periode pendinginan penskalaan ke dalam di kebijakan penskalaan berlaku untuk aktivitas penskalaan ke dalam. Menonaktifkan aktivitas penskalaan ke dalam akan mencegah kebijakan penskalaan menghapus Replika Aurora.

> **Note**
>
> Aktivitas penskalaan ke luar selalu diaktifkan sehingga kebijakan penskalaan dapat membuat Replika Aurora sesuai kebutuhan.

Menambahkan kebijakan penskalaan untuk klaster DB Aurora

Anda dapat menambahkan kebijakan penskalaan menggunakan AWS Management Console, AWS CLI, atau API Application Auto Scaling.

> **Note**
>
> Untuk contoh yang menambahkan kebijakan penskalaan menggunakan AWS CloudFormation, lihat Menetapkan kebijakan penskalaan untuk klaster DB Aurora dalam Panduan Pengguna AWS CloudFormation.

Konsol

Anda dapat menambahkan kebijakan penskalaan ke klaster DB Aurora menggunakan AWS Management Console.

Untuk menambahkan kebijakan penskalaan otomatis ke klaster DB Aurora

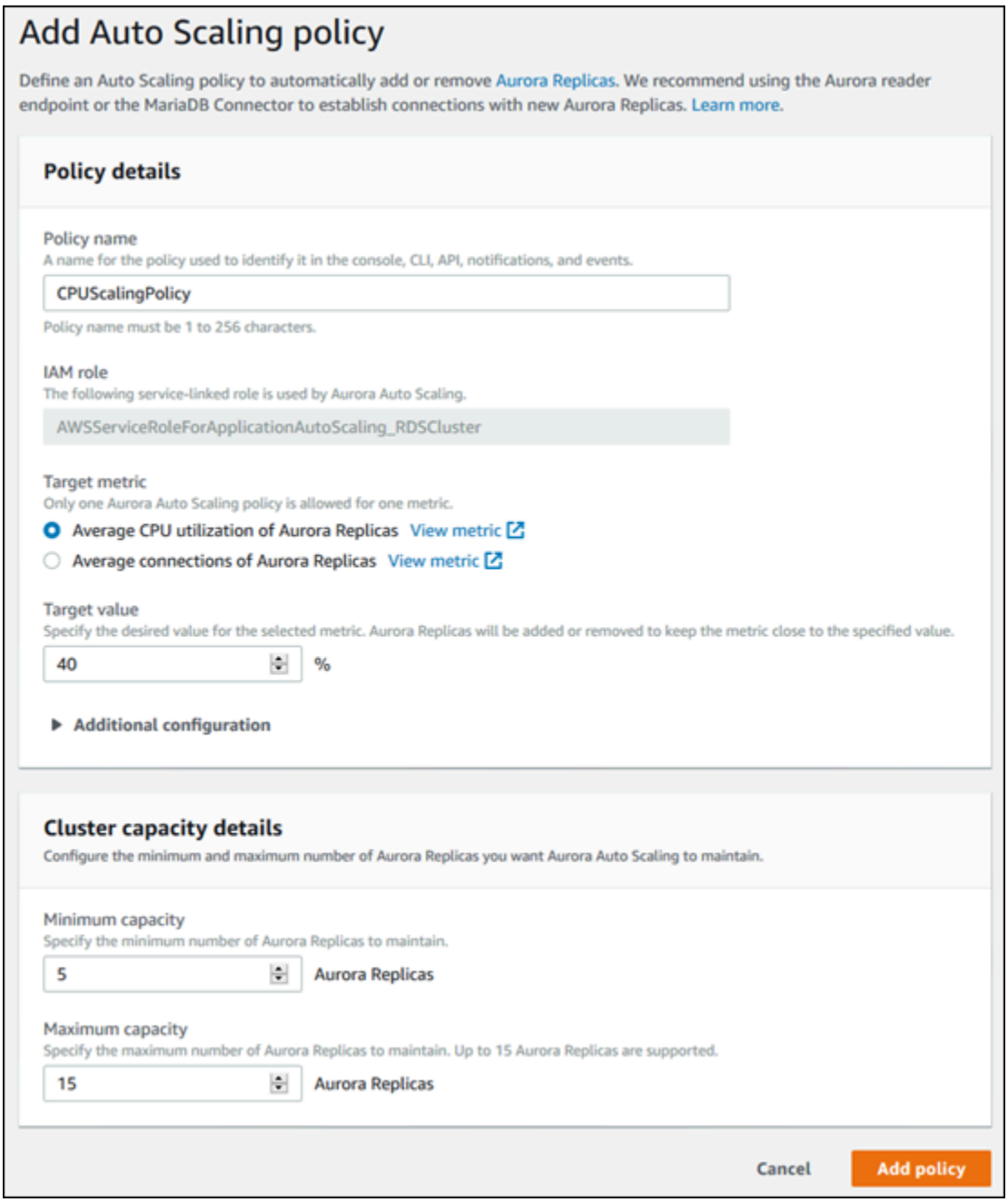
1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB Aurora yang ingin Anda tambahi kebijakan.
4. Pilih tab Log & peristiwa.
5. Pada bagian Kebijakan auto scaling, pilih Tambahkan.

   Kotak dialog Tambahkan kebijakan Auto Scaling muncul.
6. Untuk Nama Kebijakan, ketik nama kebijakan.
7. Untuk metrik target, pilih salah satu dari berikut:
   - Pemanfaatan CPU rata-rata dari Aurora Replicas untuk membuat kebijakan berdasarkan pemanfaatan CPU rata-rata.
   - Koneksi rata-rata dari Aurora Replicas untuk membuat kebijakan berdasarkan jumlah koneksi rata-rata ke Replika Aurora.
8. Untuk nilai target, ketik salah satu dari berikut ini:
   - Jika Anda memilih Pemanfaatan CPU rata-rata dari Aurora Replicas pada langkah sebelumnya, ketik persentase pemanfaatan CPU yang ingin Anda pertahankan pada Replika Aurora.
   - Jika Anda memilih Koneksi rata-rata dari Aurora Replicas pada langkah sebelumnya, ketik jumlah koneksi yang ingin Anda pertahankan.

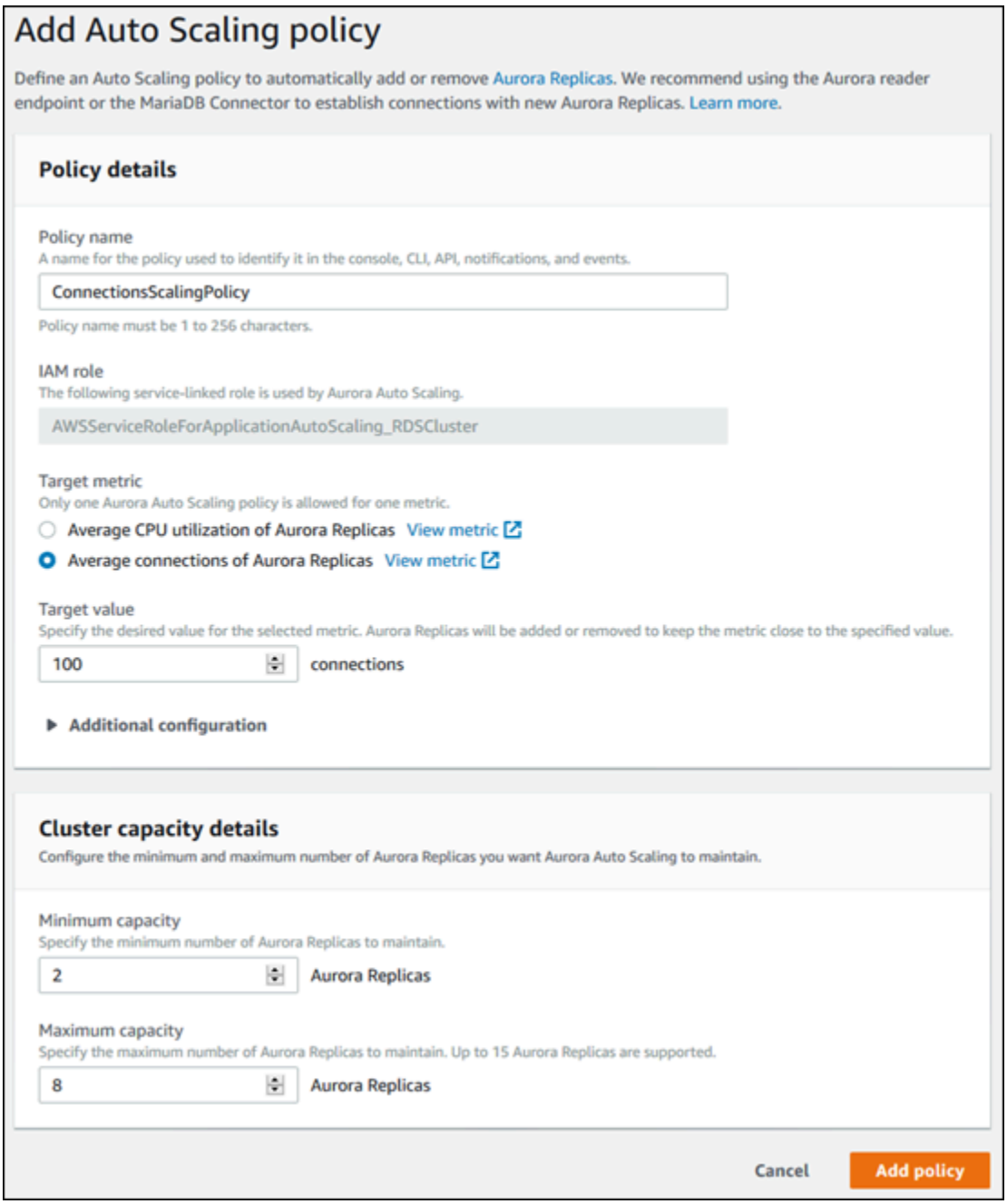
Replika Aurora ditambahkan atau dihapus untuk menjaga metrik tetap mendekati nilai yang ditentukan.

9. (Opsional) Perluas Konfigurasi Tambahan untuk membuat periode pendinginan penskalaan ke dalam atau ke luar.
10. Untuk Kapasitas minimum, ketik jumlah minimum Replika Aurora yang diwajibkan oleh kebijakan Aurora Auto Scaling untuk dipertahankan.
11. Untuk Kapasitas maksimum, ketik jumlah maksimum Replika Aurora yang diwajibkan oleh kebijakan Aurora Auto Scaling untuk dipertahankan.
12. Pilih Tambahkan kebijakan.

Kotak dialog berikut membuat kebijakan Penskalaan Otomatis berdasarkan pemanfaatan CPU rata-rata sebesar 40 persen. Kebijakan tersebut menentukan minimum 5 Replika Aurora dan maksimum 15 Replika Aurora.



Kotak dialog berikut membuat kebijakan penskalaan otomatis berdasarkan penggunaan koneksi rata-rata sebesar 100. Kebijakan tersebut menentukan minimum dua Replika Aurora dan maksimum delapan Replika Aurora.



AWS CLI atau API Application Auto Scaling

Anda dapat menerapkan kebijakan penskalaan berdasarkan metrik standar atau kustom. Untuk melakukannya, Anda dapat menggunakan AWS CLI atau API Application Auto Scaling. Langkah pertama adalah untuk mendaftarkan klaster DB Aurora Anda dengan Application Auto Scaling.

Mendaftarkan klaster DB Aurora

Sebelum Anda dapat menggunakan Aurora Auto Scaling dengan klaster DB Aurora, Anda perlu mendaftarkan klaster DB Aurora dengan Application Auto Scaling. Anda melakukannya untuk mendefinisikan dimensi dan batasan penskalaan yang akan diterapkan pada klaster tersebut. Application Auto Scaling menskalakan klaster DB Aurora secara dinamis bersama dengan dimensi `rds:cluster:ReadReplicaCount` yang dapat diskalakan, yang merepresentasikan jumlah Replika Aurora.

Untuk mendaftarkan klaster DB Aurora, Anda dapat menggunakan AWS CLI atau API Application Auto Scaling.

AWS CLI

Untuk mendaftarkan klaster DB Aurora Anda, gunakan perintah `register-scalable-target` AWS CLI dengan parameter berikut:

- `--service-namespace` – Atur nilai ini ke `rds`.
- `--resource-id` – Pengidentifikasi sumber daya untuk klaster DB Aurora. Untuk parameter ini, jenis sumber dayanya adalah `cluster` dan pengidentifikasi uniknya adalah nama klaster DB Aurora, misalnya `cluster:myscalablecluster`.
- `--scalable-dimension` – Atur nilai ini ke `rds:cluster:ReadReplicaCount`.
- `--min-capacity` – Jumlah minimum instans DB pembaca yang akan dikelola oleh Application Auto Scaling. Untuk informasi tentang hubungan antara `--min-capacity`, `--max-capacity`, dan jumlah instans DB dalam klaster Anda, lihat Kapasitas minimum dan maksimum.
- `--max-capacity` – Jumlah maksimum instans DB pembaca yang akan dikelola oleh Application Auto Scaling. Untuk informasi tentang hubungan antara `--min-capacity`, `--max-capacity`, dan jumlah instans DB dalam klaster Anda, lihat Kapasitas minimum dan maksimum.

Example

Dalam contoh berikut, Anda mendaftarkan klaster DB Aurora bernama `myscalablecluster`. Pendaftaran ini menunjukkan bahwa klaster DB harus diskalakan secara dinamis untuk memiliki satu hingga delapan Replika Aurora.

Untuk Linux, macOS, atau Unix:

```
aws application-autoscaling register-scalable-target \
```

```
    --service-namespace rds \
    --resource-id cluster:myscalablecluster \
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --min-capacity 1 \
    --max-capacity 8 \
```

Untuk Windows:

```
aws application-autoscaling register-scalable-target ^
    --service-namespace rds ^
    --resource-id cluster:myscalablecluster ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --min-capacity 1 ^
    --max-capacity 8 ^
```

API Application Auto Scaling

Untuk mendaftarkan klaster DB Aurora Anda dengan Application Auto Scaling, gunakan operasi API Application Auto Scaling `RegisterScalableTarget` dengan parameter berikut:

- `ServiceNamespace` – Atur nilai ini ke `rds`.
- `ResourceID` – Pengidentifikasi sumber daya untuk klaster DB Aurora. Untuk parameter ini, jenis sumber dayanya adalah `cluster` dan pengidentifikasi uniknya adalah nama klaster DB Aurora, misalnya `cluster:myscalablecluster`.
- `ScalableDimension` – Atur nilai ini ke `rds:cluster:ReadReplicaCount`.
- `MinCapacity` – Jumlah minimum instans DB pembaca yang akan dikelola oleh Application Auto Scaling. Untuk informasi tentang hubungan antara `MinCapacity`, `MaxCapacity`, dan jumlah instans DB dalam klaster Anda, lihat Kapasitas minimum dan maksimum.
- `MaxCapacity` – Jumlah maksimum instans DB pembaca yang akan dikelola oleh Application Auto Scaling. Untuk informasi tentang hubungan antara `MinCapacity`, `MaxCapacity`, dan jumlah instans DB dalam klaster Anda, lihat Kapasitas minimum dan maksimum.

Example

Dalam contoh berikut, Anda mendaftarkan klaster DB Aurora bernama `myscalablecluster` dengan API Application Auto Scaling. Pendaftaran ini menunjukkan bahwa klaster DB harus diskalakan secara dinamis untuk memiliki satu hingga delapan Replika Aurora.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "MinCapacity": 1,
    "MaxCapacity": 8
}
```

Menetapkan kebijakan penskalaan untuk klaster DB Aurora

Konfigurasi kebijakan penskalaan pelacakan target direpresentasikan oleh blok JSON yang digunakan untuk mendefinisikan metrik dan nilai target. Anda dapat menyimpan konfigurasi kebijakan penskalaan sebagai blok JSON dalam file teks. Anda menggunakan file teks tersebut saat menginvokasi AWS CLI atau API Application Auto Scaling. Untuk informasi selengkapnya tentang sintaksis konfigurasi kebijakan, lihat `TargetTrackingScalingPolicyConfiguration` dalam Referensi API Application Auto Scaling.

Opsi berikut tersedia untuk menetapkan konfigurasi kebijakan penskalaan pelacakan target.

Topik

- Menggunakan metrik standar
- Menggunakan metrik kustom
- Menggunakan periode pendinginan
- Menonaktifkan aktivitas penskalaan ke dalam

Menggunakan metrik standar

Dengan menggunakan metrik standar, Anda dapat dengan cepat menentukan kebijakan penskalaan pelacakan target untuk klaster DB Aurora yang berfungsi baik, dengan pelacakan target dan penskalaan dinamis dalam Aurora Auto Scaling.

Saat ini, Aurora mendukung metrik standar berikut dalam Aurora Auto Scaling:

- RDS ReaderAverage CPUUutilization — Nilai rata-rata `CPUUtilization` metrik di CloudWatch semua Replika Aurora di cluster Aurora DB.
- RDS ReaderAverageDatabaseConnections — Nilai rata-rata `DatabaseConnections` metrik di semua Replika Aurora CloudWatch di cluster Aurora DB.

Untuk informasi selengkapnya tentang metrik `CPUUtilization` dan `DatabaseConnections`, lihat CloudWatch Metrik Amazon untuk Amazon Aurora.

Untuk menggunakan metrik standar dalam kebijakan penskalaan, Anda membuat konfigurasi pelacakan target untuk kebijakan penskalaan Anda. Konfigurasi ini harus menyertakan `PredefinedMetricSpecification` untuk metrik standar dan `TargetValue` untuk nilai target metrik tersebut.

Example

Contoh berikut menjelaskan konfigurasi kebijakan umum untuk penskalaan pelacakan target untuk klaster DB Aurora. Dalam konfigurasi ini, metrik `RDSReaderAverageCPUUtilization` standar digunakan untuk menyesuaikan klaster DB Aurora berdasarkan pemanfaatan CPU rata-rata sebesar 40 persen di seluruh Replika Aurora.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    }
}
```

Menggunakan metrik kustom

Dengan menggunakan metrik kustom, Anda dapat menentukan kebijakan penskalaan pelacakan target yang memenuhi persyaratan kustom Anda. Anda dapat menentukan metrik kustom berdasarkan metrik Aurora apa pun yang berubah secara berbanding lurus dengan penskalaan.

Tidak semua metrik Aurora berfungsi untuk pelacakan target. Metrik harus berupa metrik pemanfaatan yang valid dan menjelaskan tingkat kesibukan sebuah instans. Nilai metrik harus meningkat atau menurun secara berbanding lurus dengan jumlah Replika Aurora dalam klaster DB Aurora. Peningkatan atau penurunan proporsional ini diperlukan untuk menggunakan data metrik untuk menskalakan ke luar atau ke dalam jumlah Replika Aurora secara proporsional.

Example

Contoh berikut menjelaskan konfigurasi pelacakan target untuk kebijakan penskalaan. Dalam konfigurasi ini, metrik kustom menyesuaikan klaster DB Aurora berdasarkan penggunaan CPU rata-rata sebesar 50 persen di seluruh Replika Aurora dalam klaster DB Aurora bernama `my-db-cluster`.

```
{
    "TargetValue": 50,
    "CustomizedMetricSpecification":
    {
        "MetricName": "CPUUtilization",
        "Namespace": "AWS/RDS",
        "Dimensions": [
            {"Name": "DBClusterIdentifier","Value": "my-db-cluster"},
            {"Name": "Role","Value": "READER"}
        ],
        "Statistic": "Average",
        "Unit": "Percent"
    }
}
```

Menggunakan periode pendinginan

Anda dapat menentukan nilai, dalam detik, untuk `ScaleOutCooldown` guna menambahkan periode pendinginan untuk menskalakan ke luar klaster DB Aurora Anda. Demikian pula, Anda dapat menentukan nilai, dalam detik, untuk `ScaleInCooldown` guna menambahkan periode pendinginan untuk menskalakan ke dalam klaster DB Aurora Anda. Untuk

informasi selengkapnya tentang `ScaleInCooldown` dan `ScaleOutCooldown`, lihat `TargetTrackingScalingPolicyConfiguration` dalam Referensi API Application Auto Scaling.

Example

Contoh berikut menjelaskan konfigurasi pelacakan target untuk kebijakan penskalaan. Dalam konfigurasi ini, metrik standar `RDSReaderAverageCPUUtilization` digunakan untuk menyesuaikan klaster DB Aurora berdasarkan pemanfaatan CPU rata-rata sebesar 40 persen di seluruh Replika Aurora dalam klaster DB Aurora tersebut. Konfigurasi ini menyediakan periode pendinginan penskalaan ke dalam selama 10 menit dan periode pendinginan penskalaan ke luar selama 5 menit.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "ScaleInCooldown": 600,
    "ScaleOutCooldown": 300
}
```

Menonaktifkan aktivitas penskalaan ke dalam

Anda dapat mencegah konfigurasi kebijakan penskalaan pelacakan target agar tidak menskalakan klaster DB Aurora ke dalam dengan menonaktifkan aktivitas penskalaan ke dalam. Menonaktifkan aktivitas penskalaan ke dalam akan mencegah kebijakan penskalaan menghapus Replika Aurora, tetapi masih memungkinkan kebijakan penskalaan membuat Replika Aurora sesuai kebutuhan.

Anda dapat menentukan nilai Boolean untuk `DisableScaleIn` guna mengaktifkan atau menonaktifkan aktivitas penskalaan ke dalam untuk klaster DB Aurora Anda. Untuk informasi selengkapnya tentang `DisableScaleIn`, lihat `TargetTrackingScalingPolicyConfiguration` dalam Referensi API Application Auto Scaling.

Example

Contoh berikut menjelaskan konfigurasi pelacakan target untuk kebijakan penskalaan. Dalam konfigurasi ini, metrik `RDSReaderAverageCPUUtilization` standar menggunakan klaster DB Aurora berdasarkan pemanfaatan CPU rata-rata sebesar 40 persen di seluruh Replika Aurora dalam klaster DB Aurora tersebut. Konfigurasi ini menonaktifkan aktivitas penskalaan ke dalam untuk kebijakan penskalaan.

```
{
    "TargetValue": 40.0,
    "PredefinedMetricSpecification":
    {
        "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
    },
    "DisableScaleIn": true
}
```

Menerapkan kebijakan penskalaan untuk klaster DB Aurora

Setelah mendaftarkan klaster DB Aurora Anda dengan Application Auto Scaling dan menentukan kebijakan penskalaan, Anda perlu menerapkan kebijakan penskalaan ini ke klaster DB Aurora yang terdaftar. Untuk menerapkan kebijakan penskalaan pada klaster DB Aurora, Anda dapat menggunakan AWS CLI atau API Application Auto Scaling.

AWS CLI

Untuk menerapkan kebijakan penskalaan pada klaster DB Aurora Anda, gunakan perintah `put-scaling-policy` AWS CLI dengan parameter berikut:

- `--policy-name` – Nama kebijakan penskalaan.
- `--policy-type` – Atur nilai ini ke `TargetTrackingScaling`.
- `--resource-id` – Pengidentifikasi sumber daya untuk klaster DB Aurora. Untuk parameter ini, jenis sumber dayanya adalah `cluster` dan pengidentifikasi uniknya adalah nama klaster DB Aurora, misalnya `cluster:myscalablecluster`.
- `--service-namespace` – Atur nilai ini ke `rds`.
- `--scalable-dimension` – Atur nilai ini ke `rds:cluster:ReadReplicaCount`.
- `--target-tracking-scaling-policy-configuration` – Konfigurasi kebijakan penskalaan pelacakan target untuk digunakan pada klaster DB Aurora.

Example

Dalam contoh berikut, Anda menerapkan kebijakan penskalaan pelacakan target yang disebut `myscalablepolicy` ke klaster DB Aurora bernama `myscalablecluster` dengan Application Auto Scaling. Untuk melakukannya, Anda menggunakan konfigurasi kebijakan yang disimpan dalam file bernama `config.json`.

Untuk Linux, macOS, atau Unix:

```
aws application-autoscaling put-scaling-policy \
    --policy-name myscalablepolicy \
    --policy-type TargetTrackingScaling \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
    --target-tracking-scaling-policy-configuration file://config.json
```

Untuk Windows:

```
aws application-autoscaling put-scaling-policy ^
    --policy-name myscalablepolicy ^
    --policy-type TargetTrackingScaling ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
    --target-tracking-scaling-policy-configuration file://config.json
```

API Application Auto Scaling

Untuk menerapkan kebijakan penskalaan pada klaster DB Aurora dengan API Application Auto Scaling, gunakan operasi `PutScalingPolicy` API Application Auto Scaling dengan parameter berikut:

- `PolicyName` – Nama kebijakan penskalaan.
- `ServiceNamespace` – Atur nilai ini ke `rds`.
- `ResourceID` – Pengidentifikasi sumber daya untuk klaster DB Aurora. Untuk parameter ini, jenis sumber dayanya adalah `cluster` dan pengidentifikasi uniknya adalah nama klaster DB Aurora, misalnya `cluster:myscalablecluster`.
- `ScalableDimension` – Atur nilai ini ke `rds:cluster:ReadReplicaCount`.
- `PolicyType` – Atur nilai ini ke `TargetTrackingScaling`.
- `TargetTrackingScalingPolicyConfiguration` – Konfigurasi kebijakan penskalaan pelacakan target untuk digunakan pada klaster DB Aurora.

Example

Dalam contoh berikut, Anda menerapkan kebijakan penskalaan pelacakan target yang disebut `myscalablepolicy` ke klaster DB Aurora bernama `myscalablecluster` dengan Application Auto Scaling. Anda menggunakan konfigurasi kebijakan berdasarkan pada metrik standar `RDSReaderAverageCPUUtilization`.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.PutScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 40.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "RDSReaderAverageCPUUtilization"
        }
    }
}
```

Mengedit kebijakan penskalaan

Anda dapat mengedit kebijakan penskalaan menggunakan AWS Management Console, AWS CLI, atau API Application Auto Scaling.

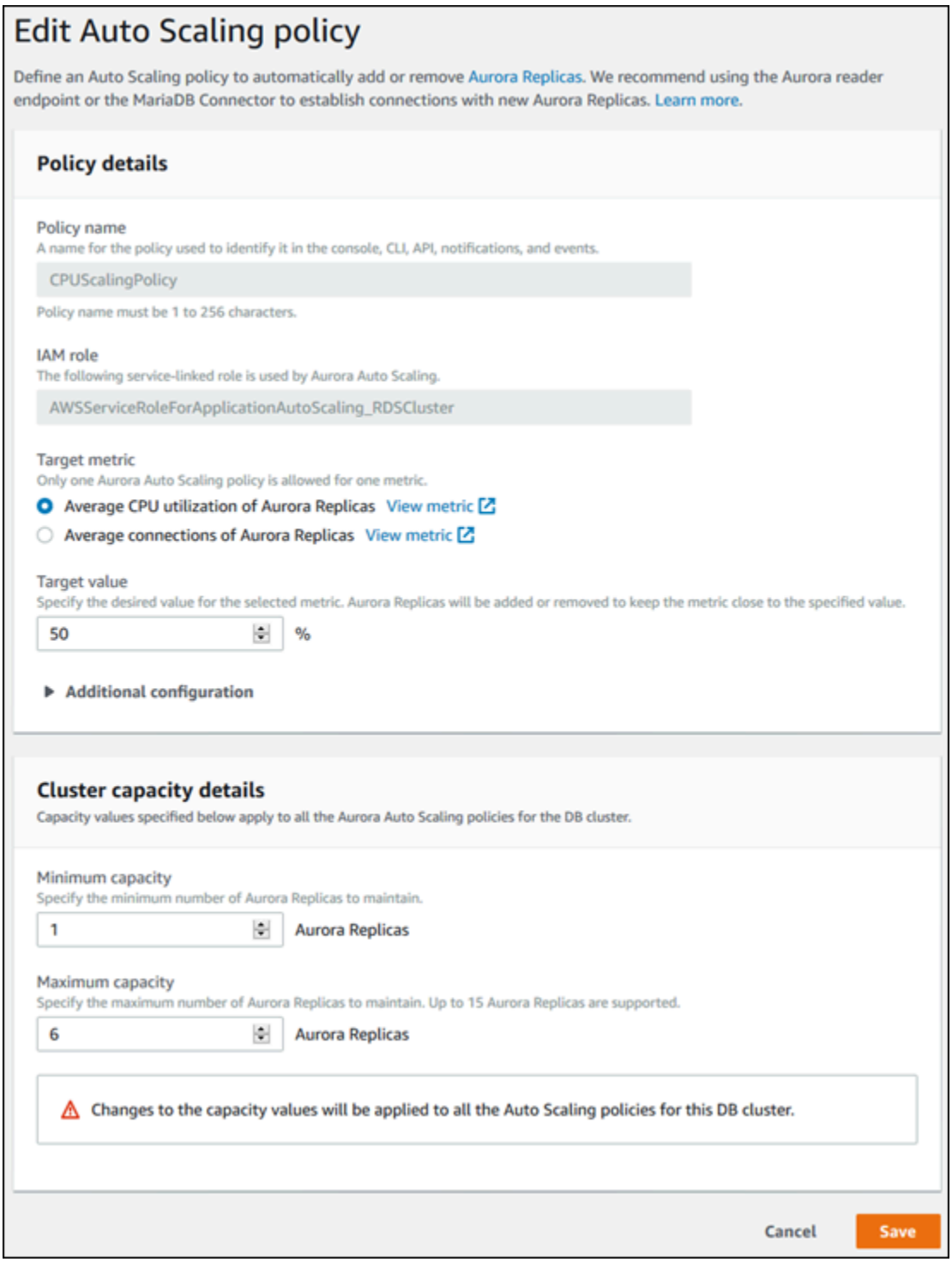
Konsol

Anda dapat mengedit kebijakan penskalaan menggunakan AWS Management Console.

Untuk mengedit kebijakan penskalaan untuk klaster DB Aurora

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB Aurora yang kebijakan penskalaan otomatisnya ingin Anda edit.
4. Pilih tab Log & peristiwa.
5. Pada bagian Kebijakan auto scaling, pilih kebijakan penskalaan otomatis, lalu pilih Edit.
6. Buat perubahan pada kebijakan.
7. Pilih Simpan.

Berikut ini adalah contoh kotak dialog Edit kebijakan Auto Scaling.



AWS CLI atau API Application Auto Scaling

Anda dapat menggunakan AWS CLI atau API Application Auto Scaling untuk mengedit kebijakan penskalaan dengan cara yang sama seperti Anda menerapkan kebijakan penskalaan:

- Saat menggunakan AWS CLI, tentukan nama kebijakan yang ingin Anda edit dalam parameter `--policy-name`. Tentukan nilai baru untuk parameter yang ingin Anda ubah.
- Saat menggunakan API Application Auto Scaling, tentukan nama kebijakan yang ingin Anda edit dalam parameter `PolicyName`. Tentukan nilai baru untuk parameter yang ingin Anda ubah.

Untuk informasi selengkapnya, lihat Menerapkan kebijakan penskalaan untuk klaster DB Aurora.

Menghapus kebijakan penskalaan

Anda dapat menghapus kebijakan penskalaan menggunakan AWS Management Console, AWS CLI, atau API Application Auto Scaling.

Konsol

Anda dapat menghapus kebijakan penskalaan menggunakan AWS Management Console.

Untuk menghapus kebijakan penskalaan untuk klaster DB Aurora

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB Aurora yang kebijakan penskalaan otomatisnya ingin Anda hapus.
4. Pilih tab Log & peristiwa.
5. Pada bagian Kebijakan auto scaling, pilih kebijakan penskalaan otomatis, lalu pilih Hapus.

AWS CLI

Untuk menghapus kebijakan penskalaan dari klaster DB Aurora Anda, gunakan perintah `delete-scaling-policy` AWS CLI dengan parameter berikut:

- `--policy-name` – Nama kebijakan penskalaan.
- `--resource-id` – Pengidentifikasi sumber daya untuk klaster DB Aurora. Untuk parameter ini, jenis sumber dayanya adalah `cluster` dan pengidentifikasi uniknya adalah nama klaster DB Aurora, misalnya `cluster:myscalablecluster`.
- `--service-namespace` – Atur nilai ini ke `rds`.
- `--scalable-dimension` – Atur nilai ini ke `rds:cluster:ReadReplicaCount`.

Example

Dalam contoh berikut, Anda menghapus kebijakan penskalaan pelacakan target bernama `myscalablepolicy` dari klaster DB Aurora bernama `myscalablecluster`.

Untuk Linux, macOS, atau Unix:

```
aws application-autoscaling delete-scaling-policy \
    --policy-name myscalablepolicy \
    --resource-id cluster:myscalablecluster \
    --service-namespace rds \
    --scalable-dimension rds:cluster:ReadReplicaCount \
```

Untuk Windows:

```
aws application-autoscaling delete-scaling-policy ^
    --policy-name myscalablepolicy ^
    --resource-id cluster:myscalablecluster ^
    --service-namespace rds ^
    --scalable-dimension rds:cluster:ReadReplicaCount ^
```

API Application Auto Scaling

Untuk menghapus kebijakan penskalaan dari klaster DB Aurora Anda, gunakan operasi `DeleteScalingPolicy` API Application Auto Scaling dengan parameter berikut:

- `PolicyName` – Nama kebijakan penskalaan.
- `ServiceNamespace` – Atur nilai ini ke `rds`.
- `ResourceID` – Pengidentifikasi sumber daya untuk klaster DB Aurora. Untuk parameter ini, jenis sumber dayanya adalah `cluster` dan pengidentifikasi uniknya adalah nama klaster DB Aurora, misalnya `cluster:myscalablecluster`.
- `ScalableDimension` – Atur nilai ini ke `rds:cluster:ReadReplicaCount`.

Example

Dalam contoh berikut, Anda menghapus kebijakan penskalaan pelacakan target bernama `myscalablepolicy` dari klaster DB Aurora bernama `myscalablecluster` dengan API Application Auto Scaling.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
Content-Length: 219
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "rds",
    "ResourceId": "cluster:myscalablecluster",
    "ScalableDimension": "rds:cluster:ReadReplicaCount"
}
```

ID dan pemberian tag instans DB

Ketika replika ditambahkan oleh Aurora Auto Scaling, ID instans DB-nya diberi awalan `application-autoscaling-`, misalnya, `application-autoscaling-61aabbcc-4e2f-4c65-b620-ab7421abc123`.

Tag berikut secara otomatis ditambahkan ke instans DB. Anda dapat melihatnya di tab Tanda pada halaman detail instans DB.

Tag	Nilai
application-autoscaling:resourceId	cluster:mynewcluster-cluster

Untuk informasi selengkapnya tentang tag sumber daya Amazon RDS, lihat Memberi tag pada sumber daya Amazon RDS.

Aurora Auto Scaling dan Wawasan Performa

Anda dapat menggunakan Wawasan Performa untuk memantau replika yang telah ditambahkan oleh Aurora Auto Scaling, sama seperti instans DB pembaca Aurora lainnya.

Anda tidak dapat mengaktifkan Wawasan Performa untuk klaster DB Aurora. Anda dapat mengaktifkan Wawasan Performa secara manual untuk setiap instans DB dalam klaster DB.

Saat Anda mengaktifkan Wawasan Performa untuk instans DB penulis di klaster DB Aurora Anda, Wawasan Performa tidak diaktifkan secara otomatis untuk instans DB pembaca. Anda harus mengaktifkan Wawasan Performa secara manual untuk instans DB pembaca yang ada dan replika baru yang ditambahkan oleh Aurora Auto Scaling.

Untuk informasi selengkapnya tentang penggunaan Wawasan Performa untuk memantau klaster DB Aurora, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

Memelihara klaster DB Amazon Aurora

Amazon RDS melakukan pemeliharaan secara berkala pada sumber daya Amazon RDS. Pemeliharaan sering kali melibatkan pembaruan ke sumber daya berikut di klaster DB:

- Perangkat keras yang mendasarinya
- Sistem operasi yang mendasarinya (OS)
- Versi mesin basis data

Pembaruan pada sistem operasi paling sering terjadi untuk masalah keamanan. Anda harus melakukannya sesegera mungkin.

Beberapa item pemeliharaan mengharuskan Amazon RDS membuat klaster DB Anda offline selama waktu yang singkat. Item pemeliharaan yang mengharuskan sumber daya untuk offline mencakup patching sistem operasi atau basis data yang diperlukan. Patching yang diperlukan secara otomatis dijadwalkan hanya untuk patch yang terkait dengan keamanan dan keandalan instans. Patching tersebut jarang terjadi, biasanya sekali setiap beberapa bulan. Ini jarang membutuhkan lebih dari periode pemeliharaan Anda.

Modifikasi klaster dan instans DB tertunda yang Anda pilih untuk tidak segera diterapkan juga diterapkan selama periode pemeliharaan. Misalnya, Anda dapat memilih untuk mengubah kelas atau klaster instans DB atau grup parameter DB selama periode pemeliharaan. Modifikasi seperti yang Anda tentukan menggunakan pengaturan boot ulang tertunda tidak muncul dalam daftar Pemeliharaan tertunda. Untuk informasi tentang cara mengubah klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Untuk melihat modifikasi yang tertunda untuk jendela pemeliharaan berikutnya, gunakan describe-db-clusters AWS CLI perintah dan periksa `PendingModifiedValues` bidangnya.

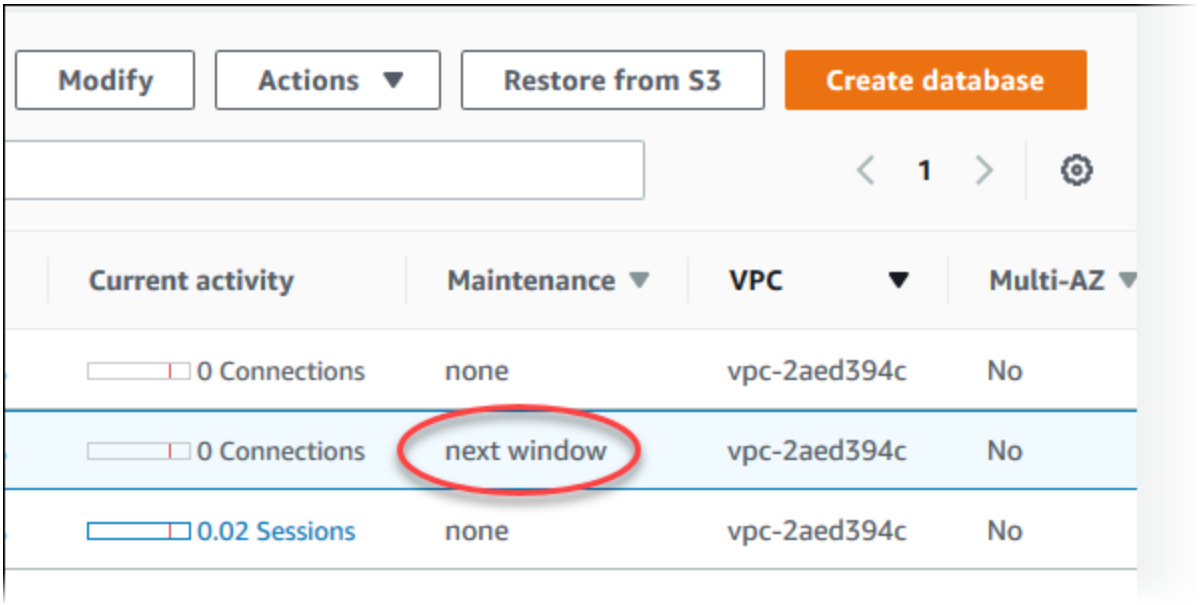
Topik

- Melihat pemeliharaan Tertunda
- Menerapkan pembaruan untuk klaster DB
- Periode pemeliharaan Amazon RDS
- Menyesuaikan periode pemeliharaan klaster DB yang dinginkan
- Peningkatan versi minor otomatis untuk klaster DB Aurora
- Memilih frekuensi pembaruan pemeliharaan Aurora MySQL

- Bekerja dengan pembaruan sistem operasi

Melihat pemeliharaan Tertunda

Lihat apakah pembaruan pemeliharaan tersedia untuk cluster DB Anda dengan menggunakan konsol RDS, API AWS CLI, atau RDS. Jika tersedia, pembaruan akan dicantumkan dalam kolom Pemeliharaan untuk klaster DB di konsol Amazon RDS, seperti yang ditunjukkan berikut.



Jika pembaruan pemeliharaan tidak tersedia untuk klaster DB, nilai kolomnya adalah tidak ada.

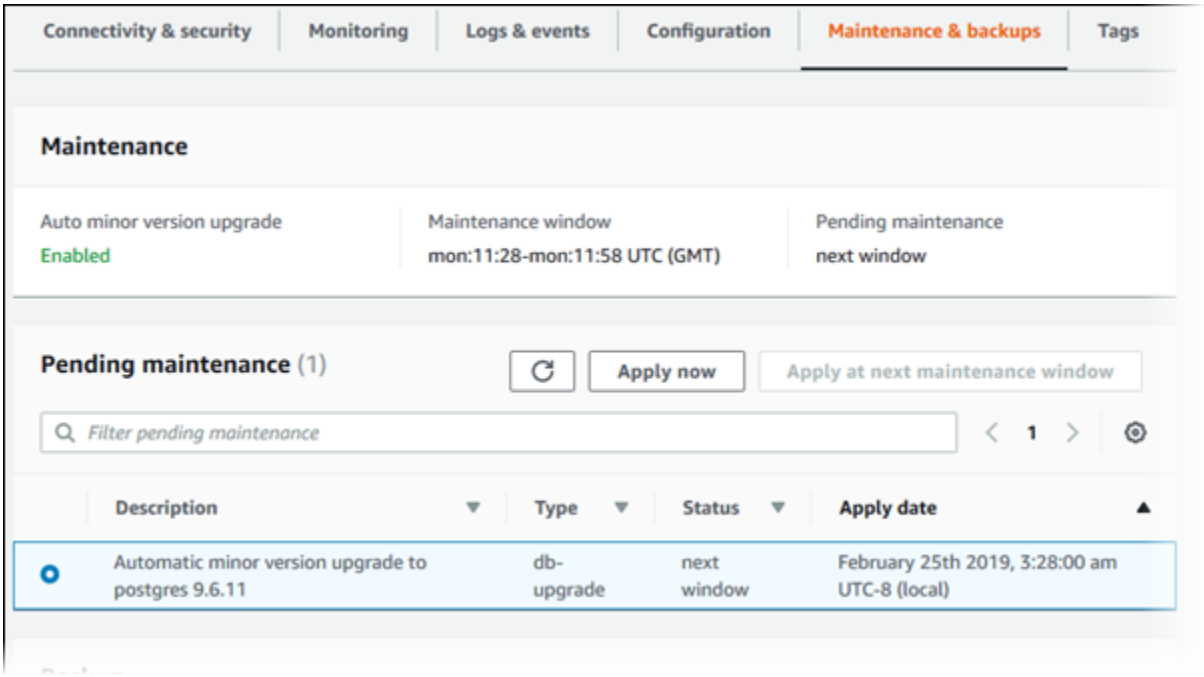
Jika pembaruan pemeliharaan tersedia untuk klaster DB, kemungkinan nilai kolomnya adalah sebagai berikut:

- diperlukan – Tindakan pemeliharaan akan diterapkan ke sumber daya dan tidak dapat ditunda tanpa batas waktu.
- tersedia – Tindakan pemeliharaan tersedia, tetapi tidak akan diterapkan ke sumber daya secara otomatis. Anda dapat menerapkannya secara manual.
- periode berikutnya – Tindakan pemeliharaan akan diterapkan ke sumber daya pada periode pemeliharaan berikutnya.
- Sedang berlangsung – Tindakan pemeliharaan sedang dalam proses penerapan ke sumber daya.

Jika pembaruan tersedia, Anda dapat melakukan salah satu tindakan berikut:

- Jika nilai pemeliharaannya periode berikutnya, tunda item pemeliharaan dengan memilih Tunda peningkatan dari Tindakan. Anda tidak dapat menunda tindakan pemeliharaan jika sudah dimulai.
- Segera terapkan item pemeliharaan.
- Jadwalkan item pemeliharaan untuk dimulai pada periode pemeliharaan berikutnya.
- Tidak melakukan tindakan apa pun.

Untuk melakukan tindakan, pilih klaster DB untuk menampilkan detailnya, kemudian pilih Pemeliharaan & pencadangan. Item pemeliharaan yang tertunda muncul.



Periode pemeliharaan menentukan kapan operasi yang tertunda dimulai, tetapi tidak membatasi total waktu eksekusi operasi ini. Operasi pemeliharaan tidak dijamin selesai sebelum periode pemeliharaan berakhir, dan dapat berlanjut melebihi waktu akhir yang ditentukan. Untuk informasi selengkapnya, lihat Periode pemeliharaan Amazon RDS.

Untuk informasi tentang pembaruan pada mesin Amazon Aurora dan petunjuk untuk meningkatkan dan mem-patching mesin, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL dan Pembaruan Amazon Aurora PostgreSQL.

Anda juga dapat melihat apakah pembaruan pemeliharaan tersedia untuk cluster DB Anda dengan menjalankan `describe-pending-maintenance-actions` AWS CLI perintah.

Menerapkan pembaruan untuk klaster DB

Dengan Amazon RDS, Anda dapat memilih waktu untuk menerapkan operasi pemeliharaan. Anda dapat memutuskan kapan Amazon RDS menerapkan pembaruan dengan menggunakan konsol RDS, AWS Command Line Interface (AWS CLI), atau RDS API.

> **Note**
>
> Untuk RDS for SQL Server, pembaruan ke sistem operasi yang mendasarinya dapat diterapkan dengan menghentikan dan memulai instans DB Anda, atau dengan menaikkan skala kelas instans DB dan kemudian menurunkannya lagi.

Konsol

Untuk mengelola pembaruan untuk instans DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB yang memiliki pembaruan yang diperlukan.
4. Untuk Tindakan, pilih salah satu opsi berikut:
   - Tingkatkan sekarang
   - Tingkatkan pada periode berikutnya

   > **Note**
   >
   > Jika memilih Tingkatkan pada periode berikutnya dan ingin menunda pembaruan di lain waktu, Anda dapat memilih Tunda peningkatan. Anda tidak dapat menunda tindakan pemeliharaan jika sudah dimulai.
   > Untuk membatalkan tindakan pemeliharaan, ubah instans DB dan nonaktifkan Peningkatan versi minor otomatis.

AWS CLI

Untuk menerapkan pembaruan yang tertunda ke cluster DB, gunakan apply-pending-maintenance-action AWS CLI perintah.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds apply-pending-maintenance-action \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db \
    --apply-action system-update \
    --opt-in-type immediate
```

Untuk Windows:

```
aws rds apply-pending-maintenance-action ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db ^
    --apply-action system-update ^
    --opt-in-type immediate
```

> **Note**
>
> Untuk menunda tindakan pemeliharaan, tentukan `undo-opt-in` untuk `--opt-in-type`. Anda tidak dapat menentukan `undo-opt-in` untuk `--opt-in-type` jika tindakan pemeliharaan sudah dimulai.
> Untuk membatalkan tindakan pemeliharaan, jalankan modify-db-instance AWS CLI perintah dan tentukan`--no-auto-minor-version-upgrade`.

Untuk mengembalikan daftar sumber daya yang memiliki setidaknya satu pembaruan yang tertunda, gunakan describe-pending-maintenance-actions AWS CLI perintah.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds describe-pending-maintenance-actions \
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

Untuk Windows:

```
aws rds describe-pending-maintenance-actions ^
    --resource-identifier arn:aws:rds:us-west-2:001234567890:db:mysql-db
```

Anda juga dapat mengembalikan daftar sumber daya untuk cluster DB dengan menentukan `--filters` parameter `describe-pending-maintenance-actions` AWS CLI perintah. Format untuk perintah `--filters` adalah `Name=`*`filter-name`*`,Value=`*`resource-id`*`,...`.

Berikut adalah nilai yang diterima untuk parameter `Name` dari filter:

- `db-instance-id` – Menerima daftar pengidentifikasi instans DB atau Amazon Resource Name (ARN). Daftar yang ditampilkan hanya mencakup tindakan pemeliharaan yang tertunda untuk instans DB yang diidentifikasi oleh pengidentifikasi atau ARN tersebut.
- `db-cluster-id` – Menerima daftar pengidentifikasi klaster DB atau ARN untuk Amazon Aurora. Daftar yang ditampilkan hanya mencakup tindakan pemeliharaan yang tertunda untuk klaster DB yang diidentifikasi oleh pengidentifikasi atau ARN tersebut.

Misalnya, contoh berikut menampilkan tindakan pemeliharaan yang tertunda untuk klaster DB `sample-cluster1` dan `sample-cluster2`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds describe-pending-maintenance-actions \
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

Untuk Windows:

```
aws rds describe-pending-maintenance-actions ^
 --filters Name=db-cluster-id,Values=sample-cluster1,sample-cluster2
```

RDS API

Untuk menerapkan pembaruan ke klaster DB, panggil operasi `ApplyPendingMaintenanceAction` Amazon RDS API.

Untuk menampilkan daftar sumber daya yang memiliki setidaknya satu pembaruan tertunda, panggil operasi `DescribePendingMaintenanceActions` Amazon RDS API.

Periode pemeliharaan Amazon RDS

Setiap klaster DB memiliki periode pemeliharaan mingguan di mana setiap perubahan sistem diterapkan. Anggap periode pemeliharaan sebagai peluang untuk mengontrol ketika modifikasi dan patching perangkat lunak terjadi. Jika peristiwa pemeliharaan dijadwalkan selama satu minggu, ini akan dimulai selama periode pemeliharaan 30 menit yang Anda identifikasi. Sebagian besar peristiwa pemeliharaan juga selesai selama periode pemeliharaan 30 menit, meskipun peristiwa pemeliharaan yang lebih besar bisa memakan waktu lebih dari 30 menit.

Periode pemeliharaan 30 menit dipilih secara acak dari blok waktu 8 jam per wilayah. Jika Anda tidak menentukan periode pemeliharaan saat membuat klaster DB, RDS akan menetapkan periode pemeliharaan 30 menit pada hari yang dipilih secara acak dalam seminggu.

RDS menggunakan beberapa sumber daya di klaster DB Anda saat pemeliharaan diterapkan. Anda mungkin mendapati efek minimal pada performa. Untuk instans DB, dalam situasi yang jarang terjadi, failover Multi-AZ mungkin diperlukan untuk menyelesaikan pembaruan pemeliharaan.

Setelah itu, Anda dapat menemukan blok waktu untuk setiap wilayah tempat asal periode pemeliharaan default ditetapkan.

Nama Wilayah	Wilayah	Blok Waktu
AS Timur (Ohio)	us-east-2	03.00–11.00 UTC
AS Timur (Virginia Utara)	us-east-1	03.00–11.00 UTC
AS Barat (California Utara)	us-west-1	06.00–14.00 UTC
AS Barat (Oregon)	us-west-2	06.00–14.00 UTC
Afrika (Cape Town)	af-south-1	03.00–11.00 UTC
Asia Pasifik (Hong Kong)	ap-east-1	06.00–14.00 UTC
Asia Pasifik (Hyderabad)	ap-south-2	06.30–14.30 UTC

Nama Wilayah	Wilayah	Blok Waktu
Asia Pasifik (Jakarta)	ap-southeast-3	08.00–16.00 UTC
Asia Pasifik (Melbourne)	ap-southeast-4	11.00–19.00 UTC
Asia Pasifik (Mumbai)	ap-south-1	06.00–14.00 UTC
Asia Pasifik (Osaka)	ap-northeast-3	22.00–23.59 UTC
Asia Pasifik (Seoul)	ap-northeast-2	13.00–21.00 UTC
Asia Pasifik (Singapura)	ap-southeast-1	14.00–22.00 UTC
Asia Pasifik (Sydney)	ap-southeast-2	12.00–20.00 UTC
Asia Pasifik (Tokyo)	ap-northeast-1	13.00–21.00 UTC
Kanada (Pusat)	ca-central-1	03.00–11.00 UTC
Kanada Barat (Calgary)	ca-west-1	18:00 — 02:00 UTC
Tiongkok (Beijing)	cn-north-1	06.00–14.00 UTC
Tiongkok (Ningxia)	cn-northwest-1	06.00–14.00 UTC
Eropa (Frankfurt)	eu-central-1	21.00–05.00 UTC
Eropa (Irlandia)	eu-west-1	22.00–06.00 UTC
Eropa (London)	eu-west-2	22.00–06.00 UTC
Eropa (Milan)	eu-south-1	02.00–10.00 UTC
Eropa (Paris)	eu-west-3	23.59–07.29 UTC
Eropa (Spanyol)	eu-south-2	02.00–10.00 UTC
Eropa (Stockholm)	eu-north-1	23.00–07.00 UTC

Nama Wilayah	Wilayah	Blok Waktu
Eropa (Zürich)	eu-central-2	02.00–10.00 UTC
Israel (Tel Aviv)	il-central-1	03.00–11.00 UTC
Timur Tengah (Bahrain)	me-south-1	06.00–14.00 UTC
Timur Tengah (UEA)	me-central-1	05.00–13.00 UTC
Amerika Selatan (Sao Paulo)	sa-east-1	00.00–08.00 UTC
AWS GovCloud (AS-Timur)	us-gov-east-1	17.00–01.00 UTC
AWS GovCloud (AS-Barat)	us-gov-west-1	06.00–14.00 UTC

Menyesuaikan periode pemeliharaan klaster DB yang dinginkan

Periode pemeliharaan klaster DB Aurora harus berada dalam waktu penggunaan terendah, sehingga kemungkinan memerlukan perubahan dari waktu ke waktu. klaster DB Anda tidak tersedia selama waktu ini hanya jika pembaruan yang diterapkan memerlukan pemadaman. Pemadaman adalah untuk waktu minimal yang diperlukan untuk melakukan pembaruan yang diperlukan.

Konsol

Untuk menyesuaikan periode pemeliharaan klaster DB yang dinginkan

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB yang periode pemeliharaannya ingin diubah.
4. Pilih Ubah.
5. Di bagian Pemeliharaan, perbarui periode pemeliharaan.
6. Pilih Lanjutkan.

Di halaman konfirmasi, tinjau perubahan Anda.

7. Untuk menerapkan perubahan pada periode pemeliharaan secara langsung, pilih Segera di bagian Penjadwalan perubahan.
8. Pilih Ubah klaster untuk menyimpan perubahan Anda.

   Atau, pilih Kembali untuk mengedit perubahan, atau pilih Batal untuk membatalkan perubahan.

AWS CLI

Untuk menyesuaikan jendela pemeliharaan cluster DB yang disukai, gunakan AWS CLI `modify-db-cluster`perintah dengan parameter berikut:

- `--db-cluster-identifier`
- `--preferred-maintenance-window`

Example

Contoh kode berikut mengatur periode pemeliharaan ke Selasa mulai pukul 04.00-04.30 UTC.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
--db-cluster-identifier my-cluster \
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

Untuk Windows:

```
aws rds modify-db-cluster ^
--db-cluster-identifier my-cluster ^
--preferred-maintenance-window Tue:04:00-Tue:04:30
```

RDS API

Untuk menyesuaikan periode pemeliharaan klaster DB yang diinginkan, gunakan operasi Amazon RDS API `ModifyDBCluster` dengan parameter berikut:

- `DBClusterIdentifier`
- `PreferredMaintenanceWindow`

Peningkatan versi minor otomatis untuk klaster DB Aurora

Pengaturan Peningkatan versi minor otomatis menentukan apakah Aurora secara otomatis menerapkan peningkatan ke klaster DB Anda. Peningkatan ini mencakup versi minor baru yang berisi fitur tambahan dan patch yang berisi perbaikan bug.

Pengaturan ini diaktifkan secara default. Untuk setiap klaster DB baru, pilih nilai yang sesuai untuk pengaturan ini. Nilai ini didasarkan pada kepentingannya, masa pakai yang diharapkan, dan jumlah pengujian verifikasi yang Anda lakukan setelah setiap peningkatan.

Untuk petunjuk tentang cara mengaktifkan atau menonaktifkan pengaturan Peningkatan versi minor otomatis, lihat referensi berikut:

- Mengaktifkan peningkatan versi minor otomatis untuk klaster DB Aurora
- Mengaktifkan peningkatan versi minor otomatis untuk instans DB individu dalam klaster DB Aurora

> **Important**
>
> Untuk klaster DB baru dan yang sudah ada, sebaiknya Anda menerapkan pengaturan ini ke klaster DB, bukan ke instans DB di klaster satu per satu. Jika ada instans DB di klaster Anda yang menonaktifkan pengaturan ini, klaster DB tersebut tidak akan ditingkatkan secara otomatis.

Tabel berikut menunjukkan cara kerja pengaturan Peningkatan versi minor otomatis saat diterapkan di tingkat klaster dan instans.

Tindakan	Pengaturan klaster	Pengaturan instans	Klaster ditingkatkan ditingkatkan secara otomatis?
Anda mengaturnya ke True di klaster DB.	True	True untuk semua instans baru dan yang sudah ada	Ya
Anda mengaturnya ke False di klaster DB.	False	False untuk semua instans baru dan yang sudah ada	Tidak

Tindakan	Pengaturan klaster	Pengaturan instans	Klaster ditingkatkan ditingkatkan secara otomatis?
Sebelumnya diatur ke True di klaster DB. Anda mengaturnya ke False setidaknya di satu instans DB.	Perubahan ke False	False untuk satu atau beberapa instans	Tidak
Itu sebelum ditetapkan ke False pada klaster DB. Anda mengaturnya ke True setidaknya di satu instans DB, tetapi tidak semua instans.	False	True untuk satu atau beberapa instans, tetapi tidak semua instans	Tidak
Itu sebelum ditetapkan ke False pada klaster DB. Anda mengaturnya ke True di semua instans DB.	Perubahan ke True	True untuk semua instans	Ya

Peningkatan versi minor otomatis dikomunikasikan terlebih dahulu melalui peristiwa klaster DB Amazon RDS dengan kategori `maintenance` dan ID `RDS-EVENT-0156`. Untuk informasi selengkapnya, lihat Kategori peristiwa dan pesan peristiwa Amazon RDS.

Peningkatan otomatis terjadi selama periode pemeliharaan. Jika instans DB individu di klaster DB memiliki periode pemeliharaan yang berbeda dengan periode pemeliharaan klaster, periode pemeliharaan klaster akan diutamakan.

Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora PostgreSQL, lihat Pembaruan Amazon Aurora PostgreSQL.

Untuk informasi selengkapnya tentang pengaturan Peningkatan versi minor otomatis untuk Aurora MySQL, lihat Mengaktifkan peningkatan otomatis di antara versi minor Aurora MySQL. Untuk informasi umum tentang pembaruan mesin untuk Aurora MySQL, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL.

Mengaktifkan peningkatan versi minor otomatis untuk klaster DB Aurora

Ikuti prosedur umum dalam Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.

Konsol

Di halaman Mengubah klaster DB, di bagian Pemeliharaan, pilih kotak centang Aktifkan peningkatan versi minor otomatis.

AWS CLI

Panggil modify-db-cluster AWS CLI perintahnya. Tentukan nama klaster DB untuk opsi `--db-cluster-identifier` dan `true` untuk opsi `--auto-minor-version-upgrade`. Secara opsional, tentukan opsi `--apply-immediately` untuk segera mengaktifkan pengaturan ini untuk klaster DB Anda.

RDS API

Panggil operasi API ModifyDBCluster dan tentukan nama klaster DB Anda untuk parameter `DBClusterIdentifier` dan `true` untuk parameter `AutoMinorVersionUpgrade`. Atau, atur parameter `ApplyImmediately` untuk `true` untuk segera mengaktifkan pengaturan ini untuk klaster DB Anda.

Mengaktifkan peningkatan versi minor otomatis untuk instans DB individu dalam klaster DB Aurora

Ikuti prosedur umum dalam Memodifikasi instans DB dalam klaster DB.

Konsol

Di halaman Mengubah instans DB, di bagian Pemeliharaan, pilih kotak centang Aktifkan peningkatan versi minor otomatis.

AWS CLI

Panggil modify-db-instance AWS CLI perintahnya. Tentukan nama instans DB Anda untuk opsi `--db-instance-identifier` dan `true` untuk opsi `--auto-minor-version-upgrade`. Secara opsional, tentukan opsi `--apply-immediately` untuk segera mengaktifkan pengaturan ini untuk instans DB Anda. Jalankan perintah `modify-db-instance` terpisah untuk setiap instans DB di klaster.

RDS API

Panggil operasi ModifyDBInstance API dan tentukan nama klaster DB Anda untuk parameter `DBInstanceIdentifier` dan `true` untuk parameter `AutoMinorVersionUpgrade`. Atau, atur parameter `ApplyImmediately` ke `true` untuk segera mengaktifkan pengaturan ini untuk instans DB Anda. Panggil operasi `ModifyDBInstance` terpisah untuk setiap instans DB dalam klaster.

Anda dapat menggunakan perintah CLI seperti berikut ini untuk memeriksa status pengaturan `AutoMinorVersionUpgrade` untuk semua instans DB di klaster Aurora MySQL.

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

Perintah ini menghasilkan output yang serupa dengan berikut:

```
[
  {
      "DBInstanceIdentifier": "db-writer-instance",
      "DBClusterIdentifier": "my-db-cluster-57",
      "AutoMinorVersionUpgrade": true
  },
  {
      "DBInstanceIdentifier": "db-reader-instance1",
      "DBClusterIdentifier": "my-db-cluster-57",
      "AutoMinorVersionUpgrade": false
  },
  {
      "DBInstanceIdentifier": "db-writer-instance2",
      "DBClusterIdentifier": "my-db-cluster-80",
      "AutoMinorVersionUpgrade": true
  },
... output omitted ...
```

Dalam contoh ini, opsi Aktifkan peningkatan versi minor otomatis dinonaktifkan untuk klaster DB `my-db-cluster-57`, karena dinonaktifkan untuk salah satu instans DB dalam klaster.

Memilih frekuensi pembaruan pemeliharaan Aurora MySQL

Anda dapat mengendalikan apakah peningkatan Aurora MySQL sering atau jarang terjadi untuk setiap klaster DB. Pilihan terbaiknya bergantung pada penggunaan Aurora MySQL Anda dan prioritas untuk aplikasi yang berjalan di Aurora. Untuk informasi tentang rilis stabilitas jangka panjang (LTS) Aurora MySQL yang membutuhkan peningkatan yang jarang, lihat Rilis dukungan jangka panjang (LTS) Aurora MySQL.

Anda dapat memilih untuk jarang meningkatkan klaster Aurora MySQL jika sebagian atau semua kondisi berikut berlaku:

- Siklus pengujian untuk aplikasi Anda membutuhkan waktu yang lama untuk setiap pembaruan ke mesin basis data Aurora MySQL.
- Anda memiliki banyak klaster DB atau banyak aplikasi yang semuanya berjalan di versi Aurora MySQL yang sama. Anda lebih suka meningkatkan semua klaster DB dan aplikasi yang terkait secara bersamaan.
- Anda menggunakan Aurora MySQL dan RDS for MySQL. Anda lebih suka untuk tetap menggunakan klaster Aurora MySQL dan instans DB RDS for MySQL yang kompatibel dengan tingkat MySQL yang sama.
- Aplikasi Aurora MySQL Anda sedang dalam produksi atau penting untuk bisnis. Anda tidak dapat memberikan waktu henti untuk peningkatan di luar peristiwa langka untuk patch penting.
- Aplikasi Aurora MySQL Anda tidak dibatasi oleh masalah performa atau celah fitur yang ditangani di versi Aurora MySQL selanjutnya.

Jika faktor-faktor sebelumnya terjadi pada situasi Anda, Anda dapat membatasi jumlah peningkatan paksa untuk klaster DB Aurora MySQL. Anda dapat melakukannya dengan memilih versi Aurora MySQL spesifik yang dikenal sebagai versi "Dukungan Jangka Panjang" (LTS) saat Anda membuat atau meningkatkan klaster DB tersebut. Tindakan ini dapat meminimalkan jumlah siklus peningkatan, siklus pengujian, dan pemadaman terkait peningkatan untuk klaster DB tersebut.

Anda dapat memilih untuk jarang meningkatkan klaster Aurora MySQL jika sebagian atau semua kondisi berikut berlaku:

- Siklus pengujian untuk aplikasi Anda itu mudah dan singkat.

- Aplikasi Anda masih dalam tahap pengembangan.
- Lingkungan basis data Anda menggunakan berbagai versi Aurora MySQL, atau versi Aurora MySQL dan RDS for MySQL. Setiap klaster Aurora MySQL memiliki siklus peningkatan-nya sendiri.
- Anda menunggu peningkatan performa atau peningkatan fitur spesifik sebelum Anda meningkatkan penggunaan Aurora MySQL.

Jika faktor-faktor sebelumnya terjadi pada situasi Anda, Anda dapat mengaktifkan Aurora untuk lebih sering menerapkan peningkatan penting. Untuk melakukannya, tingkatkan klaster DB Aurora MySQL ke Aurora MySQL versi yang lebih baru dari versi LTS. Dengan begitu, peningkatan performa, perbaikan bug, dan fitur-fitur terbaru akan tersedia bagi Anda dengan lebih cepat.

Bekerja dengan pembaruan sistem operasi

Instans DB dalam klaster DB Aurora MySQL dan Aurora PostgreSQL terkadang memerlukan pembaruan sistem operasi. Amazon RDS meningkatkan sistem operasi ke versi yang lebih baru untuk meningkatkan performa basis data dan postur keamanan pelanggan secara keseluruhan. Pembaruan biasanya memerlukan waktu sekitar 10 menit. Pembaruan sistem operasi tidak akan mengubah versi mesin DB atau kelas instans DB dari instans DB.

Sebaiknya Anda memperbarui instans DB pembaca dalam klaster DB terlebih dahulu, kemudian instans DB penulis. Sebaiknya Anda tidak memperbarui instans pembaca dan penulis secara bersamaan, karena Anda mungkin mengalami waktu henti jika terjadi failover.

Kami menyarankan Anda menggunakan Driver AWS JDBC untuk mencapai failover database yang lebih cepat. Untuk informasi selengkapnya, lihat Driver AWS JDBC untuk MySQL dan Driver JDBC untuk PostgreSQL.AWS

Ada dua jenis pembaruan sistem operasi, dibedakan dengan deskripsi yang terlihat dalam tindakan pemeliharaan tertunda pada instans DB:

- Peningkatan distribusi sistem operasi - Digunakan untuk migrasi ke versi utama Amazon Linux terbaru yang didukung. Penjelasannya dalam tindakan pemeliharaan yang tertunda adalah `New Operating System upgrade is available`.
- Patch sistem operasi - Digunakan untuk menerapkan berbagai perbaikan keamanan dan terkadang untuk meningkatkan performa basis data. Penjelasannya dalam tindakan pemeliharaan yang tertunda adalah `New Operating System patch is available`.

Pembaruan sistem operasi bisa opsional atau wajib:

- Pembaruan opsional dapat diterapkan kapan saja. Meskipun pembaruan ini bersifat opsional, sebaiknya Anda menerapkannya secara berkala agar armada RDS Anda tetap diperbarui. RDS tidak menerapkan pembaruan ini secara otomatis.

  Untuk menerima pemberitahuan saat patch sistem operasi opsional yang baru tersedia, Anda dapat berlangganan RDS-EVENT-0230 dalam kategori peristiwa patching keamanan. Untuk informasi tentang berlangganan peristiwa RDS, lihat Berlangganan pemberitahuan peristiwa Amazon RDS.

  > **Note**
  >
  > `RDS-EVENT-0230` tidak berlaku untuk peningkatan distribusi sistem operasi.

  > **Note**
  >
  > Jika Anda menerima `RDS-EVENT-0230` untuk instans DB RDS for SQL Server, pembaruan OS tidak dapat diterapkan melalui tindakan `apply-pending-maintenance`. Untuk informasi selengkapnya, lihat Menerapkan pembaruan untuk klaster DB.

- Pembaruan wajib diperlukan, dan kami mengirim pemberitahuan sebelum pembaruan wajib. Pemberitahuan mungkin berisi tanggal jatuh tempo. Rencanakan untuk menjadwalkan pembaruan sebelum tanggal jatuh tempo ini. Setelah tanggal jatuh tempo yang ditentukan, Amazon RDS secara otomatis meningkatkan sistem operasi untuk instans DB Anda ke versi terbaru selama salah satu periode pemeliharaan yang ditetapkan.

  Peningkatan distribusi sistem operasi bersifat wajib.

> **Note**
>
> Tetap mengikuti semua pembaruan opsional dan wajib mungkin diperlukan untuk memenuhi berbagai kewajiban kepatuhan. Sebaiknya Anda menerapkan semua pembaruan yang disediakan oleh RDS secara rutin selama periode pemeliharaan Anda.

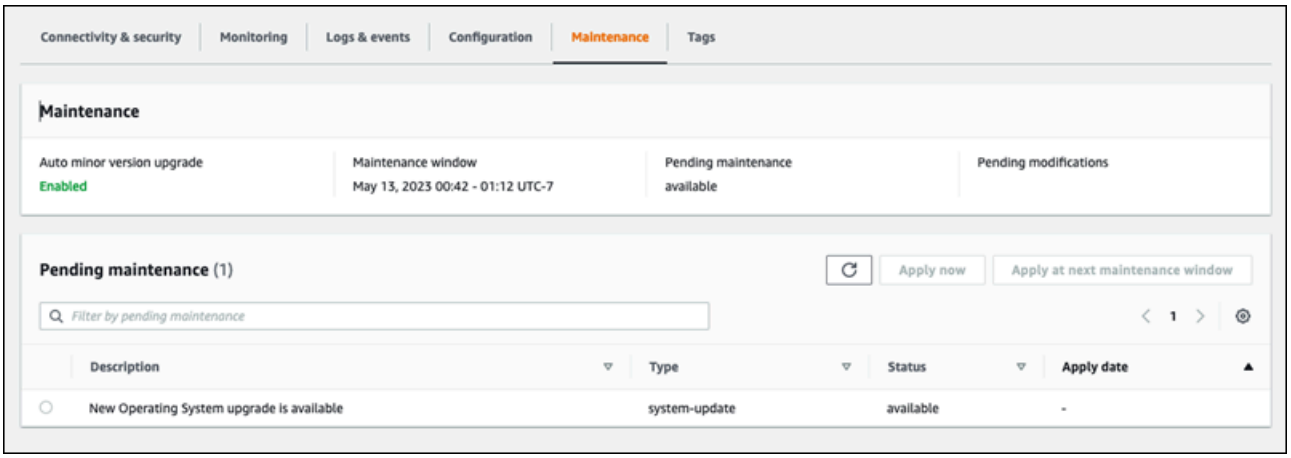
Anda dapat menggunakan AWS Management Console atau AWS CLI untuk mendapatkan informasi tentang jenis upgrade sistem operasi.

Konsol

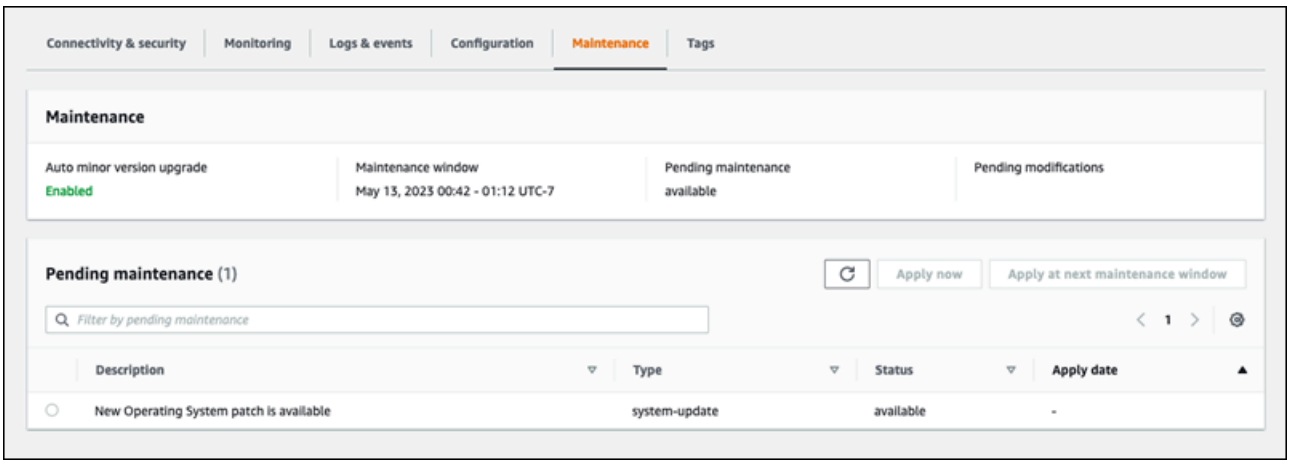
Untuk mendapatkan informasi pembaruan menggunakan AWS Management Console

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, lalu pilih instans DB.
3. Pilih Pemeliharaan .
4. Di bagian Pemeliharaan tertunda, cari pembaruan sistem operasi, dan periksa nilai Deskripsi.

Dalam AWS Management Console, upgrade distribusi sistem operasi memiliki Deskripsi yang disetel ke Upgrade Sistem Operasi Baru tersedia, seperti yang ditunjukkan pada gambar berikut. Peningkatan ini bersifat wajib.



Nilai Deskripsi patch sistem operasi diatur ke Patch Sistem Operasi Baru tersedia, seperti yang ditunjukkan pada gambar berikut.



AWS CLI

Untuk mendapatkan informasi pembaruan dari AWS CLI, gunakan describe-pending-maintenance-actionsperintah.

```
aws rds describe-pending-maintenance-actions
```

Output berikut menunjukkan peningkatan distribusi sistem operasi.

```
{
  "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:mydb1",
  "PendingMaintenanceActionDetails": [
    {
      "Action": "system-update",
      "Description": "New Operating System upgrade is available"
    }
  ]
}
```

Output berikut menunjukkan patch sistem operasi.

```
{
  "ResourceIdentifier": "arn:aws:rds:us-east-1:123456789012:db:mydb2",
  "PendingMaintenanceActionDetails": [
    {
      "Action": "system-update",
      "Description": "New Operating System patch is available"
    }
  ]
```

```
}
```

Ketersediaan pembaruan sistem operasi

Pembaruan sistem operasi khusus untuk versi mesin DB dan kelas instans DB. Oleh karena itu, instans DB menerima atau memerlukan pembaruan di waktu yang berbeda. Ketika pembaruan sistem operasi tersedia untuk instans DB Anda berdasarkan versi mesin dan kelas instansnya, pembaruan akan muncul di konsol. Hal ini juga dapat dilihat dengan menjalankan AWS CLI describe-pending-maintenance-actionsperintah atau dengan memanggil operasi RDS DescribePendingMaintenanceActionsAPI. Jika pembaruan tersedia untuk instans Anda, Anda dapat memperbarui sistem operasi dengan mengikuti petunjuk di Menerapkan pembaruan untuk klaster DB.

Mem-boot ulang klaster DB Amazon Aurora atau instans DB Amazon Aurora

Anda mungkin perlu mem-boot ulang klaster DB atau beberapa instans dalam klaster Anda, biasanya karena alasan pemeliharaan. Sebagai contoh, misalkan Anda mengubah parameter dalam grup parameter atau menghubungkan grup parameter lainnya dengan klaster Anda. Dalam kasus ini, Anda harus mem-boot ulang klaster agar perubahan diterapkan. Sama halnya, Anda mungkin mem-boot ulang satu atau beberapa instans DB pembaca dalam klaster. Anda dapat mengatur operasi boot ulang untuk setiap instans guna meminimalkan waktu henti untuk seluruh klaster.

Waktu yang diperlukan untuk mem-boot ulang setiap instans DB dalam klaster Anda bergantung pada aktivitas basis data pada saat boot ulang. Hal ini juga bergantung pada proses pemulihan mesin DB khusus Anda. Jika memungkinkan, kurangi aktivitas basis data pada instans tertentu sebelum memulai proses boot ulang. Opsi ini dapat mengurangi waktu yang diperlukan untuk memulai ulang basis data.

Anda hanya dapat mem-boot ulang setiap instans DB dalam klaster Anda jika status instans tersebut tersedia. Instans DB mungkin tidak tersedia karena beberapa alasan. Alasan ini meliputi klaster dalam status dihentikan, modifikasi sedang diterapkan pada instans, dan tindakan periode pemeliharaan seperti peningkatan versi.

Boot ulang instans DB akan memulai ulang proses mesin basis data. Mem-boot ulang instans DB akan menyebabkan pemadaman sementara, dan selama pemadaman sementara, status instans DB diatur ke mem-boot ulang.

> **Note**
>
> Jika instans DB tidak menggunakan perubahan terbaru pada grup parameter DB terkait, akan AWS Management Console menampilkan grup parameter DB dengan status pending-reboot. Status grup parameter boot ulang tertunda tidak menyebabkan boot ulang otomatis selama periode pemeliharaan berikutnya. Untuk menerapkan perubahan parameter terbaru pada instans DB tersebut, boot ulang instans DB secara manual. Untuk informasi selengkapnya tentang grup parameter, lihat Bekerja dengan grup parameter.

Topik

- Mem-boot ulang instans DB dalam klaster Aurora
- Mem-boot ulang klaster Aurora dengan ketersediaan baca

- Mem-boot ulang klaster Aurora tanpa ketersediaan baca
- Memeriksa waktu aktif untuk klaster dan instans Aurora
- Contoh operasi boot ulang Aurora

Mem-boot ulang instans DB dalam klaster Aurora

Prosedur ini adalah operasi terpenting yang Anda lakukan saat mem-boot ulang dengan Aurora. Banyak prosedur pemeliharaan yang mewajibkan Anda mem-boot ulang satu atau beberapa instans DB Aurora dalam urutan tertentu.

Konsol

Untuk mem-boot ulang instans DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, lalu pilih instans DB yang ingin di-boot ulang.
3. Untuk Tindakan, pilih Boot ulang.

   Halaman Boot ulang Instans DB akan muncul.
4. Pilih Boot ulang untuk mem-boot ulang instans DB.

   Atau pilih Batalkan.

AWS CLI

Untuk me-reboot instance DB dengan menggunakan AWS CLI, panggil `reboot-db-instance`perintah.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds reboot-db-instance \
    --db-instance-identifier mydbinstance
```

Untuk Windows:

```
aws rds reboot-db-instance ^
```

```
    --db-instance-identifier mydbinstance
```

RDS API

Untuk mem-boot ulang instans DB menggunakan Amazon RDS API, panggil operasi `RebootDBInstance`.

Mem-boot ulang klaster Aurora dengan ketersediaan baca

Dengan fitur ketersediaan baca, Anda dapat me-reboot instance penulis cluster Aurora Anda tanpa me-reboot instance pembaca di cluster DB primer atau sekunder. Tindakan ini dapat membantu menjaga ketersediaan tinggi klaster untuk operasi baca saat Anda mem-boot ulang instans penulis. Anda dapat mem-boot ulang instans pembaca di lain waktu pada jadwal yang dikehendaki. Misalnya, dalam cluster produksi Anda dapat me-reboot instance pembaca satu per satu, dimulai hanya setelah reboot instance utama selesai. Untuk setiap instans DB yang di-boot ulang, ikuti prosedur di Mem-boot ulang instans DB dalam klaster Aurora.

Fitur ketersediaan baca untuk cluster DB primer tersedia di Aurora MySQL versi 2.10 dan lebih tinggi. Ketersediaan baca untuk cluster DB sekunder tersedia di Aurora MySQL versi 3.06 dan lebih tinggi.

Untuk Aurora PostgreSQL, fitur ini tersedia secara default dalam versi berikut:

- Versi 15.2 dan versi 15 yang lebih tinggi
- Versi 14.7 dan versi 14 yang lebih tinggi
- Versi 13.10 dan versi 13 yang lebih tinggi
- Versi 12.14 dan versi 12 yang lebih tinggi

Untuk informasi selengkapnya tentang fitur ketersediaan baca di Aurora PostgreSQL, lihat Meningkatkan ketersediaan baca Aurora Replica.

Sebelum fitur ini, me-reboot instance utama menyebabkan reboot untuk setiap instance pembaca secara bersamaan. Jika klaster Aurora Anda menjalankan versi yang lebih lama, gunakan prosedur boot ulang di Mem-boot ulang klaster Aurora tanpa ketersediaan baca.

> **Note**
>
> Perubahan perilaku reboot di cluster Aurora DB dengan ketersediaan baca berbeda untuk database global Aurora di versi MySQL Aurora lebih rendah dari 3,06. Jika Anda mem-boot ulang instans penulis untuk klaster primer dalam basis data global Aurora, instans pembaca

dalam klaster primer tetap tersedia. Namun, instans DB di setiap klaster sekunder akan di-boot ulang secara bersamaan.
Versi terbatas dari fitur ketersediaan baca yang ditingkatkan didukung oleh database global Aurora untuk Aurora PostgreSQL versi 12.16, 13.12, 14.9, 15.4, dan lebih tinggi.

Anda sering mem-boot ulang klaster setelah menerapkan perubahan pada grup parameter klaster. Anda menerapkan perubahan parameter dengan mengikuti prosedur di Bekerja dengan grup parameter. Misalkan Anda mem-boot ulang instans penulis DB dalam klaster Aurora untuk menerapkan perubahan pada parameter klaster. Beberapa atau semua instans DB pembaca dapat terus menggunakan pengaturan parameter lama. Namun, pengaturan parameter yang berbeda tidak memengaruhi integritas data klaster. Setiap parameter klaster yang memengaruhi susunan file data hanya digunakan oleh instans DB penulis.

Sebagai contoh, dalam klaster Aurora MySQL, Anda dapat memperbarui parameter klaster seperti `binlog_format` dan `innodb_purge_threads` pada instans penulis sebelum instans pembaca. Hanya instans penulis yang menulis log biner dan membersihkan data undo. Untuk parameter yang mengubah cara kueri menafsirkan pernyataan SQL atau output kueri, Anda mungkin perlu segera mem-boot ulang instans pembaca. Dengan demikian, Anda dapat menghindari perilaku aplikasi tak terduga selama kueri. Sebagai contoh, misalkan Anda mengubah parameter `lower_case_table_names` dan mem-boot ulang instans penulis. Dalam kasus ini, instans pembaca mungkin tidak dapat mengakses tabel yang baru dibuat hingga semua instans di-boot ulang.

Untuk daftar semua parameter klaster Aurora MySQL, lihat Parameter tingkat klaster.

Untuk daftar semua parameter klaster Aurora PostgreSQL, lihat Parameter tingkat klaster Aurora PostgreSQL.

Tip

Aurora MySQL mungkin masih mem-boot ulang beberapa instans pembaca beserta instans penulis jika klaster Anda sedang memproses beban kerja dengan throughput yang tinggi. Pengurangan jumlah boot ulang juga berlaku selama operasi failover. Aurora MySQL hanya memulai ulang instans DB penulis dan target failover selama failover. Instans DB pembaca lain dalam klaster ini tetap tersedia untuk terus memproses kueri melalui koneksi ke titik akhir pembaca. Oleh karena itu, Anda dapat meningkatkan ketersediaan selama failover dengan menyediakan lebih dari satu instans DB pembaca dalam sebuah klaster.

Mem-boot ulang klaster Aurora tanpa ketersediaan baca

Dengan fitur ketersediaan baca, Anda dapat mem-boot ulang seluruh klaster DB Aurora dengan mem-boot ulang instans DB dalam klaster tersebut. Untuk melakukannya, ikuti prosedurnya di Mem-boot ulang instans DB dalam klaster Aurora.

Mem-boot ulang instans DB penulis juga akan memulai boot ulang untuk setiap instans DB pembaca dalam klaster. Dengan begitu, setiap perubahan parameter tingkat klaster diterapkan ke semua instans DB secara bersamaan. Namun, jika semua instans DB di-boot ulang, klaster tersebut akan mengalami pemadaman singkat. Instans DB pembaca tetap tidak tersedia hingga instans DB penulis menyelesaikan boot ulang dan menjadi tersedia.

Perilaku boot ulang ini berlaku untuk semua klaster DB yang dibuat di Aurora MySQL versi 2.09 dan yang lebih rendah.

Untuk Aurora PostgreSQL, perilaku ini berlaku untuk versi berikut:

- 14.6 dan versi 14 yang lebih rendah
- 13.9 dan versi 13 yang lebih rendah
- 12.13 dan versi 12 yang lebih rendah
- Semua PostgreSQL versi 11

Dalam konsol RDS, instans DB penulis memiliki nilai Penulis di bawah kolom Peran pada halaman Basis Data. Dalam RDS CLI, output perintah `describe-db-clusters` mencakup bagian `DBClusterMembers`. Elemen `DBClusterMembers` yang mewakili instans DB penulis memiliki nilai `true` untuk kolom `IsClusterWriter`.

> **Important**
>
> Dengan fitur ketersediaan baca, perilaku boot ulang ini berbeda di Aurora MySQL dan Aurora PostgreSQL: instans DB pembaca biasanya tetap tersedia saat Anda mem-boot ulang instans penulis. Kemudian, Anda dapat mem-boot ulang instans pembaca pada waktu yang tepat. Anda dapat mem-boot ulang instans pembaca pada jadwal yang berurutan jika menginginkan beberapa instans pembaca selalu tersedia. Untuk informasi selengkapnya, lihat Mem-boot ulang klaster Aurora dengan ketersediaan baca.

Memeriksa waktu aktif untuk klaster dan instans Aurora

Anda dapat memeriksa dan memantau jangka waktu sejak boot ulang terakhir untuk setiap instans DB di klaster Aurora Anda. CloudWatch Metrik Amazon `EngineUptime` melaporkan jumlah detik sejak terakhir kali instans DB dimulai. Anda dapat memeriksa metrik ini pada titik waktu tertentu guna mengetahui waktu aktif instans DB. Anda juga dapat memantau metrik ini dari waktu ke waktu untuk mendeteksi kapan instans di-boot ulang.

Anda juga dapat memeriksa metrik `EngineUptime` pada tingkat klaster. Dimensi `Minimum` dan `Maximum` melaporkan nilai waktu aktif terkecil dan terbesar untuk semua instans DB dalam klaster. Untuk memeriksa waktu terbaru saat setiap instans pembaca dalam klaster di-boot ulang, atau dimulai ulang karena alasan lain, pantau metrik tingkat klaster menggunakan dimensi `Minimum`. Untuk memeriksa instans dalam klaster telah berjalan paling lama tanpa boot ulang, pantau metrik tingkat klaster menggunakan dimensi `Maximum`. Sebagai contoh, sebaiknya Anda mengonfirmasi bahwa semua instans DB dalam klaster di-boot ulang setelah perubahan konfigurasi.

> **Tip**
>
> Untuk pemantauan jangka panjang, sebaiknya Anda memantau metrik `EngineUptime` untuk setiap instans, bukan di tingkat klaster. Metrik `EngineUptime` tingkat klaster diatur ke nol saat instans DB baru ditambahkan ke klaster. Perubahan klaster tersebut bisa terjadi sebagai bagian dari operasi pemeliharaan dan penskalaan seperti yang dilakukan oleh Penskalaan Otomatis.

Contoh CLI berikut menunjukkan cara memeriksa metrik `EngineUptime` untuk instans penulis dan pembaca dalam klaster. Contoh tersebut menggunakan sebuah klaster bernama `tpch100g`. Klaster ini memiliki instans DB penulis `instance-1234`. Klaster ini juga memiliki dua instans DB pembaca, `instance-7448` dan `instance-6305`.

Pertama, perintah `reboot-db-instance` mem-boot ulang salah satu instans pembaca. Perintah `wait` menunggu hingga instans selesai di-boot ulang.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
...
```

```
$ aws rds wait db-instance-available --db-instance-id instance-6305
```

CloudWatch `get-metric-statistics`Perintah memeriksa `EngineUptime` metrik selama lima menit terakhir dengan interval satu menit. Waktu aktif untuk instans `instance-6305` diatur ulang ke nol dan mulai menghitung ke atas lagi. AWS CLI Contoh untuk Linux ini menggunakan substitusi `$()` variabel untuk memasukkan stempel waktu yang sesuai ke dalam perintah CLI. Contoh ini juga menggunakan perintah `sort` Linux untuk mengurutkan output pada saat metrik dikumpulkan. Nilai stempel waktu tersebut adalah kolom ketiga di setiap baris output.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 231.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 291.0 2021-03-16T18:20:00+00:00 Seconds
DATAPOINTS 351.0 2021-03-16T18:21:00+00:00 Seconds
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
```

Waktu aktif minimum untuk klaster diatur ulang ke nol karena salah satu instans dalam klaster di-boot ulang. Waktu aktif maksimum untuk klaster tidak diatur ulang karena setidaknya salah satu instans DB dalam klaster tetap tersedia.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Minimum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63099.0 2021-03-16T18:12:00+00:00 Seconds
DATAPOINTS 63159.0 2021-03-16T18:13:00+00:00 Seconds
DATAPOINTS 63219.0 2021-03-16T18:14:00+00:00 Seconds
DATAPOINTS 63279.0 2021-03-16T18:15:00+00:00 Seconds
DATAPOINTS 51.0 2021-03-16T18:16:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBClusterIdentifier,Value=tpch100g --output text \
```

```
  | sort -k 3
EngineUptime
DATAPOINTS 63389.0 2021-03-16T18:16:00+00:00 Seconds
DATAPOINTS 63449.0 2021-03-16T18:17:00+00:00 Seconds
DATAPOINTS 63509.0 2021-03-16T18:18:00+00:00 Seconds
DATAPOINTS 63569.0 2021-03-16T18:19:00+00:00 Seconds
DATAPOINTS 63629.0 2021-03-16T18:20:00+00:00 Seconds
```

Kemudian, perintah `reboot-db-instance` lainnya mem-boot ulang instans penulis pada klaster. Perintah `wait` lainnya dijeda hingga instans penulis selesai di-boot ulang.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
  "DBInstanceIdentifier": "instance-1234",
  "DBInstanceStatus": "rebooting",
...
$ aws rds wait db-instance-available --db-instance-id instance-1234
```

Saat ini, metrik `EngineUptime` untuk instans penulis menunjukkan bahwa instans `instance-1234` baru-baru ini di-boot ulang. Instans pembaca `instance-6305` juga di-boot ulang secara otomatis bersama dengan instans penulis. Klaster ini menjalankan Aurora MySQL 2.09, yang tidak menjaga instans pembaca tetap berjalan saat instans penulis di-boot ulang.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-1234 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 63749.0 2021-03-16T18:22:00+00:00 Seconds
DATAPOINTS 63809.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 63869.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 41.0 2021-03-16T18:25:00+00:00 Seconds
DATAPOINTS 101.0 2021-03-16T18:26:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" \
  --period 60 --namespace "AWS/RDS" --statistics Maximum \
  --dimensions Name=DBInstanceIdentifier,Value=instance-6305 --output text \
  | sort -k 3
EngineUptime
DATAPOINTS 411.0 2021-03-16T18:22:00+00:00 Seconds
```

```
DATAPOINTS 471.0 2021-03-16T18:23:00+00:00 Seconds
DATAPOINTS 531.0 2021-03-16T18:24:00+00:00 Seconds
DATAPOINTS 49.0 2021-03-16T18:26:00+00:00 Seconds
```

Contoh operasi boot ulang Aurora

Contoh Aurora MySQL berikut menunjukkan beberapa kombinasi operasi boot ulang untuk instans DB pembaca dan penulis dalam klaster DB Aurora. Setelah setiap boot ulang, kueri SQL menunjukkan waktu aktif untuk instans dalam klaster.

Topik

- Menemukan instans penulis dan pembaca untuk klaster Aurora
- Mem-boot ulang satu instans pembaca
- Mem-boot ulang instans penulis
- Mem-boot ulang penulis dan pembaca secara independen
- Menerapkan perubahan parameter klaster ke klaster Aurora MySQL versi 2.10

Menemukan instans penulis dan pembaca untuk klaster Aurora

Dalam klaster Aurora MySQL dengan beberapa instans DB, Anda harus mengetahui instans mana yang merupakan instans penulis dan mana yang merupakan instans pembaca. Instans penulis dan pembaca juga dapat beralih peran ketika terjadi operasi failover. Maka, sangat disarankan untuk memeriksa hal berikut sebelum melakukan operasi apa pun yang membutuhkan instans penulis atau pembaca. Dalam kasus ini, nilai `False` untuk `IsClusterWriter` mengidentifikasi instans pembaca, `instance-6305` dan `instance-7448`. Nilai `True` mengidentifikasi instans penulis, `instance-1234`.

```
$ aws rds describe-db-clusters --db-cluster-id tpch100g \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
Cluster:     tpch100g
Instance:    instance-6305    False
Instance:    instance-7448    False
Instance:    instance-1234    True
```

Sebelum kita memulai contoh boot ulang, instans penulis memiliki waktu aktif sekitar satu minggu. Kueri SQL dalam contoh ini menunjukkan cara khusus MySQL dalam memeriksa waktu aktif. Anda

dapat menggunakan teknik ini dalam aplikasi basis data. Untuk teknik lain yang menggunakan AWS CLI dan bekerja untuk kedua mesin Aurora, lihat. Memeriksa waktu aktif untuk klaster dan instans Aurora

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-08 17:49:06.000000 | 174h 42m|
+----------------------------+---------+
```

Mem-boot ulang satu instans pembaca

Contoh ini mem-boot ulang salah satu instans DB pembaca. Mungkin instans ini mengalami kelebihan beban oleh kueri besar atau banyak koneksi bersamaan. Atau mungkin tertinggal di belakang instans penulis karena masalah jaringan. Setelah memulai operasi boot ulang, contoh tersebut menggunakan perintah `wait` untuk menjeda hingga instans menjadi tersedia. Pada saat itu, instans memiliki waktu aktif beberapa menit.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6305
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-6305",
        "DBInstanceStatus": "rebooting",
...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-6305
$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status
    -> where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
```

```
| 2021-03-16 00:35:02.000000 | 00h 03m |
+----------------------------+---------+
```

Boot ulang instans pembaca tidak memengaruhi waktu aktif instans penulis. Instans ini tetap memiliki waktu aktif sekitar satu minggu.

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+----------+
| Last Startup               | Uptime   |
+----------------------------+----------+
| 2021-03-08 17:49:06.000000 | 174h 49m |
+----------------------------+----------+
```

Mem-boot ulang instans penulis

Contoh ini mem-boot ulang instans penulis. Klaster ini menjalankan Aurora MySQL versi 2.09. Karena versi Aurora MySQL lebih rendah dari 2.10, mem-boot ulang instans penulis juga akan mem-boot ulang setiap instans pembaca dalam klaster.

Perintah `wait` dijeda hingga proses boot ulang selesai. Sekarang, waktu aktif untuk instans tersebut diatur ulang ke nol. Ada kemungkinan bahwa operasi boot ulang mungkin memakan waktu yang sangat berbeda untuk instans DB penulis dan pembaca. Instans DB penulis dan pembaca melakukan berbagai jenis operasi pembersihan bergantung pada perannya.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-1234
{
    "DBInstance": {
        "DBInstanceIdentifier": "instance-1234",
        "DBInstanceStatus": "rebooting",
...
    }
}
$ aws rds wait db-instance-available --db-instance-id instance-1234
$ mysql -h instance-1234.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
```

```
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:27.000000 | 00h 00m |
+----------------------------+---------+
```

Setelah instans DB penulis di-boot ulang, waktu aktif kedua instans DB pembaca juga akan diatur ulang. Mem-boot ulang instans penulis juga menyebabkan instans pembaca di-boot ulang. Perilaku ini berlaku untuk klaster Aurora PostgreSQL dan Aurora MySQL sebelum versi 2.10.

```
$ mysql -h instance-7448.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:35.000000 | 00h 00m |
+----------------------------+---------+

$ mysql -h instance-6305.a12345.us-east-1.rds.amazonaws.com -P 3306 -u my-user -p
...
mysql> select date_sub(now(), interval variable_value second) "Last Startup",
    -> time_format(sec_to_time(variable_value),'%Hh %im') as "Uptime"
    -> from performance_schema.global_status where variable_name='Uptime';
+----------------------------+---------+
| Last Startup               | Uptime  |
+----------------------------+---------+
| 2021-03-16 00:40:33.000000 | 00h 01m |
+----------------------------+---------+
```

Mem-boot ulang penulis dan pembaca secara independen

Contoh-contoh berikut menunjukkan klaster yang menjalankan Aurora MySQL versi 2.10. Dalam Aurora MySQL versi ini dan yang lebih tinggi, Anda dapat mem-boot ulang instans penulis tanpa menyebabkan boot ulang untuk semua instans pembaca. Dengan demikian, aplikasi padat kueri Anda tidak mengalami pemadaman saat Anda mem-boot ulang instans penulis. Anda dapat mem-boot ulang instans pembaca di lain waktu. Anda juga dapat melakukan beberapa boot ulang saat lalu lintas kueri rendah. Anda juga dapat mem-boot ulang beberapa instans pembaca satu per satu.

Dengan demikian, setidaknya satu instans pembaca selalu tersedia untuk lalu lintas kueri aplikasi Anda.

Contoh berikut menggunakan sebuah klaster bernama `cluster-2393`, yang menjalankan Aurora MySQL versi `5.7.mysql_aurora.2.10.0`. Klaster ini memiliki instans penulis bernama `instance-9404` dan tiga instans pembaca bernama `instance-6772`, `instance-2470`, dan `instance-5138`.

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
  --query "*[].['Cluster:',DBClusterIdentifier,DBClusterMembers[*].
['Instance:',DBInstanceIdentifier,IsClusterWriter]]" \
  --output text
Cluster:        cluster-2393
Instance:       instance-5138        False
Instance:       instance-2470        False
Instance:       instance-6772        False
Instance:       instance-9404        True
```

Memeriksa nilai `uptime` dari setiap instans basis data melalui perintah `mysql` menunjukkan bahwa setiap instans memiliki waktu aktif yang kurang lebih sama. Contoh berikut adalah uptime untuk `instance-5138`.

```
mysql> SHOW GLOBAL STATUS LIKE 'uptime';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Uptime        | 3866  |
+---------------+-------+
```

Dengan menggunakan CloudWatch, kita bisa mendapatkan informasi uptime yang sesuai tanpa benar-benar masuk ke instance. Dengan demikian, administrator dapat memantau basis data, tetapi tidak dapat melihat atau mengubah data tabel. Dalam hal ini, kita menentukan jangka waktu selama lima menit, dan memeriksa nilai waktu aktif setiap menit. Peningkatan nilai waktu aktif menunjukkan bahwa instans tidak dimulai ulang selama periode tersebut.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
```

```
EngineUptime
DATAPOINTS 4648.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4708.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4768.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4828.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4888.0 2021-03-17T23:46:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4315.0 2021-03-17T23:42:00+00:00 Seconds
DATAPOINTS 4375.0 2021-03-17T23:43:00+00:00 Seconds
DATAPOINTS 4435.0 2021-03-17T23:44:00+00:00 Seconds
DATAPOINTS 4495.0 2021-03-17T23:45:00+00:00 Seconds
DATAPOINTS 4555.0 2021-03-17T23:46:00+00:00 Seconds
```

Sekarang kita mem-boot ulang salah satu instans pembaca, `instance-5138`. Kita tunggu instans tersedia lagi setelah boot ulang. Sekarang, pemantauan waktu aktif selama jangka waktu lima menit menunjukkan bahwa waktu aktif diatur ulang ke nol selama waktu tersebut. Nilai waktu aktif terbaru diukur lima detik setelah boot ulang selesai.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 4500.0 2021-03-17T23:46:00+00:00 Seconds
DATAPOINTS 4560.0 2021-03-17T23:47:00+00:00 Seconds
DATAPOINTS 4620.0 2021-03-17T23:48:00+00:00 Seconds
DATAPOINTS 4680.0 2021-03-17T23:49:00+00:00 Seconds
DATAPOINTS  5.0 2021-03-17T23:50:00+00:00 Seconds
```

Selanjutnya, kita lakukan boot ulang untuk instans penulis, `instance-9404`. Kita bandingkan nilai waktu aktif untuk instans penulis dan salah satu instans pembaca. Dengan demikian, kita dapat melihat bahwa mem-boot ulang penulis tidak akan mem-boot ulang pembaca. Dalam versi sebelum Aurora MySQL 2.10, nilai-nilai uptime untuk semua pembaca akan diatur ulang pada waktu yang sama dengan penulis.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
{
  "DBInstanceIdentifier": "instance-9404",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-9404

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 371.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 431.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 491.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 551.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS  37.0  2021-03-18T00:01:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-6772 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 5215.0 2021-03-17T23:57:00+00:00 Seconds
DATAPOINTS 5275.0 2021-03-17T23:58:00+00:00 Seconds
DATAPOINTS 5335.0 2021-03-17T23:59:00+00:00 Seconds
DATAPOINTS 5395.0 2021-03-18T00:00:00+00:00 Seconds
DATAPOINTS 5455.0 2021-03-18T00:01:00+00:00 Seconds
```

Untuk memastikan bahwa semua instans pembaca memiliki semua perubahan pada parameter konfigurasi yang sama dengan instans penulis, boot ulang seluruh instans pembaca setelah penulis. Contoh ini mem-boot ulang semua pembaca, lalu menunggu hingga semua pembaca tersedia sebelum melanjutkan.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-6772
{
  "DBInstanceIdentifier": "instance-6772",
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceStatus": "rebooting"
}

$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}

$ aws rds wait db-instance-available --db-instance-id instance-6772
$ aws rds wait db-instance-available --db-instance-id instance-2470
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

Sekarang kita dapat melihat bahwa instans DB penulis memiliki waktu aktif tertinggi. Nilai waktu aktif instans ini meningkat dengan stabil selama periode pemantauan. Semua instans DB pembaca di-boot ulang setelah pembaca. Kita dapat melihat titik dalam periode pemantauan saat setiap pembaca di-boot ulang dan waktu aktif-nya diatur ulang ke nol.

```
$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-9404 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 457.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 517.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 577.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 637.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS  697.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
```

```
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-2470 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 5819.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS  35.0  2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 95.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS 155.0 2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 215.0 2021-03-18T00:12:00+00:00 Seconds

$ aws cloudwatch get-metric-statistics --metric-name "EngineUptime" \
  --start-time "$(date -d '5 minutes ago')" --end-time "$(date -d 'now')" --period 60 \
  --namespace "AWS/RDS" --statistics Minimum --dimensions
 Name=DBInstanceIdentifier,Value=instance-5138 \
  --output text | sort -k 3
EngineUptime
DATAPOINTS 1085.0 2021-03-18T00:08:00+00:00 Seconds
DATAPOINTS 1145.0 2021-03-18T00:09:00+00:00 Seconds
DATAPOINTS 1205.0 2021-03-18T00:10:00+00:00 Seconds
DATAPOINTS  49.0  2021-03-18T00:11:00+00:00 Seconds
DATAPOINTS 109.0 2021-03-18T00:12:00+00:00 Seconds
```

Menerapkan perubahan parameter klaster ke klaster Aurora MySQL versi 2.10

Contoh berikut menunjukkan cara menerapkan perubahan parameter untuk semua instans DB dalam klaster Aurora MySQL 2.10 Anda. Dengan Aurora MySQL versi ini, Anda mem-boot ulang instans penulis dan semua instans pembaca secara independen.

Contoh ini menggunakan parameter konfigurasi MySQL `lower_case_table_names` sebagai ilustrasi. Jika pengaturan parameter ini berbeda antara instans DB penulis dan pembaca, kueri mungkin tidak dapat mengakses tabel yang dinyatakan dengan nama dalam huruf besar atau campuran huruf besar dan kecil. Atau, jika dua nama tabel hanya berbeda dari segi huruf besar dan huruf kecil, kueri mungkin mengakses tabel yang salah.

Contoh ini menunjukkan cara menentukan instans penulis dan pembaca dalam klaster dengan memeriksa atribut `IsClusterWriter` pada setiap instans. Klaster ini bernama `cluster-2393`. Klaster ini memiliki satu instans penulis bernama `instance-9404`. Instans pembaca dalam klaster tersebut bernama `instance-5138` dan `instance-2470`.

```
$ aws rds describe-db-clusters --db-cluster-id cluster-2393 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
```

```
  --output text
cluster-2393
instance-5138        False
instance-2470        False
instance-9404        True
```

Untuk menunjukkan efek dari perubahan parameter `lower_case_table_names`, kita mengatur dua grup parameter klaster DB. Parameter dari grup parameter `lower-case-table-names-0` ini diatur ke 0. Grup parameter dari grup parameter `lower-case-table-names-1` ini diatur ke 1.

```
$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-0' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-0
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-0",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-0"
    }
}

$ aws rds create-db-cluster-parameter-group --description 'lower-case-table-names-1' \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "lower-case-table-names-1",
        "DBParameterGroupFamily": "aurora-mysql5.7",
        "Description": "lower-case-table-names-1"
    }
}

$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lower-case-table-names-0 \
  --parameters
 ParameterName=lower_case_table_names,ParameterValue=0,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-0"
}

$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lower-case-table-names-1 \
```

```
    --parameters
 ParameterName=lower_case_table_names,ParameterValue=1,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "lower-case-table-names-1"
}
```

Nilai default `lower_case_table_names` adalah 0. Dengan pengaturan parameter ini, tabel `foo` berbeda dengan tabel F00. Contoh ini memverifikasi bahwa parameter tersebut masih memiliki pengaturan default. Kemudian, contoh tersebut membuat tiga tabel yang hanya berbeda dari segi huruf besar dan huruf kecil dalam namanya.

```
mysql> create database lctn;
Query OK, 1 row affected (0.07 sec)

mysql> use lctn;
Database changed
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+

mysql> create table foo (s varchar(128));
mysql> insert into foo values ('Lowercase table name foo');

mysql> create table Foo (s varchar(128));
mysql> insert into Foo values ('Mixed-case table name Foo');

mysql> create table FOO (s varchar(128));
mysql> insert into FOO values ('Uppercase table name FOO');

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
```

```
| Mixed-case table name Foo |
+---------------------------+

mysql> select * from FOO;
+---------------------------+
| s                         |
+---------------------------+
| Uppercase table name FOO |
+---------------------------+
```

Selanjutnya, kita mengaitkan grup parameter DB dengan klaster untuk mengatur parameter `lower_case_table_names` ke 1. Perubahan ini hanya diterapkan setelah setiap instans DB di-boot ulang.

```
$ aws rds modify-db-cluster --db-cluster-identifier cluster-2393 \
  --db-cluster-parameter-group-name lower-case-table-names-1
{
  "DBClusterIdentifier": "cluster-2393",
  "DBClusterParameterGroup": "lower-case-table-names-1",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.0"
}
```

Boot ulang pertama yang kita lakukan adalah untuk instans DB penulis. Kemudian, kita tunggu hingga instans tersedia lagi. Pada saat itu, kita terhubung ke titik akhir penulis dan memverifikasi bahwa nilai parameter instans penulis berubah. Perintah `SHOW TABLES` menegaskan bahwa basis data berisi tiga tabel yang berbeda. Namun, semua kueri yang merujuk ke tabel bernama `foo`, `Foo`, atau F00 mengakses tabel yang namanya berisi huruf kecil semua, `foo`.

```
# Rebooting the writer instance
$ aws rds reboot-db-instance --db-instance-identifier instance-9404
$ aws rds wait db-instance-available --db-instance-id instance-9404
```

Sekarang, kueri yang menggunakan titik akhir klaster menunjukkan efek dari perubahan parameter. Baik huruf besar, huruf kecil, maupun gabungan huruf kecil dan besar pada name tabel dalam kueri, pernyataan SQL mengakses tabel yang namanya memiliki huruf kecil semua.

```
mysql> select @@lower_case_table_names;
+--------------------------+
| @@lower_case_table_names |
```

```
+--------------------------+
|                        1 |
+--------------------------+

mysql> use lctn;
mysql> show tables;
+----------------+
| Tables_in_lctn |
+----------------+
| FOO            |
| Foo            |
| foo            |
+----------------+

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+
```

Contoh berikutnya menunjukkan kueri yang sama seperti sebelumnya. Dalam hal ini, kueri menggunakan titik akhir pembaca dan berjalan di salah satu instans DB pembaca. Instans tersebut belum di-boot ulang. Dengan demikian, kueri tersebut masih memiliki pengaturan awal untuk parameter `lower_case_table_names`. Hal ini berarti bahwa kueri dapat mengakses setiap tabel `foo`, `Foo`, dan `FOO`.

```
mysql> select @@lower_case_table_names;
+--------------------------+
```

```
| @@lower_case_table_names |
+--------------------------+
|                        0 |
+--------------------------+

mysql> use lctn;

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+---------------------------+
| s                         |
+---------------------------+
| Mixed-case table name Foo |
+---------------------------+

mysql> select * from FOO;
+--------------------------+
| s                        |
+--------------------------+
| Uppercase table name FOO |
+--------------------------+
```

Selanjutnya, kita mem-boot ulang salah satu instans pembaca dan menunggu hingga instans tersebut tersedia lagi.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-2470
{
  "DBInstanceIdentifier": "instance-2470",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-2470
```

Selagi terhubung ke titik akhir instans untuk `instance-2470`, satu kueri menunjukkan bahwa parameter baru sedang diterapkan.

```
mysql> select @@lower_case_table_names;
```

```
+--------------------------+
| @@lower_case_table_names |
+--------------------------+
|                        1 |
+--------------------------+
```

Pada titik ini, dua instans pembaca dalam klaster sedang berjalan dengan pengaturan `lower_case_table_names` yang berbeda. Dengan demikian, setiap koneksi ke titik akhir pembaca dalam klaster menggunakan nilai untuk pengaturan yang tidak dapat diprediksi ini. Instans pembaca lain juga perlu segera di-boot ulang agar keduanya memiliki pengaturan yang konsisten.

```
$ aws rds reboot-db-instance --db-instance-identifier instance-5138
{
  "DBInstanceIdentifier": "instance-5138",
  "DBInstanceStatus": "rebooting"
}
$ aws rds wait db-instance-available --db-instance-id instance-5138
```

Contoh berikut menegaskan bahwa semua instans pembaca memiliki pengaturan yang sama untuk parameter `lower_case_table_names`. Perintah ini memeriksa nilai pengaturan `lower_case_table_names` pada setiap instans pembaca. Kemudian, perintah yang sama yang menggunakan titik akhir pembaca menunjukkan bahwa setiap koneksi ke titik akhir pembaca menggunakan salah satu instans pembaca, tetapi tidak dapat memprediksi instans yang mana.

```
# Check lower_case_table_names setting on each reader instance.

$ mysql -h instance-5138.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-5138            |                        1 |
+--------------------------+--------------------------+

$ mysql -h instance-2470.a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-2470            |                        1 |
+--------------------------+--------------------------+
```

```
# Check lower_case_table_names setting on the reader endpoint of the cluster.

$ mysql -h cluster-2393.cluster-ro-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-5138            |                        1 |
+--------------------------+--------------------------+

# Run query on writer instance

$ mysql -h cluster-2393.cluster-a12345.us-east-1.rds.amazonaws.com \
  -u my-user -p -e 'select @@aurora_server_id, @@lower_case_table_names'
+--------------------------+--------------------------+
| @@aurora_server_id       | @@lower_case_table_names |
+--------------------------+--------------------------+
| instance-9404            |                        1 |
+--------------------------+--------------------------+
```

Dengan perubahan parameter yang diterapkan di mana-mana, kita dapat melihat efek pengaturan `lower_case_table_names=1`. Baik tabel ini disebut sebagai `foo`, `Foo`, maupun `F00`, kueri akan mengonversi nama menjadi `foo` dan mengakses tabel yang sama dalam setiap kasus.

```
mysql> use lctn;

mysql> select * from foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from Foo;
+--------------------------+
| s                        |
+--------------------------+
| Lowercase table name foo |
+--------------------------+

mysql> select * from FOO;
+--------------------------+
```

```
| s                         |
+---------------------------+
| Lowercase table name foo |
+---------------------------+
```

Menghapus instans DB atau klaster DB Aurora

Anda dapat menghapus klaster DB Aurora jika tidak lagi membutuhkannya. Menghapus klaster akan menghapus volume klaster yang berisi semua data Anda. Sebelum menghapus klaster, Anda dapat menyimpan snapshot data Anda. Anda dapat memulihkan snapshot di lain waktu dan membuat klaster baru yang berisi data yang sama.

Anda juga dapat menghapus instans DB dari suatu klaster, sekaligus mempertahankan klaster itu sendiri dan data yang disimpannya. Menghapus instans DB dapat membantu Anda mengurangi biaya jika klaster tidak sibuk, atau Anda tidak memerlukan kapasitas perhitungan beberapa instans DB.

Topik

- Menghapus klaster DB Aurora
- Perlindungan penghapusan untuk klaster Aurora
- Menghapus klaster Aurora yang berhenti
- Menghapus klaster Aurora MySQL yang merupakan replika baca
- Snapshot akhir saat menghapus klaster
- Menghapus instans dari klaster DB Aurora

Menghapus klaster DB Aurora

Aurora tidak menyediakan metode satu langkah untuk menghapus klaster DB. Pilihan desain ini dimaksudkan untuk mencegah Anda kehilangan data secara tidak sengaja atau mengambil aplikasi Anda secara offline. Aplikasi Aurora biasanya bersifat kritis misi dan membutuhkan ketersediaan tinggi. Dengan demikian, Aurora memudahkan untuk memperbesar dan memperkecil skala kapasitas klaster dengan menambah dan menghapus instans DB. Menghapus klaster DB itu sendiri mengharuskan Anda membuat penghapusan terpisah.

Gunakan prosedur umum berikut untuk menghapus semua instans DB dari klaster, kemudian menghapus klaster itu sendiri.

1. Hapus setiap instans pembaca dalam klaster. Gunakan prosedur di Menghapus instans dari klaster DB Aurora.

   Jika klaster memiliki instans pembaca, menghapus salah satu instans hanya akan mengurangi kapasitas komputasi klaster. Menghapus instans pembaca terlebih dahulu akan memastikan

bahwa klaster tetap tersedia di sepanjang prosedur dan tidak melakukan operasi failover yang tidak perlu.

2. Hapus instans penulis dari klaster. Gunakan lagi prosedur di Menghapus instans dari klaster DB Aurora.

   Saat Anda menghapus instans DB, klaster dan volume klaster terkaitnya tidak akan berubah bahkan setelah Anda menghapus semua instans DB.

3. Hapus klaster DB.

   - AWS Management Console – Pilih klaster, lalu pilih Hapus dari menu Tindakan. Anda dapat memilih opsi berikut untuk mempertahankan data dari klaster jika Anda membutuhkannya nanti:
     - Buat snapshot akhir dari volume klaster. Pengaturan default-nya adalah membuat snapshot akhir.
     - Pertahankan cadangan otomatis. Pengaturan default bukan untuk mempertahankan cadangan otomatis.

     > **Note**
     >
     > Cadangan otomatis untuk klaster DB Aurora Serverless v1 tidak dipertahankan.

     Aurora juga meminta Anda mengonfirmasi bahwa Anda berniat untuk menghapus klaster.
   - CLI dan API – Panggil perintah CLI `delete-db-cluster` atau operasi API `DeleteDBCluster`. Anda dapat memilih opsi berikut untuk mempertahankan data dari klaster jika Anda membutuhkannya nanti:
     - Buat snapshot akhir dari volume klaster.
     - Pertahankan cadangan otomatis.

     > **Note**
     >
     > Cadangan otomatis untuk klaster DB Aurora Serverless v1 tidak dipertahankan.

Topik

- Menghapus klaster Aurora kosong
- Menghapus klaster Aurora dengan satu instans DB
- Menghapus klaster Aurora dengan beberapa instans DB

Menghapus klaster Aurora kosong

Anda dapat menghapus klaster DB kosong menggunakan AWS Management Console, AWS CLI, atau Amazon RDS API.

> **Tip**
>
> Anda dapat menyimpan klaster tanpa instans DB untuk mempertahankan data tanpa menimbulkan biaya CPU untuk klaster. Anda dapat segera mulai menggunakan klaster tersebut lagi dengan membuat satu atau beberapa instans DB baru untuk klaster tersebut. Anda dapat melakukan operasi administratif spesifik Aurora pada klaster meskipun klaster tersebut tidak memiliki instans DB terkait. Anda hanya tidak dapat mengakses data atau melakukan operasi apa pun yang memerlukan koneksi ke instans DB.

Konsol

Untuk menghapus klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih klaster DB yang ingin dihapus.
3. Untuk Tindakan, pilih Hapus.
4. Untuk membuat snapshot DB akhir untuk klaster DB, pilih Buat snapshot akhir?. Ini adalah pengaturan default.
5. Jika Anda memilih untuk membuat snapshot akhir, masukkan Nama snapshot akhir.
6. Untuk mempertahankan cadangan otomatis, pilih Pertahankan cadangan otomatis. Ini bukan pengaturan default.
7. Masukkan **`delete me`** di kotak.
8. Pilih Hapus.

CLI

Untuk menghapus cluster Aurora DB kosong dengan menggunakanAWS CLI, panggil perintah. delete-db-cluster

Misalkan klaster kosong `deleteme-zero-instances` hanya digunakan untuk pengembangan dan pengujian dan tidak berisi data penting. Dalam hal ini, Anda tidak perlu untuk mempertahankan snapshot volume klaster saat menghapus klaster. Contoh berikut menunjukkan bahwa klaster tidak berisi instans DB, kemudian menghapus klaster kosong tanpa membuat snapshot akhir atau mempertahankan cadangan otomatis.

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-zero-instances --output
 text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-zero-instances

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-zero-instances \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-zero-instances",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

RDS API

Untuk menghapus klaster DB Aurora kosong menggunakan Amazon RDS API, panggil operasi DeleteDBCluster.

Menghapus klaster Aurora dengan satu instans DB

Anda dapat menghapus instans DB terakhir, meskipun klaster DB mengaktifkan perlindungan penghapusan. Dalam kasus ini, klaster DB sendiri masih ada dan data Anda dipertahankan. Anda dapat mengakses data lagi dengan melampirkan instans DB baru ke klaster.

Contoh berikut menunjukkan bagaimana perintah `delete-db-cluster` tidak bekerja jika klaster masih memiliki instans DB terkait. Klaster ini memiliki satu instans DB penulis. Saat memeriksa instans DB di klaster, kita memeriksa atribut `IsClusterWriter` masing-masing. Klaster bisa memiliki nol atau satu instans DB penulis. Nilai `true` menandakan instans DB penulis. Nilai `false` menandakan instans DB pembaca. Klaster bisa memiliki nol, satu, atau banyak instans DB pembaca. Dalam hal ini, kita menghapus instans DB penulis menggunakan perintah `delete-db-instance`. Segera setelah instans DB masuk ke status `deleting`, kita juga dapat menghapus klaster. Untuk contoh ini juga, misalkan klaster tidak berisi data yang layak untuk disimpan. Oleh karena itu, kita tidak membuat snapshot dari volume klaster atau mempertahankan cadangan otomatis.

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only --skip-final-
snapshot
An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-only \
  --query '*[].[DBClusterIdentifier,Status,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]'
[
    [
        "deleteme-writer-only",
        "available",
        [
            [
                "instance-2130",
                true
            ]
        ]
    ]
]

$ aws rds delete-db-instance --db-instance-identifier instance-2130
{
  "DBInstanceIdentifier": "instance-2130",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-only \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-only",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

Menghapus klaster Aurora dengan beberapa instans DB

Jika klaster Anda berisi beberapa instans DB, biasanya ada satu instans penulis dan satu atau beberapa instans pembaca. Instans pembaca membantu dengan ketersediaan tinggi, dengan

bersiaga untuk mengambil alih jika instans penulis mengalami masalah. Anda juga dapat menggunakan instans pembaca untuk memperbesar skala klaster guna menangani beban kerja padat baca tanpa menambahkan overhead ke instans penulis.

Untuk menghapus klaster dengan beberapa instans DB pembaca, hapus instans pembaca terlebih dahulu, kemudian instans penulis. Menghapus instans penulis meninggalkan klaster dan data di tempat. Anda menghapus klaster melalui tindakan terpisah.

- Untuk prosedur penghapusan instans Aurora DB, lihat Menghapus instans dari klaster DB Aurora.
- Untuk prosedur menghapus instans DB penulis dalam klaster Aurora, lihat Menghapus klaster Aurora dengan satu instans DB.
- Untuk prosedur penghapusan klaster Aurora kosong, lihat Menghapus klaster Aurora kosong.

Contoh CLI ini menunjukkan cara menghapus klaster yang berisi instans DB penulis dan satu instans DB pembaca. Output `describe-db-clusters` menunjukkan bahwa `instance-7384` adalah instans penulis dan `instance-1039` adalah instans pembaca. Contoh ini menghapus instans pembaca terlebih dahulu, karena menghapus instans penulis saat instans pembaca masih ada akan menyebabkan operasi failover. Mempromosikan instans pembaca menjadi penulis jika Anda juga berencana untuk menghapus instans tersebut bukanlah hal yang masuk akal. Sekali lagi, misalkan instans `db.t2.small` ini hanya digunakan untuk pengembangan dan pengujian, sehingga operasi hapus melewati snapshot akhir dan tidak mempertahankan cadangan otomatis..

```
$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader --skip-
final-snapshot

An error occurred (InvalidDBClusterStateFault) when calling the DeleteDBCluster
 operation:
  Cluster cannot be deleted, it still contains DB instances in non-deleting state.

$ aws rds describe-db-clusters --db-cluster-identifier deleteme-writer-and-reader --
output text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-writer-and-reader
Instance:       instance-1039  False
Instance:       instance-7384   True

$ aws rds delete-db-instance --db-instance-identifier instance-1039
{
  "DBInstanceIdentifier": "instance-1039",
```

```
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-7384
{
  "DBInstanceIdentifier": "instance-7384",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-writer-and-reader \
  --skip-final-snapshot \
  --delete-automated-backups
{
  "DBClusterIdentifier": "deleteme-writer-and-reader",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

Contoh berikut menunjukkan cara menghapus klaster DB yang berisi instans DB penulis dan beberapa instans DB pembaca. Contoh tersebut menggunakan output ringkas dari perintah `describe-db-clusters` untuk mendapatkan laporan instans penulis dan pembaca. Sekali lagi, kita menghapus semua instans DB pembaca sebelum menghapus instans DB penulis. Urutan penghapusan instans DB pembaca bukanlah hal penting.

Misalkan klaster dengan beberapa instans DB ini berisi data yang layak disimpan. Maka, perintah `delete-db-cluster` dalam contoh ini mencakup parameter `--no-skip-final-snapshot` dan `--final-db-snapshot-identifier` untuk menentukan detail snapshot yang akan dibuat. Ini juga mencakup parameter `--no-delete-automated-backups` untuk mempertahankan cadangan otomatis.

```
$ aws rds describe-db-clusters --db-cluster-identifier deleteme-multiple-readers --
output text \
  --query '*[].["Cluster:",DBClusterIdentifier,DBClusterMembers[*].
["Instance:",DBInstanceIdentifier,IsClusterWriter]]
Cluster:        deleteme-multiple-readers
Instance:       instance-1010   False
Instance:       instance-5410   False
Instance:       instance-9948   False
Instance:       instance-8451   True
```

```
$ aws rds delete-db-instance --db-instance-identifier instance-1010
{
  "DBInstanceIdentifier": "instance-1010",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-5410
{
  "DBInstanceIdentifier": "instance-5410",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-9948
{
  "DBInstanceIdentifier": "instance-9948",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-instance --db-instance-identifier instance-8451
{
  "DBInstanceIdentifier": "instance-8451",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql"
}

$ aws rds delete-db-cluster --db-cluster-identifier deleteme-multiple-readers \
  --no-delete-automated-backups \
  --no-skip-final-snapshot \
  --final-db-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
  "DBClusterIdentifier": "deleteme-multiple-readers",
  "Status": "available",
  "Engine": "aurora-mysql"
}
```

Contoh berikut menunjukkan cara mengonfirmasi bahwa Aurora membuat snapshot yang diminta. Anda dapat meminta detail untuk snapshot tertentu dengan menentukan ID–nya `deleteme-multiple-readers-final-snapshot`. Anda juga bisa mendapatkan laporan tentang semua snapshot untuk klaster yang telah dihapus dengan menentukan ID klaster `deleteme-multiple-readers`. Kedua perintah tersebut menampilkan informasi tentang snapshot yang sama.

```
$ aws rds describe-db-cluster-snapshots \
  --db-cluster-snapshot-identifier deleteme-multiple-readers-final-snapshot
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
...

$ aws rds describe-db-cluster-snapshots --db-cluster-identifier deleteme-multiple-
readers
{
    "DBClusterSnapshots": [
        {
            "AvailabilityZones": [],
            "DBClusterSnapshotIdentifier": "deleteme-multiple-readers-final-snapshot",
            "DBClusterIdentifier": "deleteme-multiple-readers",
            "SnapshotCreateTime": "11T01:40:07.354000+00:00",
            "Engine": "aurora-mysql",
...
```

Perlindungan penghapusan untuk klaster Aurora

Anda tidak dapat menghapus klaster yang perlindungan penghapusannya diaktifkan. Anda dapat menghapus instans DB dalam klaster, tapi bukan klaster itu sendiri. Dengan demikian, volume klaster yang berisi semua data Anda aman dari penghapusan yang tidak disengaja. Aurora memberlakukan perlindungan penghapusan untuk klaster DB, baik Anda mencoba menghapus klaster menggunakan konsol, AWS CLI, atau RDS API.

Perlindungan penghapusan diaktifkan secara default ketika Anda membuat klaster DB produksi menggunakan AWS Management Console. Namun, perlindungan penghapusan dinonaktifkan secara default jika Anda membuat klaster menggunakan AWS CLI atau API. Mengaktifkan atau menonaktifkan perlindungan penghapusan tidak menyebabkan pemadaman. Untuk dapat menghapus klaster, ubah klaster dan nonaktifkan perlindungan penghapusan. Untuk informasi selengkapnya tentang cara mengaktifkan dan menonaktifkan perlindungan penghapusan, lihat Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.

> **Tip**
>
> Meskipun semua instans DB dihapus, Anda dapat mengakses data dengan membuat instans DB baru di klaster.

Menghapus klaster Aurora yang berhenti

Anda tidak dapat menghapus klaster jika berada dalam status `stopped`. Dalam kasus ini, mulailah klaster sebelum menghapusnya. Untuk informasi selengkapnya, lihat Memulai klaster DB Aurora.

Menghapus klaster Aurora MySQL yang merupakan replika baca

Untuk Aurora MySQL, Anda tidak dapat menghapus instans DB dalam klaster DB jika kedua kondisi berikut terpenuhi:

- Klaster DB adalah replika baca dari klaster DB Aurora lainnya.
- Instans DB adalah satu-satunya instans dalam klaster DB.

Untuk menghapus instans DB dalam kasus ini, pertama, promosikan klaster DB agar tidak lagi menjadi replika baca. Setelah promosi selesai, Anda dapat menghapus instans DB akhir dalam klaster DB. Untuk informasi selengkapnya, lihat Mereplikasi klaster DB Amazon Aurora MySQL di seluruh Wilayah AWS.

Snapshot akhir saat menghapus klaster

Sepanjang bagian ini, contoh ini menunjukkan bagaimana Anda dapat memilih apakah akan mengambil snapshot akhir ketika menghapus klaster Aurora. Jika Anda memilih untuk mengambil snapshot akhirh, tetapi nama yang Anda tentukan sama dengan snapshot yang ada, operasi akan berhenti dengan kesalahan. Dalam kasus ini, periksa detail snapshot untuk mengonfirmasi apakah snapshot tersebut mewakili detail Anda saat ini atau apakah snapshot ini snapshot lama. Jika snapshot yang ada tidak memiliki data terbaru yang ingin dipertahankan, ganti nama snapshot dan coba lagi, atau tentukan nama yang berbeda untuk parameter snapshot akhir.

Menghapus instans dari klaster DB Aurora

Anda dapat menghapus instans DB dari klaster DB Aurora sebagai bagian dari proses penghapusan seluruh klaster. Jika klaster Anda berisi sejumlah instans DB, maka penghapusan klaster

mengharuskan penghapusan setiap instans DB. Anda juga dapat menghapus satu atau beberapa instans pembaca dari klaster sekaligus membiarkan klaster berjalan. Anda dapat melakukannya untuk mengurangi kapasitas komputasi dan biaya terkait jika klaster Anda tidak sibuk.

Untuk menghapus instans DB, tentukan nama instans.

Anda dapat menghapus instans DB menggunakan AWS Management Console, AWS CLI, atau RDS API.

> **Note**
>
> Saat Aurora Replica dihapus, titik akhir instansnya akan segera dihilangkan, dan Aurora Replica akan dihilangkan dari titik akhir pembaca. Jika ada pernyataan yang dijalankan dari Aurora Replica yang sedang dihapus, masa tenggangnya adalah tiga menit. Pernyataan yang ada dapat diselesaikan selama masa tenggang. Setelah masa tenggang berakhir, Aurora Replica akan dimatikan dan dihapus.

Untuk klaster DB Aurora, menghapus instans DB tidak selalu menghapus seluruh klaster. Anda dapat menghapus instans DB di klaster Aurora untuk mengurangi kapasitas komputasi dan biaya terkait ketika klaster tidak sibuk. Untuk informasi tentang keadaan khusus bagi klaster Aurora yang memiliki satu instans DB atau nol instans DB, lihat Menghapus klaster Aurora dengan satu instans DB dan Menghapus klaster Aurora kosong.

> **Note**
>
> Anda tidak dapat menghapus klaster DB saat perlindungan penghapusan diaktifkan. Untuk informasi selengkapnya, lihat Perlindungan penghapusan untuk klaster Aurora.
> Anda dapat menonaktifkan perlindungan penghapusan dengan mengubah klaster DB. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora.

Konsol

Untuk menghapus instans DB dalam klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih instans DB yang ingin dihapus.

3. Untuk Tindakan, pilih Hapus.
4. Masukkan **`delete me`** dalam kotak.
5. Pilih Hapus.

AWS CLI

Untuk menghapus instance DB dengan menggunakanAWS CLI, panggil delete-db-instanceperintah dan tentukan `--db-instance-identifier` nilainya.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds delete-db-instance \
    --db-instance-identifier mydbinstance
```

Untuk Windows:

```
aws rds delete-db-instance ^
    --db-instance-identifier mydbinstance
```

RDS API

Untuk menghapus instans DB dengan menggunakan Amazon RDS API, panggil operasi DeleteDBInstance dan tentukan parameter `DBInstanceIdentifier`.

> **Note**
>
> Jika status untuk instans DB adalah `deleting`, nilai sertifikat CA-nya tidak muncul di konsol RDS atau di output untuk perintah AWS CLI atau operasi RDS API. Untuk informasi selengkapnya tentang sertifikat CA, lihat .

Memberi tag pada sumber daya Amazon RDS

Anda dapat menggunakan tag Amazon RDS untuk menambahkan metadata ke sumber daya Amazon RDS Anda. Anda dapat menggunakan tag untuk menambahkan notasi Anda sendiri tentang instans basis data, snapshot, klaster Aurora, dan sebagainya. Tindakan ini dapat membantu Anda mendokumentasikan sumber daya Amazon RDS Anda. Anda juga dapat menggunakan tag dengan prosedur pemeliharaan otomatis.

Khususnya, Anda dapat menggunakan tag ini dengan kebijakan IAM. Anda dapat menggunakannya untuk mengelola akses ke sumber daya RDS dan mengontrol tindakan yang dapat diterapkan pada sumber daya RDS. Anda dapat menggunakan tag ini untuk melacak biaya dengan mengelompokkan pengeluaran untuk sumber daya serupa yang diberi tag.

Anda dapat memberi tag pada sumber daya Amazon RDS berikut:

- Instans DB
- Klaster DB
- Titik akhir cluster DB
- Replika baca
- Snapshot DB
- Snapshot klaster DB
- Instans DB terpesan
- Langganan peristiwa
- Grup opsi DB
- Grup parameter DB
- Grup parameter klaster DB
- Grup subnet DB
- Proksi RDS
- Titik akhir Proksi RDS
- Deployment blue/green
- Integrasi nol-ETL (pratinjau)

> **Note**
>
> Saat ini, Anda tidak dapat menandai RDS Proxies dan RDS Proxy endpoint dengan menggunakan. AWS Management Console

Topik

- Gambaran umum tag sumber daya Amazon RDS
- Menggunakan tag untuk kontrol akses dengan IAM
- Menggunakan tag untuk menghasilkan laporan penagihan mendetail
- Menambahkan, menampilkan daftar, dan menghapus tag
- Menggunakan Editor AWS Tag
- Menyalin tag ke snapshot klaster DB
- Tutorial: Menggunakan tag untuk menentukan klaster DB Aurora yang akan dihentikan.

Gambaran umum tag sumber daya Amazon RDS

Tag Amazon RDS adalah pasangan nama-nilai yang Anda tentukan dan tautkan dengan sumber daya Amazon RDS. Nama ini disebut sebagai kunci. Memberikan nilai untuk kunci bersifat opsional. Anda dapat menggunakan tag untuk memberikan informasi tambahan ke sumber daya Amazon RDS. Anda dapat menggunakan kunci tag, misalnya, untuk menentukan kategori, dan nilai tag mungkin merupakan item dalam kategori tersebut. Misalnya, Anda dapat menentukan kunci tag “project” dan nilai tag “Salix”. Hal ini menunjukkan bahwa sumber daya Amazon RDS ditetapkan ke proyek Salix. Anda juga dapat menggunakan tag untuk menunjukkan bahwa sumber daya Amazon RDS sedang digunakan untuk pengujian atau produksi dengan menggunakan kunci seperti `environment=test` atau `environment=production`. Sebaiknya Anda menggunakan kumpulan kunci tag yang konsisten guna mempermudah palacakan metadata yang terkait dengan sumber daya Amazon RDS.

Selain itu, Anda dapat menggunakan kondisi dalam kebijakan IAM Anda untuk mengontrol akses ke AWS sumber daya berdasarkan tag pada sumber daya tersebut. Anda dapat melakukan ini dengan menggunakan kunci kondisi `aws:ResourceTag/`*`tag-key`* global. Untuk informasi selengkapnya, lihat Mengontrol akses ke AWS sumber daya di Panduan Pengguna AWS Identity and Access Management.

Setiap sumber daya Amazon RDS memiliki serangkaian tag, yang berisi semua tag yang ditetapkan ke sumber daya Amazon RDS tersebut. Rangkaian tag dapat berisi 50 tag atau kosong. Jika Anda

menambahkan tag ke sumber daya RDS dengan kunci yang sama dengan tag sumber daya yang ada, nilai yang baru akan menimpa yang lama.

AWS tidak menerapkan makna semantik apa pun pada tag Anda; tag ditafsirkan secara ketat sebagai string karakter. RDS dapat mengatur tag pada instans DB atau sumber daya RDS lainnya. Pengaturan tag bergantung pada opsi yang Anda gunakan saat membuat sumber daya. Misalnya, Amazon RDS dapat menambahkan tag yang menunjukkan bahwa instans DB digunakan untuk produksi atau pengujian.

- Kunci tag adalah nama wajib tag. Nilai string dapat terdiri dari 1 hingga 128 karakter Unicode dan tidak boleh diawali dengan `aws:` atau `rds:`. String hanya dapat berisi kumpulan huruf Unicode, angka, spasi, '_', '.', ':', '/', '=', '+', '-', '@' (regex Java: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$").

- Nilai tag adalah nilai string opsional dari tag. Nilai string dapat terdiri dari 1 hingga 256 karakter Unicode. String hanya dapat berisi kumpulan huruf Unicode, angka, spasi, '_', '.', ':', '/', '=', '+', '-', '@' (regex Java: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-@]*)$").

  Nilai rangkaian tag tidak harus unik dan bisa nol. Misalnya, Anda dapat menggunakan pasangan kunci-nilai dalam satu rangkaian tag `project=Trinity` dan `cost-center=Trinity`.

Anda dapat menggunakan, API AWS Management Console AWS CLI, atau Amazon RDS untuk menambahkan, membuat daftar, dan menghapus tag di sumber daya Amazon RDS. Saat menggunakan CLI atau API, pastikan untuk menyediakan Amazon Resource Name (ARN) milik sumber daya RDS yang akan ditangani. Untuk informasi selengkapnya tentang cara menyusun ARN, lihat Membuat konsep ARN untuk Amazon RDS.

Tag disimpan di cache untuk diotorisasi. Oleh karena itu, penambahan dan pembaruan tag di sumber daya Amazon RDS dapat memakan waktu beberapa menit sebelum tersedia.

Menggunakan tag untuk kontrol akses dengan IAM

Anda dapat menggunakan tag dengan kebijakan IAM untuk mengelola akses ke sumber daya Amazon RDS. Anda juga dapat menggunakan tag untuk mengontrol tindakan yang dapat diterapkan ke sumber daya Amazon RDS.

Untuk informasi tentang pengelolaan akses ke sumber daya yang diberi tag dengan kebijakan IAM, lihat Manajemen identitas dan akses untuk Amazon Aurora.

Menggunakan tag untuk menghasilkan laporan penagihan mendetail

Anda dapat menggunakan tag ini untuk melacak biaya dengan mengelompokkan pengeluaran untuk sumber daya serupa yang diberi tag.

Gunakan tag untuk mengatur AWS tagihan Anda untuk mencerminkan struktur biaya Anda sendiri. Untuk melakukan ini, daftar untuk mendapatkan Akun AWS tagihan Anda dengan nilai kunci tag disertakan. Kemudian, untuk melihat biaya sumber daya gabungan, atur informasi penagihan Anda sesuai dengan sumber daya Anda dengan nilai kunci tag yang sama. Misalnya, Anda dapat memberi tag beberapa sumber daya dengan nama aplikasi tertentu, kemudian susun informasi penagihan Anda untuk melihat biaya total aplikasi tersebut pada beberapa layanan. Untuk informasi selengkapnya, lihat Menggunakan Tag Alokasi Biaya dalam Panduan Pengguna AWS Billing .

> **Note**
>
> Anda dapat menambahkan tag ke snapshot cluster DB; namun, tagihan Anda tidak akan mencerminkan pengelompokan ini.
> Agar tag alokasi biaya diterapkan ke snapshot cluster DB, tag harus dilampirkan ke cluster DB induk, dan cluster induk harus ada Wilayah AWS sama dengan snapshot. Biaya untuk snapshot yatim piatu digabungkan dalam satu item yang tidak ditandai.

Menambahkan, menampilkan daftar, dan menghapus tag

Prosedur berikut menunjukkan cara melakukan operasi pemberian tag standar pada sumber daya yang terkait dengan instans DB dan klaster Aurora DB.

Konsol

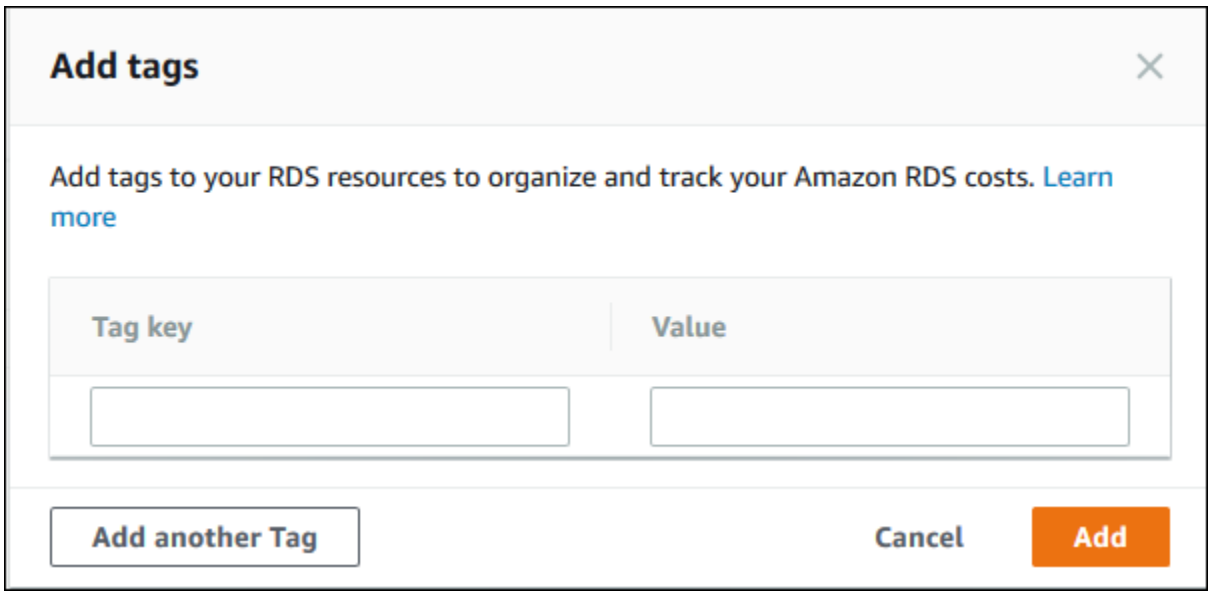
Proses pemberian tag pada sumber daya Amazon RDS untuk semua sumber daya dilakukan secara sama. Prosedur berikut menunjukkan cara memberi tag pada instans DB Amazon RDS.

Untuk menambahkan tag ke instans DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.

> **Note**
>
> Untuk memfilter daftar instans DB dalam panel Basis data, masukkan string teks untuk Filter basis data. Hanya instans DB yang berisi string yang muncul.

3. Pilih nama instans DB yang ingin Anda beri tag untuk menampilkan detailnya.
4. Di bagian detail, gulir ke bawah hingga bagian Tag.
5. Pilih Tambahkan. Jendela Tambahkan tag akan muncul.



6. Masukkan nilai untuk Kunci tag dan nilai.
7. Untuk menambahkan tag lain, Anda dapat memilih Tambahkan Tag lain dan memasukkan nilai untuk Kunci tag dan Nilai-nya.

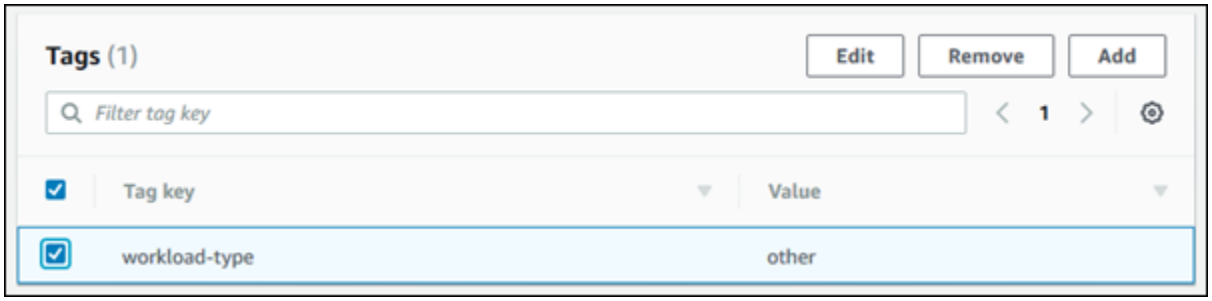
   Ulangi langkah ini seperlunya.
8. Pilih Tambahkan.

Untuk menghapus tag dari instans DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.

> **Note**
>
> Untuk memfilter daftar instans DB dalam panel Basis data, masukkan string teks dalam kotak Filter basis data. Hanya instans DB yang berisi string yang muncul.

3. Pilih nama instans DB untuk menampilkan detailnya.
4. Di bagian detail, gulir ke bawah hingga bagian Tag.
5. Pilih tag yang ingin Anda hapus.



6. Pilih Hapus, lalu pilih Hapus di jendela Hapus tag.

AWS CLI

Anda dapat menambahkan, menampilkan daftar, atau menghapus tag untuk instans DB menggunakan AWS CLI.

- Untuk menambahkan satu atau beberapa tag ke sumber daya Amazon RDS, gunakan AWS CLI perintah `add-tags-to-resource`.
- Untuk membuat daftar tag pada sumber daya Amazon RDS, gunakan AWS CLI perintah `list-tags-for-resource`.
- Untuk menghapus satu atau beberapa tag dari sumber daya Amazon RDS, gunakan AWS CLI perintah `remove-tags-from-resource`.

Untuk mempelajari lebih lanjut tentang cara membuat ARN yang diperlukan, lihat Membuat konsep ARN untuk Amazon RDS.

API RDS

Anda dapat menambahkan, menampilkan daftar, atau menghapus tag untuk instans DB menggunakan API Amazon RDS.

- Untuk menambahkan tag ke sumber daya Amazon RDS, gunakan operasi `AddTagsToResource`.
- Untuk menampilkan daftar tag yang ditetapkan ke sumber daya Amazon RDS, gunakan `ListTagsForResource`.
- Untuk menghapus tag dari sumber daya Amazon RDS, gunakan operasi `RemoveTagsFromResource`.

Untuk mempelajari selengkapnya tentang cara menyusun ARN yang diperlukan, lihat Membuat konsep ARN untuk Amazon RDS.

Ketika menggunakan XML dengan API Amazon RDS, tag menggunakan skema berikut:

```
<Tagging>
    <TagSet>
        <Tag>
            <Key>Project</Key>
            <Value>Trinity</Value>
        </Tag>
        <Tag>
            <Key>User</Key>
            <Value>Jones</Value>
        </Tag>
    </TagSet>
</Tagging>
```

Tabel berikut menyediakan daftar tag XML yang diizinkan beserta karakteristiknya. Nilai untuk Kunci dan Nilai bersifat peka huruf besar/kecil. Misalnya, project=Trinity dan PROJECT=Trinity adalah dua tag yang berbeda.

Elemen tag	Deskripsi
TagSet	Rangkaian tag adalah wadah untuk semua tag yang ditetapkan ke sumber daya Amazon RDS. Hanya ada satu rangkaian tag per sumber daya. Anda bekerja dengan TagSet hanya melalui Amazon RDS API.
Tag	Tag adalah pasangan kunci-nilai yang ditentukan pengguna. Satu rangkaian tag bisa berisi 1 hingga 50 tag.
Kunci	Kunci adalah nama wajib tag. Nilai string dapat terdiri dari 1 hingga 128 karakter Unicode dan tidak boleh diawali dengan `aws:` atau `rds:`. String

Elemen tag	Deskripsi
	hanya dapat berisi kumpulan huruf Unicode, angka, spasi, '_', '.', '/', '=', '+', '-' (regex Java: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$"). Kunci dalam rangkaian tag harus unik. Misalnya, Anda tidak dapat memiliki pasangan kunci dalam satu rangkaian tag dengan kunci yang sama tetapi dengan nilai yang berbeda, seperti project/Trinity dan project/Xanadu.
Nilai	Nilai adalah nilai opsional tag. Nilai string dapat terdiri dari 1 hingga 256 karakter Unicode dan tidak boleh diawali dengan `aws:` atau `rds:`. String hanya dapat berisi kumpulan huruf Unicode, angka, spasi, '_', '.', '/', '=', '+', '-' (regex Java: "^([\\p{L}\\p{Z}\\p{N}_.:/=+\\-]*)$"). Nilai rangkaian tag tidak harus unik dan bisa nol. Misalnya, Anda dapat menggunakan pasangan kunci-nilai dalam satu rangkaian tag project/Trinity dan cost-center/Trinity.

Menggunakan Editor AWS Tag

Anda dapat menelusuri dan mengedit tag pada sumber daya RDS Anda AWS Management Console dengan menggunakan editor AWS Tag. Untuk informasi selengkapnya, lihat Editor Tag dalam Panduan Pengguna AWS .

Menyalin tag ke snapshot klaster DB

Saat membuat atau memulihkan instans DB, Anda dapat menentukan bahwa tag dari klaster DB disalin ke snapshot klaster DB. Penyalinan tag memastikan bahwa metadata untuk snapshot DB cocok dengan metadata klaster DB sumber. Ini juga memastikan bahwa kebijakan akses apa pun untuk snapshot DB juga cocok dengan kebijakan akses klaster DB sumber. Tag tidak disalin secara default.

Anda dapat menentukan bahwa tag disalin ke snapshot DB untuk tindakan berikut:

- Membuat klaster DB.
- Memulihkan klaster DB.
- Membuat replika baca.

- Menyalin snapshot klaster DB.

> **Note**
>
> Dalam beberapa kasus, Anda mungkin menyertakan nilai untuk `--tags` parameter create-db-snapshot AWS CLI perintah. Atau Anda mungkin perlu memasukkan setidaknya satu tag ke operasi CreateDBSnapshot API. Dalam kasus ini, RDS tidak menyalin tag dari instans DB sumber ke snapshot DB baru. Fungsi ini berlaku meskipun opsi `--copy-tags-to-snapshot` (`CopyTagsToSnapshot`) di instans DB sumber diaktifkan.
> Jika menggunakan pendekatan ini, Anda dapat membuat salinan instans DB dari snapshot DB. Pendekatan ini menghindari penambahan tag yang tidak berlaku untuk instans DB baru. Anda membuat snapshot DB menggunakan AWS CLI `create-db-snapshot` perintah (atau operasi `CreateDBSnapshot` RDS API). Setelah membuat snapshot DB, Anda dapat menambahkan tag seperti yang dijelaskan nanti dalam topik ini.

Tutorial: Menggunakan tag untuk menentukan klaster DB Aurora yang akan dihentikan.

Misalkan Anda membuat sejumlah klaster DB Aurora dalam lingkungan pengembangan atau pengujian. Anda perlu mempertahankan semua klaster ini selama beberapa hari. Beberapa klaster menjalankan pengujian di malam hari. Klaster lainnya dapat dihentikan di malam hari dan dimulai lagi keesokan harinya. Contoh berikut menunjukkan cara menetapkan tag untuk klaster yang cocok untuk dihentikan di malam hari. Kemudian contoh berikut menunjukkan bagaimana skrip dapat mendeteksi klaster yang memiliki tag dan kemudian menghentikan klaster tersebut. Dalam contoh ini, porsi nilai dari pasangan kunci-nilai tidaklah penting. Keberadaan tag `stoppable` menandakan bahwa klaster memiliki properti yang ditetapkan pengguna ini.

Untuk menentukan klaster DB Aurora yang akan dihentikan

1. Tentukan ARN klaster yang ingin ditentukan sebagai dapat dihentikan.

   Perintah dan API untuk pemberian tag berfungsi dengan ARN. Dengan begitu, mereka dapat bekerja dengan lancar di seluruh AWS Wilayah, AWS akun, dan berbagai jenis sumber daya yang mungkin memiliki nama pendek yang identik. Anda dapat menentukan ARN, bukan ID klaster, dalam perintah CLI yang beroperasi pada klaster. Gantikan nama cluster Anda sendiri *`dev-test-cluster`*. Dalam perintah berikutnya yang menggunakan parameter ARN, ganti

ARN klaster Anda sendiri. ARN menyertakan ID AWS akun Anda sendiri dan nama AWS Wilayah tempat cluster Anda berada.

```
$ aws rds describe-db-clusters --db-cluster-identifier dev-test-cluster \
  --query "*[].{DBClusterArn:DBClusterArn}" --output text
arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
```

2. Tambahkan tag `stoppable` ke klaster ini.

   Pilih nama untuk tag ini. Dengan pendekatan ini, berarti Anda tidak perlu merancang konvensi penamaan yang mengenkode semua informasi yang relevan dalam nama. Dalam konvensi tersebut, Anda dapat mengenkode informasi dalam nama instans DB atau nama-nama sumber daya lainnya. Karena memperlakukan tag sebagai atribut yang ada atau tidak ada, contoh ini menghilangkan bagian `Value=` dari parameter `--tags`.

```
$ aws rds add-tags-to-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster \
  --tags Key=stoppable
```

3. Konfirmasi bahwa tag tersebut ada dalam klaster.

   Perintah ini mengambil informasi tag untuk klaster dalam format JSON dan dalam bentuk teks biasa yang dipisahkan tab.

```
$ aws rds list-tags-for-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
{
    "TagList": [
        {
            "Key": "stoppable",
            "Value": ""

        }
    ]
}
$ aws rds list-tags-for-resource \
  --resource-name arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster --output
 text
TAGLIST stoppable
```

4. Untuk menghentikan semua klaster yang ditetapkan sebagai `stoppable`, siapkan daftar semua klaster Anda. Lihat daftar dan periksa apakah setiap klaster diberi tag dengan atribut yang relevan.

   Contoh Linux ini menggunakan skrip shell untuk menyimpan daftar ARN klaster ke file sementara lalu menjalankan perintah CLI untuk setiap klaster.

```
$ aws rds describe-db-clusters --query "*[].[DBClusterArn]" --output text >/tmp/
cluster_arns.lst
$ for arn in $(cat /tmp/cluster_arns.lst)
do
  match="$(aws rds list-tags-for-resource --resource-name $arn --output text | grep
 'TAGLIST\tstoppable')"
  if [[ ! -z "$match" ]]
  then
      echo "Cluster $arn is tagged as stoppable. Stopping it now."
# Note that you can specify the full ARN value as the parameter instead of the
 short ID 'dev-test-cluster'.
      aws rds stop-db-cluster --db-cluster-identifier $arn
  fi
done

Cluster arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster is tagged as
 stoppable. Stopping it now.
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "us-east-1e",
            "us-east-1c",
            "us-east-1d"
        ],
        "BackupRetentionPeriod": 1,
        "DBClusterIdentifier": "dev-test-cluster",
        ...
```

Anda dapat menjalankan skrip seperti ini pada akhir setiap hari untuk memastikan bahwa klaster yang tidak penting dihentikan. Anda juga dapat menjadwalkan pekerjaan menggunakan utilitas seperti `cron` untuk melakukan pemeriksaan setiap malam. Misalnya, Anda mungkin melakukan ini jika beberapa klaster dibiarkan berjalan secara tidak sengaja. Di sini, Anda dapat menyesuaikan perintah yang mempersiapkan daftar klaster DB yang akan diperiksa.

Perintah berikut menghasilkan daftar klaster, tetapi hanya yang berada dalam status `available`. Skrip ini dapat mengabaikan klaster yang sudah dihentikan karena klaster tersebut akan memiliki nilai status yang berbeda seperti `stopped` atau `stopping`.

```
$ aws rds describe-db-clusters \
  --query '*[].{DBClusterArn:DBClusterArn,Status:Status}|[?Status == `available`]|[].
{DBClusterArn:DBClusterArn}' \
  --output text
arn:aws:rds:us-east-1:123456789:cluster:cluster-2447
arn:aws:rds:us-east-1:123456789:cluster:cluster-3395
arn:aws:rds:us-east-1:123456789:cluster:dev-test-cluster
arn:aws:rds:us-east-1:123456789:cluster:pg2-cluster
```

> **Tip**
>
> Anda dapat menggunakan penetapan tag dan pencarian klaster dengan tag tersebut untuk mengurangi biaya dengan cara lain. Misalnya, ikuti skenario ini dengan klaster DB Aurora yang digunakan untuk pengembangan dan pengujian. Di sini, Anda dapat menetapkan beberapa klaster untuk dihapus di akhir setiap hari, atau hanya instans DB pembaca saja yang dihapus. Atau Anda dapat menetapkan beberapa klaster agar instans DB-nya diubah menjadi kelas instans DB kecil selama waktu penggunaan rendah yang diharapkan.

Bekerja dengan Amazon Resource Name (ARN) di Amazon RDS

Sumber daya yang dibuat di Amazon Web Services masing-masing diidentifikasi secara unik dengan Amazon Resource Name (ARN). Untuk operasi Amazon RDS tertentu, Anda harus mengidentifikasi sumber daya Amazon RDS secara unik dengan menentukan ARN-nya. Misalnya, saat membuat replika baca instans DB RDS, Anda harus menyediakan ARN untuk instans DB sumber.

Membuat konsep ARN untuk Amazon RDS

Sumber daya yang dibuat di Amazon Web Services masing-masing diidentifikasi secara unik dengan Amazon Resource Name (ARN). Anda dapat membuat konsep ARN untuk sumber daya Amazon RDS menggunakan sintaks berikut.

`arn:aws:rds:<region>:<account number>:<resourcetype>:<name>`

Nama Wilayah	Wilayah	Titik Akhir	Protokol
AS Timur (Ohio)	us–east-2	rds.us-east-2.amazonaws.com	HTTPS
		rds-fips.us-east-2.api.aws	HTTPS
		rds.us-east-2.api.aws	HTTPS
		rds-fips.us-east-2.amazonaws.com	HTTPS
AS Timur (Virginia Utara)	us-east-1	rds.us-east-1.amazonaws.com	HTTPS
		rds-fips.us-east-1.api.aws	HTTPS
		rds-fips.us-east-1.amazonaws.com	HTTPS
		rds.us-east-1.api.aws	HTTPS
AS Barat (California Utara)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS
		rds.us-west-1.api.aws	HTTPS
		rds-fips.us-west-1.amazonaws.com	HTTPS
		rds-fips.us-west-1.api.aws	HTTPS

Nama Wilayah	Wilayah	Titik Akhir	Protokol
AS Barat (Oregon)	us-west-2	rds.us-west-2.amazonaws.com rds-fips.us-west-2.amazonaws.com rds.us-west-2.api.aws rds-fips.us-west-2.api.aws	HTTPS HTTPS HTTPS HTTPS
Afrika (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com rds.af-south-1.api.aws	HTTPS HTTPS
Asia Pasifik (Hong Kong)	ap-east-1	rds.ap-east-1.amazonaws.com rds.ap-east-1.api.aws	HTTPS HTTPS
Asia Pasifik (Hyderabad)	ap-south-2	rds.ap-south-2.amazonaws.com rds.ap-south-2.api.aws	HTTPS HTTPS
Asia Pasifik (Jakarta)	ap-southeast-3	rds.ap-southeast-3.amazonaws.com rds.ap-southeast-3.api.aws	HTTPS HTTPS
Asia Pasifik (Melbourne)	ap-southeast-4	rds.ap-southeast-4.amazonaws.com rds.ap-southeast-4.api.aws	HTTPS HTTPS
Asia Pasifik (Mumbai)	ap-south-1	rds.ap-south-1.amazonaws.com rds.ap-south-1.api.aws	HTTPS HTTPS

Nama Wilayah	Wilayah	Titik Akhir	Protokol
Asia Pasifik (Osaka)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS
		rds.ap-northeast-3.api.aws	HTTPS
Asia Pasifik (Seoul)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS
		rds.ap-northeast-2.api.aws	HTTPS
Asia Pasifik (Singapura)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS
		rds.ap-southeast-1.api.aws	HTTPS
Asia Pasifik (Sydney)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS
		rds.ap-southeast-2.api.aws	HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS
		rds.ap-northeast-1.api.aws	HTTPS
Kanada (Pusat)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS
		rds.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.api.aws	HTTPS
		rds-fips.ca-central-1.amazonaws.com	HTTPS
Kanada Barat (Calgary)	ca-west-1	rds.ca-west-1.amazonaws.com	HTTPS
		rds-fips.ca-west-1.amazonaws.com	HTTPS
Eropa (Frankfurt)	eu-central-1	rds.eu-central-1.amazonaws.com	HTTPS
		rds.eu-central-1.api.aws	HTTPS

Nama Wilayah	Wilayah	Titik Akhir	Protokol
Eropa (Irlandia)	eu-west-1	rds.eu-west-1.amazonaws.com	HTTPS
		rds.eu-west-1.api.aws	HTTPS
Eropa (London)	eu-west-2	rds.eu-west-2.amazonaws.com	HTTPS
		rds.eu-west-2.api.aws	HTTPS
Eropa (Milan)	eu-south-1	rds.eu-south-1.amazonaws.com	HTTPS
		rds.eu-south-1.api.aws	HTTPS
Eropa (Paris)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS
		rds.eu-west-3.api.aws	HTTPS
Eropa (Spanyol)	eu-south-2	rds.eu-south-2.amazonaws.com	HTTPS
		rds.eu-south-2.api.aws	HTTPS
Eropa (Stockholm)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS
		rds.eu-north-1.api.aws	HTTPS
Eropa (Zürich)	eu-central-2	rds.eu-central-2.amazonaws.com	HTTPS
		rds.eu-central-2.api.aws	HTTPS
Israel (Tel Aviv)	il-central-1	rds.il-central-1.amazonaws.com	HTTPS
		rds.il-central-1.api.aws	HTTPS
Timur Tengah (Bahrain)	me-south-1	rds.me-south-1.amazonaws.com	HTTPS
		rds.me-south-1.api.aws	HTTPS

Nama Wilayah	Wilayah	Titik Akhir	Protokol
Timur Tengah (UAE)	me-central-1	rds.me-central-1.amazonaws.com rds.me-central-1.api.aws	HTTPS HTTPS
Amerika Selatan (Sao Paulo)	sa-east-1	rds.sa-east-1.amazonaws.com rds.sa-east-1.api.aws	HTTPS HTTPS
AWS GovCloud (AS-Timur)	us-gov-east-1	rds.us-gov-east-1.amazonaws.com rds.us-gov-east-1.api.aws	HTTPS HTTPS
AWS GovCloud (AS-Barat)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com rds.us-gov-west-1.api.aws	HTTPS HTTPS

Tabel berikut menunjukkan format yang harus Anda gunakan saat membuat konsep ARN untuk jenis sumber daya Amazon RDS tertentu.

Jenis sumber daya	Format ARN
Instans DB	arn:aws:rds:*<region>*:*<account>* :db:*<name>* Misalnya: `arn:aws:rds: us-east-2 :123456789012 :db:my-mysql-instance-1`
Klaster DB	arn:aws:rds:*<region>*:*<account>* :cluster:*<name>* Misalnya:

Jenis sumber daya	Format ARN
	`arn:aws:rds:` *us-east-2* `:`*123456789012* `:cluster:` *my-aurora-cluster-1*
Langganan peristiwa	arn:aws:rds:*<region>*:*<account>* :es:*<name>* Misalnya: `arn:aws:rds:` *us-east-2* `:`*123456789012* `:es:`*my-subscription*
Grup parameter DB	arn:aws:rds:*<region>*:*<account>* :pg:*<name>* Misalnya: `arn:aws:rds:` *us-east-2* `:`*123456789012* `:pg:`*my-param-enable-logs*
Grup parameter klaster DB	arn:aws:rds:*<region>*:*<account>* :cluster-pg:*<name>* Misalnya: `arn:aws:rds:` *us-east-2* `:`*123456789012* `:cluster-pg:` *my-cluster-param-timezone*
Instans DB yang dicadangkan	arn:aws:rds:*<region>*:*<account>* :ri:*<name>* Misalnya: `arn:aws:rds:` *us-east-2* `:`*123456789012* `:ri:`*my-reserved-postgresql*
Grup keamanan DB	arn:aws:rds:*<region>*:*<account>* :secgrp:*<name>* Misalnya: `arn:aws:rds:` *us-east-2* `:`*123456789012* `:secgrp:`*my-public*

Jenis sumber daya	Format ARN
Snapshot DB otomatis	arn:aws:rds:*<region>*:*<account>* :snapshot:rds:*<name>* Misalnya: `arn:aws:rds: us-east-2 :123456789012 :snapshot:rds: my-mysql-db-2019-07-22-07-23`
Snapshot klaster DB otomatis	arn:aws:rds:*<region>*:*<account>* :cluster-snapshot:rds:*<name>* Misalnya: `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot:rds: my-aurora-cluster-2019-07-22-16-16`
Snapshot DB manual	arn:aws:rds:*<region>*:*<account>* :snapshot:*<name>* Misalnya: `arn:aws:rds: us-east-2 :123456789012 :snapshot: my-mysql-db-snap`
Snapshot klaster DB manual	arn:aws:rds:*<region>*:*<account>* :cluster-snapshot:*<name>* Misalnya: `arn:aws:rds: us-east-2 :123456789012 :cluster-snapshot: my-aurora-cluster-snap`
Grup subnet DB	arn:aws:rds:*<region>*:*<account>* :subgrp:*<name>* Misalnya: `arn:aws:rds: us-east-2 :123456789012 :subgrp:my-subnet-10`

Mendapatkan ARN yang sudah ada

Anda bisa mendapatkan ARN dari sumber daya RDS dengan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau RDS API.

Konsol

Untuk mendapatkan ARN dari AWS Management Console, navigasikan ke sumber daya yang Anda inginkan untuk ARN, dan lihat detail untuk sumber daya itu.

Misalnya, Anda bisa mendapatkan ARN untuk klaster DB dari tab Konfigurasi detail klaster DB.

AWS CLI

Untuk mendapatkan ARN dari sumber daya RDS tertentu, Anda menggunakan `describe` perintah untuk sumber daya itu. AWS CLI Tabel berikut menunjukkan setiap AWS CLI perintah, dan properti ARN yang digunakan dengan perintah untuk mendapatkan ARN.

AWS CLI perintah	Properti ARN
describe-event-subscriptions	EventSubscriptionArn
describe-certificates	CertificateArn
describe-db-parameter-groups	DB ParameterGroupArn
describe-db-cluster-parameter-kelompok	DB ClusterParameterGroupArn
describe-db-instances	DB InstanceArn
describe-db-security-groups	DB SecurityGroupArn
describe-db-snapshots	DB SnapshotArn
describe-events	SourceArn
describe-reserved-db-instances	ReservedDB InstanceArn
describe-db-subnet-groups	DB SubnetGroupArn

AWS CLI perintah	Properti ARN
describe-db-clusters	DB ClusterArn
describe-db-cluster-snapshots	DB ClusterSnapshotArn

Misalnya, AWS CLI perintah berikut mendapatkan ARN untuk instance DB.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-instances \
--db-instance-identifier DBInstanceIdentifier \
--region us-west-2 \
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

Untuk Windows:

```
aws rds describe-db-instances ^
--db-instance-identifier DBInstanceIdentifier ^
--region us-west-2 ^
--query "*[].{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceArn:DBInstanceArn}"
```

Output dari perintah tersebut adalah sebagai berikut:

```
[
    {
        "DBInstanceArn": "arn:aws:rds:us-west-2:account_id:db:instance_id",
        "DBInstanceIdentifier": "instance_id"
    }
]
```

RDS API

Untuk mendapatkan ARN untuk sumber daya RDS tertentu, Anda dapat memanggil operasi API RDS berikut dan menggunakan properti ARN yang ditunjukkan sebagai berikut.

Operasi API RDS	Properti ARN
DescribeEventSubscriptions	EventSubscriptionArn
DescribeCertificates	CertificateArn
DijelaskanB ParameterGroups	DB ParameterGroupArn
DijelaskanB ClusterParameterGroups	DB ClusterParameterGroupArn
DescribeDBInstances	DB InstanceArn
DijelaskanB SecurityGroups	DB SecurityGroupArn
DescribeDBSnapshots	DB SnapshotArn
DescribeEvents	SourceArn
DescribeReservedDBInstances	ReservedDB InstanceArn
DijelaskanB SubnetGroups	DB SubnetGroupArn
DescribeDBClusters	DB ClusterArn
DijelaskanB ClusterSnapshots	DB ClusterSnapshotArn

Pembaruan Amazon Aurora

Amazon Aurora merilis pembaruan secara berkala. Pembaruan diterapkan pada klaster DB Amazon Aurora selama periode pemeliharaan sistem. Waktu ketika pembaruan diterapkan akan bergantung pada wilayah dan pengaturan periode pemeliharaan untuk klaster DB, dan juga jenis pembaruannya. Pembaruan memerlukan pengaktifan ulang basis data, sehingga Anda biasanya mengalami waktu henti selama 20–30 detik. Setelah waktu henti ini, Anda dapat lanjut menggunakan klaster DB Anda. Anda dapat melihat atau mengubah pengaturan periode pemeliharaan dari AWS Management Console.

> **Note**
>
> Waktu yang diperlukan untuk melakukan boot ulang instans DB Anda bergantung pada proses pemulihan crash, aktivitas basis data pada saat boot ulang, dan perilaku mesin DB spesifik Anda. Untuk mempersingkat waktu boot ulang, sebaiknya kurangi aktivitas basis data sebanyak mungkin selama proses boot ulang. Pengurangan aktivitas basis data akan menurunkan aktivitas rollback untuk transaksi saat bergerak.

Untuk informasi tentang pembaruan sistem operasi untuk Amazon Aurora, lihat Bekerja dengan pembaruan sistem operasi.

Beberapa pembaruan dikhususkan untuk mesin basis data yang didukung Aurora. Untuk informasi selengkapnya tentang pembaruan mesin basis data, lihat tabel berikut.

Mesin basis data	Pembaruan
Amazon Aurora MySQL	Lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL
Amazon Aurora PostgreSQL	Lihat Pembaruan Amazon Aurora PostgreSQL

Mengidentifikasi versi Amazon Aurora Anda

Amazon Aurora menyertakan fitur tertentu yang bersifat umum untuk Aurora dan tersedia untuk semua klaster DB Aurora. Aurora mencakup fitur lain yang dikhususkan untuk mesin basis

data tertentu yang didukung Aurora. Fitur-fitur ini tersedia hanya untuk klaster DB Aurora yang menggunakan mesin basis data tersebut, seperti Aurora PostgreSQL.

Instans DB Aurora memberikan dua nomor versi, yaitu nomor versi Aurora dan nomor versi mesin basis data Aurora. Nomor versi Aurora menggunakan format berikut.

```
<major version>.<minor version>.<patch version>
```

Untuk mendapatkan nomor versi Aurora dari instans DB Aurora menggunakan mesin basis data tertentu, gunakan salah satu kueri berikut.

Mesin basis data	Kueri
Amazon Aurora MySQL	`SELECT AURORA_VERSION();` `SHOW @@aurora_version;`
Amazon Aurora PostgreSQL	`SELECT AURORA_VERSION();`

Menggunakan Dukungan Diperpanjang Amazon RDS

Dengan Dukungan Diperpanjang Amazon RDS, Anda dapat terus menjalankan basis data Anda pada suatu versi mesin utama melewati akhir tanggal dukungan standar Aurora dengan biaya tambahan. Pada akhir tanggal dukungan standar Aurora, Aurora secara otomatis mendaftarkan database Anda di RDS Extended Support. Pendaftaran otomatis ke RDS Extended Support tidak mengubah mesin database dan tidak memengaruhi waktu aktif atau kinerja instans DB Anda.

Penawaran berbayar ini memberi Anda lebih banyak waktu untuk meningkatkan ke versi mesin utama yang didukung. Selama RDS Extended Support, Amazon RDS akan memasok patch untuk CVE Kritis dan Tinggi seperti yang didefinisikan oleh peringkat tingkat keparahan CVSS National Vulnerability Database (NVD). Lihat informasi yang lebih lengkap di Metrik Kerentanan.

Anda juga dapat membuat basis data baru dengan versi mesin utama yang telah mencapai akhir tanggal dukungan standar Aurora. Aurora secara otomatis mendaftarkan database baru ini di RDS Extended Support dan menagih Anda untuk penawaran ini.

Untuk melakukan ini, gunakan AWS CLI atau API RDS. Dalam AWS CLI, `open-source-rds-extended-support-disabled` tentukan `--engine-lifecycle-support` opsi. Di RDS API, tentukan `open-source-rds-extended-support-disabled` `LifeCycleSupport` parameternya.

Misalnya, akhir tanggal dukungan standar Aurora untuk Aurora MySQL versi 2 adalah 31 Oktober 2024. Namun, Anda belum siap untuk meng-upgrade secara manual ke Aurora MySQL versi 3 sebelum tanggal tersebut. Pada tanggal 31 Oktober 2024, Amazon Aurora secara otomatis mendaftarkan klaster Anda di RDS Extended Support, dan Anda dapat terus menjalankan Aurora MySQL versi 2. Mulai 1 Desember 2024, Amazon Aurora secara otomatis menagih Anda untuk RDS Extended Support.

RDS Extended Support tersedia hingga 3 tahun setelah akhir dari tanggal dukungan standar untuk versi mesin utama (3 tahun dan 4 bulan untuk Aurora MySQL versi 2). Setelah waktu ini, jika Anda belum memutakhirkan versi mesin utama Anda ke versi yang didukung, Aurora akan secara otomatis meningkatkan versi mesin utama Anda. Kami menyarankan agar Anda meningkatkan ke versi mesin utama sesegera mungkin.

Topik

- Biaya Amazon RDS Extended Support

- Versi dengan Amazon RDS Extended Support
- Aurora DB atau cluster global dengan Amazon RDS Extended Support
- Melihat pendaftaran klaster global di Amazon RDS Extended Support
- Aurora DB atau cluster global dengan Amazon RDS Extended Support

Biaya Amazon RDS Extended Support

Biaya tambahan untuk RDS Extended Support secara otomatis berhenti segera setelah Anda meningkatkan ke versi engine yang tercakup dalam dukungan standar, atau Anda menghapus database yang menjalankan versi utama melewati akhir dari tanggal dukungan standar. Namun, pengisian daya akan dimulai kembali jika versi mesin target Anda memasuki RDS Extended Support di masa mendatang.

Misalnya, Aurora PostgreSQL 11 memasuki Extended Support pada 1 Maret 2024, tetapi biaya tidak dimulai hingga 1 April 2024. Namun, jika Anda terus menjalankan Aurora PostgreSQL 12 pada instans DB ini melewati akhir RDS tanggal dukungan standar 28 Februari 2025, maka database Anda akan kembali dikenakan biaya RDS Extended Support mulai 1 Maret 2025.

Lihat informasi yang lebih lengkap di Struktur harga Amazon Aurora.

Versi dengan Amazon RDS Extended Support

RDS Extended Support tersedia untuk Aurora MySQL versi 2 dan 3, dan untuk Aurora PostgreSQL versi 11 dan lebih tinggi. Untuk informasi selengkapnya, lihat Versi mayor Amazon Aurora.

RDS Extended Support hanya tersedia pada versi minor tertentu. Untuk informasi selengkapnya, lihat Versi minor Amazon Aurora.

RDS Extended Support hanya tersedia diAurora Serverless v2. Itu tidak tersedia diAurora Serverless v1.

Aurora DB atau cluster global dengan Amazon RDS Extended Support

Saat Anda Aurora DB atau cluster global, pilih Aktifkan RDS Extended Support di konsol, atau gunakan opsi Extended Support AWS CLI di atau parameter di RDS API.

> **Note**
>
> Jika Anda tidak menentukan pengaturan RDS Extended Support, RDS Extended Support. Perilaku default ini menjaga ketersediaan database Anda melewati akhir Aurora dari tanggal dukungan standar.

Topik

- Pertimbangan untuk RDS Extended Support
- Aurora DB atau cluster global dengan RDS Extended Support

Pertimbangan untuk RDS Extended Support

Sebelum Aurora DB atau cluster global, pertimbangkan item berikut:

- Setelah tanggal dukungan standar Aurora berakhir, Anda dapat mencegah pembuatan cluster global baru dan menghindari biaya RDS Extended Support. Untuk melakukan ini, gunakan AWS CLI atau RDS API. Dalam AWS CLI, `open-source-rds-extended-support-disabled` tentukan `--engine-lifecycle-support` opsi. Di RDS API, tentukan `open-source-rds-extended-support-disabled` `LifeCycleSupport` parameternya. Jika Anda menentukan `open-source-rds-extended-support-disabled` dan akhir Aurora dari tanggal dukungan standar telah berlalu, Aurora DB atau cluster global akan selalu gagal.
- RDS Extended Support diatur pada tingkat cluster. Anggota cluster akan selalu memiliki pengaturan yang sama untuk RDS Extended Support di konsol RDS, `--engine-lifecycle-support` di AWS CLI, dan `EngineLifecycleSupport` di RDS API.

Untuk informasi selengkapnya, lihat Versi Amazon Aurora.

Aurora DB atau cluster global dengan RDS Extended Support

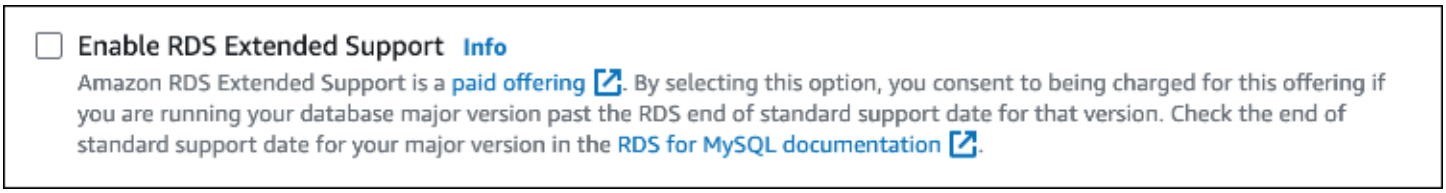
Anda dapat Aurora DB atau cluster global dengan versi RDS Extended Support AWS Management Console menggunakan AWS CLI,, atau RDS API.

> **Note**
>
> AWS CLI `--engine-lifecycle-support`Opsi dan `EngineLifeCycle` parameter RDS API saat ini hanya tersedia untuk Aurora PostgreSQL. Mereka akan tersedia untuk Aurora MySQL lebih dekat ke akhir RDS dari tanggal dukungan standar.

Konsol

Saat Anda membuat cluster Aurora DB atau cluster global instans DB atau cluster , di bagian opsi Engine, pilih Enable RDS Extended Support.

Gambar berikut menunjukkan pengaturan Enable RDS Extended Support:



AWS CLI

Saat Anda menggunakan AWS CLI perintah create-db-clusteror or create-global-clustercreate-db-instanceor create-db-clusterhttps://docs.aws.amazon.com/cli/latest/reference/rds/create-db-cluster.html , pilih RDS Extended Support dengan menentukan `open-source-rds-extended-support` opsi tersebut. `--engine-lifecycle-support` Secara default, opsi ini diatur keo`pen-source-rds-extended-support`.

Untuk mencegah pembuatan Aurora DB baru atau cluster global setelah Aurora berakhir dari tanggal dukungan standar, tentukan opsi. `open-source-rds-extended-support-disabled` `--engine-lifecycle-support` Dengan demikian, Anda akan menghindari biaya RDS Extended Support terkait.

API RDS

Saat Anda menggunakan operasi CreatedBCluster atau CreateGlobalClusterCreateDBInstance atau CreateDBCluster (Multi-AZ DB cluster ke. `EngineLifecycleSupport` `open-source-rds-extended-support` Secara default, parameter ini diatur ke`open-source-rds-extended-support`.

Untuk mencegah pembuatan Aurora DB baru atau cluster global setelah Aurora berakhir dari tanggal dukungan standar, tentukan parameternya. `open-source-rds-extended-support-disabled`

`EngineLifecycleSupport` Dengan demikian, Anda akan menghindari biaya RDS Extended Support terkait.

Untuk informasi selengkapnya, lihat topik berikut:

- Untuk membuat cluster Aurora DB, ikuti instruksi untuk mesin DB Anda. Membuat klaster DB Amazon Aurora
- Untuk membuat cluster global, ikuti petunjuk untuk mesin DB Anda diMembuat basis data global Amazon Aurora.

Melihat pendaftaran klaster global di Amazon RDS Extended Support

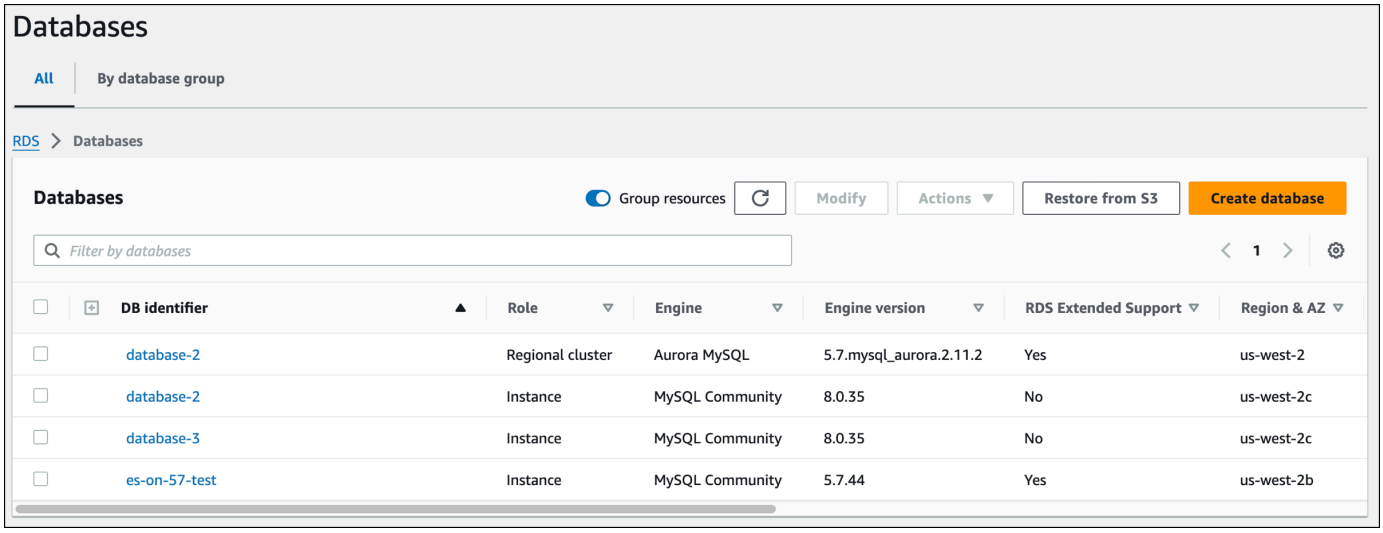
Anda dapat melihat pendaftaran global di RDS Extended Support menggunakan file. AWS Management Console

Konsol

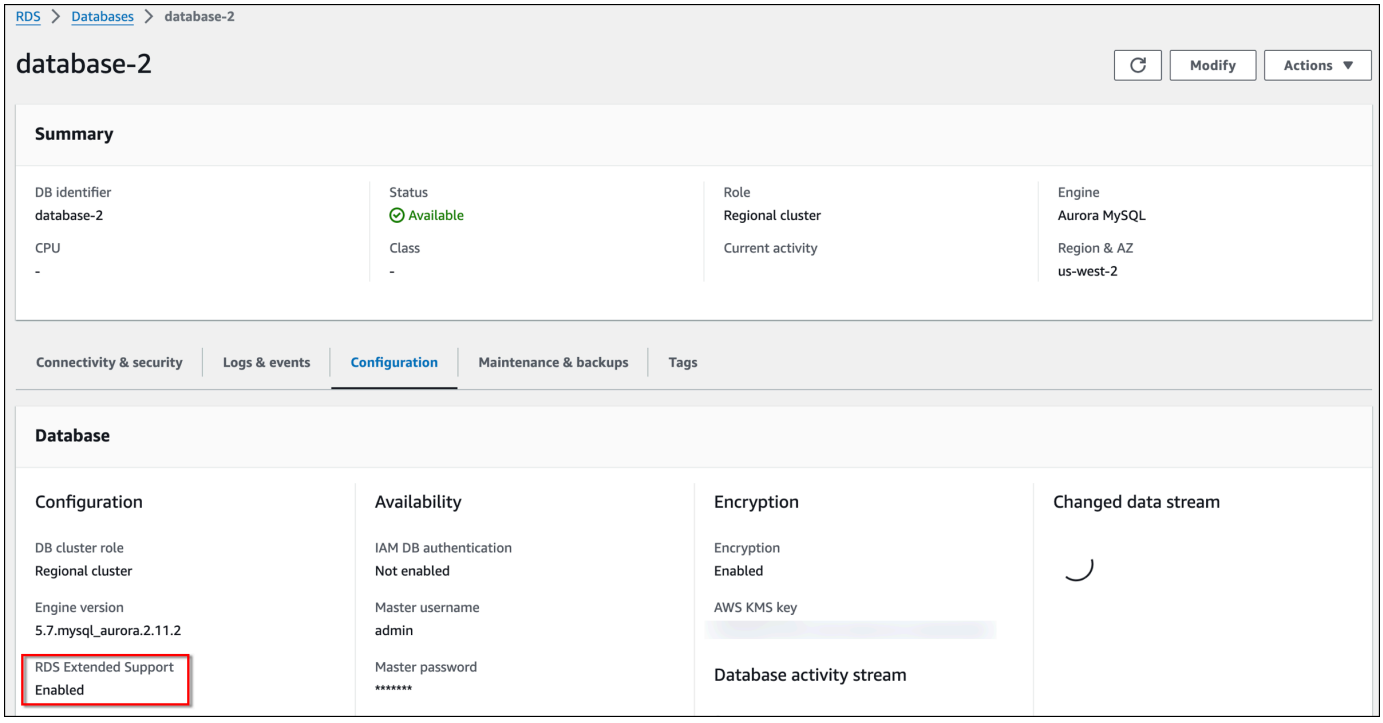
Untuk melihat pendaftaran di RDS Extended Support

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data. Nilai di bawah RDS Extended Support menunjukkan jika Aurora DB atau cluster global terdaftar di RDS Extended Support. Jika tidak ada nilai yang muncul, maka RDS Extended Support tidak tersedia untuk database Anda.

> **Tip**
>
> Jika kolom RDS Extended Support tidak muncul, pilih ikon Preferensi, lalu aktifkan RDS Extended Support.



3. Anda juga dapat melihat pendaftaran pada tab Konfigurasi untuk setiap database. Pilih database di bawah pengenal DB. Pada tab Configuration, lihat di bawah Extended Support untuk melihat apakah database terdaftar atau tidak.



Aurora DB atau cluster global dengan Amazon RDS Extended Support

Saat memulihkan Aurora DB atau cluster global, pilih Aktifkan RDS Extended Support di konsol, atau gunakan opsi Extended Support AWS CLI di atau parameter di RDS API.

> **Note**
>
> Jika Anda tidak menentukan pengaturan RDS Extended Support, RDS Extended Support. Perilaku default ini menjaga ketersediaan database Anda melewati akhir Aurora dari tanggal dukungan standar.

Topik

- Pertimbangan untuk RDS Extended Support
- Kembalikan Aurora DB atau cluster global dengan RDS Extended Support

Pertimbangan untuk RDS Extended Support

Sebelum memulihkan Aurora DB atau cluster global, pertimbangkan item berikut:

- Setelah tanggal dukungan standar Aurora berakhir, jika Anda ingin Aurora DB atau cluster global dari Amazon S3, Anda hanya dapat melakukannya dengan menggunakan atau RDS API. AWS CLI Gunakan `--engine-lifecycle-support` opsi dalam AWS CLI perintah restore-db-cluster-from-s3 atau `EngineLifecycleSupport` parameter dalam operasi API RDS RestoreDB ClusterFrom S3.
- Jika Anda ingin mencegah Aurora memulihkan database Anda ke versi RDS Extended Support, `open-source-rds-extended-support-disabled` tentukan di AWS CLI atau RDS API. Dengan demikian, Anda akan menghindari biaya RDS Extended Support terkait.

  Jika Anda menentukan setelan ini, Amazon Aurora akan secara otomatis memutakhirkan database Anda yang dipulihkan ke versi utama yang lebih baru dan didukung. Jika pemutakhiran gagal dalam pemeriksaan pra-pemutakhiran, Amazon Aurora akan kembali dengan aman ke versi mesin RDS Extended Support. Basis data ini akan tetap dalam mode RDS Extended Support, dan Aurora akan menagih Anda untuk RDS Extended Support hingga Anda memutakhirkan database secara manual.

- RDS Extended Support diatur pada tingkat cluster. Anggota cluster akan selalu memiliki pengaturan yang sama untuk RDS Extended Support di konsol RDS, `--engine-lifecycle-support` di AWS CLI, dan `EngineLifecycleSupport` di RDS API.

Untuk informasi selengkapnya, lihat Versi Amazon Aurora.

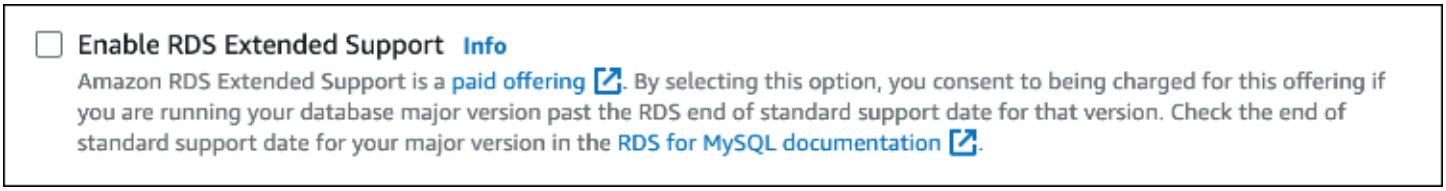
Kembalikan Aurora DB atau cluster global dengan RDS Extended Support

Anda dapat memulihkan Aurora DB atau cluster global dengan versi RDS Extended Support AWS Management Console menggunakan AWS CLI,, atau RDS API.

Konsol

Saat Anda memulihkan cluster Aurora DB atau cluster global instans DB atau cluster , pilih Aktifkan RDS Extended Support di bagian opsi Engine.

Gambar berikut menunjukkan pengaturan Enable RDS Extended Support:



AWS CLI

Saat Anda menggunakan AWS CLI perintah restore-db-cluster-from-snapshot restore-db-instance-from -db-snapshot Support dengan menentukan opsi. `open-source-rds-extended-support` `--engine-lifecycle-support`

Jika Anda ingin menghindari biaya yang terkait dengan RDS Extended Support, atur `--engine-lifecycle-support` opsi ke`open-source-rds-extended-support-disabled`. Secara default, opsi ini diatur ke`open-source-rds-extended-support`.

Anda juga dapat menentukan nilai ini menggunakan AWS CLI perintah berikut:

- restore-db-cluster-from-s3
- restore-db-cluster-to-point-in-time

API RDS

Saat Anda menggunakan operasi RestoreDB RestoreDB InstanceFrom DBSnapshot RDS API, pilih RDS Extended Support dengan menyetel parameter ke. `EngineLifecycleSupport` `open-source-rds-extended-support`

Jika Anda ingin menghindari biaya yang terkait dengan RDS Extended Support, atur `EngineLifecycleSupport` parameternya ke`open-source-rds-extended-support-disabled`. Secara default, parameter ini diatur ke`open-source-rds-extended-support`.

Anda juga dapat menentukan nilai ini menggunakan operasi API RDS berikut:

- DipulihkanB S3 ClusterFrom
- DipulihkanB ClusterToPointInTime

Untuk informasi lebih lanjut tentang memulihkan cluster Aurora DB, ikuti petunjuk untuk mesin DB Anda. Mencadangkan dan memulihkan klaster DB Amazon Aurora

Menggunakan Deployment Blue/Green Amazon RDS untuk pembaruan basis data

Deployment blue/green menyalin lingkungan basis data produksi ke lingkungan penahapan yang terpisah dan tersinkron. Dengan menggunakan Deployment Blue/Green Amazon RDS, Anda dapat membuat perubahan pada basis data di lingkungan penahapan tanpa memengaruhi lingkungan produksi. Misalnya, Anda dapat meningkatkan versi mesin DB besar atau kecil, mengubah parameter basis data, atau membuat perubahan skema di lingkungan penahapannya. Saat Anda siap, Anda dapat mempromosikan lingkungan pementasan menjadi lingkungan basis data produksi baru, dengan waktu henti biasanya di bawah satu menit.

Amazon Aurora menciptakan lingkungan pementasan dengan mengkloning volume penyimpanan Aurora yang mendasarinya di lingkungan produksi. Volume cluster di lingkungan pementasan hanya menyimpan perubahan tambahan yang dibuat pada lingkungan itu.

> **Note**
>
> Saat ini, Penerapan Biru/Hijau didukung untuk RDS Aurora MySQL dan Aurora PostgreSQL. Untuk ketersediaan engine Amazon RDS, lihat Menggunakan Amazon RDS Blue/Green Deployment untuk pembaruan database di Panduan Pengguna Amazon RDS.

Topik

- Gambaran Umum Deployment Blue/Green Amazon RDS untuk Aurora
- Membuat deployment blue/green
- Melihat deployment blue/green
- Mengganti deployment blue/green
- Menghapus deployment blue/green

Gambaran Umum Deployment Blue/Green Amazon RDS untuk Aurora

Dengan menggunakan Deployment Blue/Green Amazon RDS, Anda dapat membuat dan menguji perubahan basis data sebelum menerapkannya di lingkungan produksi. Deployment blue/green menciptakan lingkungan pementasan yang menyalin lingkungan produksi. Dalam deployment blue/green, lingkungan biru adalah lingkungan produksi saat ini. Lingkungan hijau adalah lingkungan pementasannya. Lingkungan pementasan tetap sinkron dengan lingkungan produksi saat ini menggunakan replikasi logis.

Anda dapat membuat perubahan pada klaster DB Aurora di lingkungan hijau tanpa memengaruhi beban kerja produksi. Misalnya, Anda dapat meningkatkan versi mesin DB mayor atau minor atau mengubah parameter basis data di lingkungan pementasannya. Anda dapat menguji perubahan di lingkungan hijau secara menyeluruh. Setelah siap, Anda dapat melakukan switchover lingkungan untuk mempromosikan lingkungan hijau menjadi lingkungan produksi baru. Switchover biasanya memakan waktu kurang dari satu menit tanpa kehilangan data dan tidak perlu mengubah aplikasi.

Karena lingkungan hijau adalah salinan topologi lingkungan dari produksi, klaster DB dan semua instans DB-nya disalin dalam deployment. Lingkungan hijau juga mencakup fitur yang digunakan oleh klaster DB, seperti snapshot klaster DB, Wawasan Performa, Pemantauan yang Ditingkatkan, dan Aurora Serverless v2.

> **Note**
>
> Penerapan Biru/Hijau didukung untuk Aurora MySQL dan Aurora PostgreSQL. Untuk ketersediaan Amazon RDS, lihat Menggunakan Amazon RDS Blue/Green Deployment untuk pembaruan database di Panduan Pengguna Amazon RDS.

Topik

- Manfaat menggunakan Deployment Blue/Green Amazon RDS
- Alur kerja deployment blue/green
- Mengizinkan akses ke operasi deployment blue/green
- Pertimbangan untuk deployment blue/green
- Praktik terbaik untuk deployment blue/green
- Ketersediaan wilayah dan versi

- Batasan untuk deployment blue/green

Manfaat menggunakan Deployment Blue/Green Amazon RDS

Dengan menggunakan Deployment Blue/Green Amazon RDS, Anda dapat tetap mengikuti perkembangan patch keamanan, meningkatkan performa basis data, dan mengadopsi fitur basis data yang lebih baru dengan waktu henti yang singkat dan dapat diprediksi. Deployment blue/green mengurangi risiko dan waktu henti untuk pembaruan basis data, seperti peningkatan versi mesin mayor atau minor.

Deployment blue/green memberikan manfaat berikut:

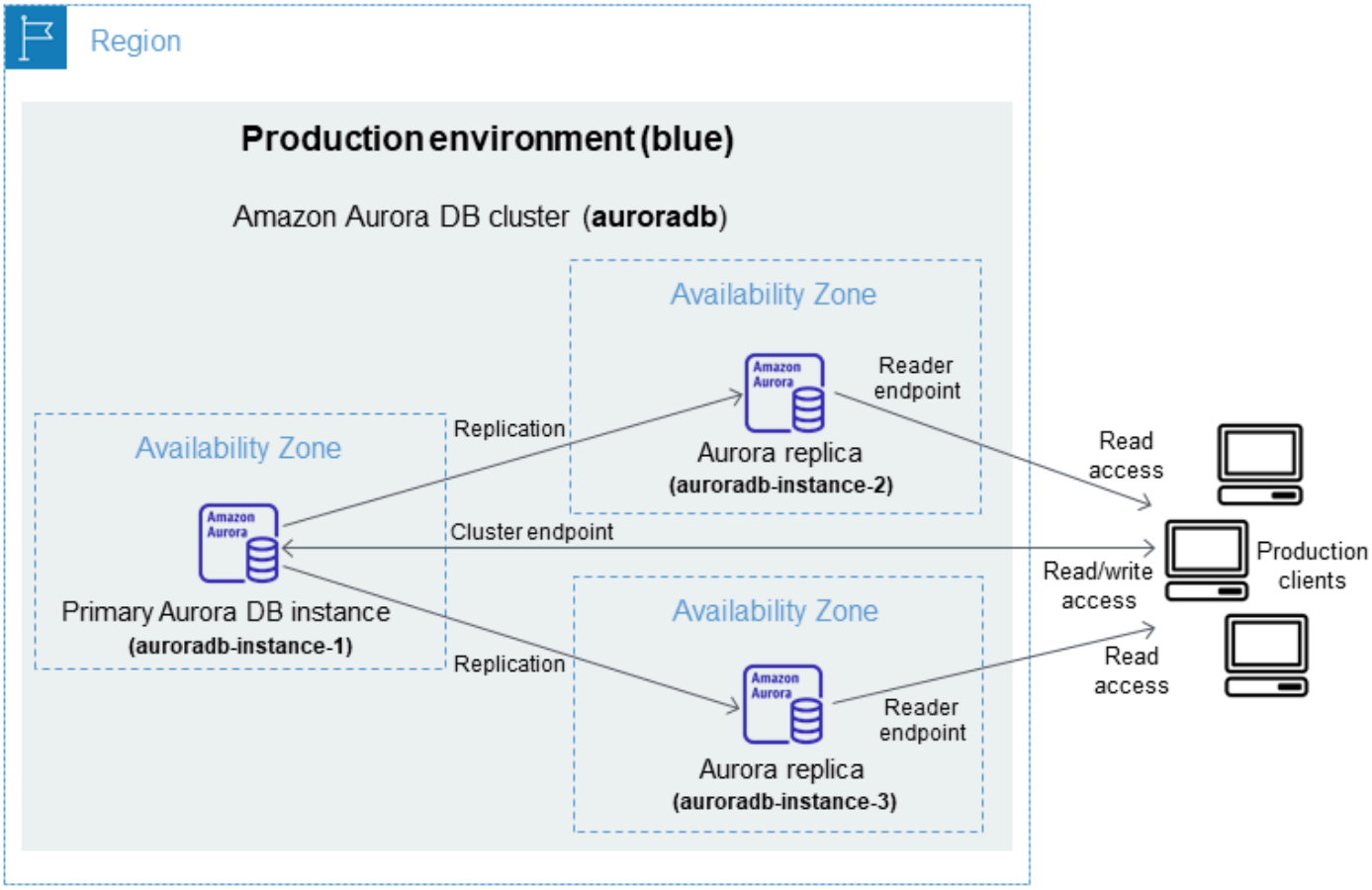
- Memudahkan pembuatan lingkungan pementasan siap produksi.
- Mereplikasi otomatis perubahan basis data dari lingkungan produksi ke lingkungan pementasan.
- Menguji perubahan basis data di lingkungan pementasan yang aman tanpa memengaruhi lingkungan produksi.
- Anda dapat mengikuti perkembangan terbaru dengan patch basis data dan pembaruan sistem.
- Menerapkan dan menguji fitur basis data yang lebih baru.
- Melakukan switchover pada lingkungan pementasan untuk menjadi lingkungan produksi baru tanpa perubahan pada aplikasi.
- Melakukan switchover dengan aman melalui penggunaan pagar pembatas switchover default.
- Tidak ada kehilangan data selama switchover.
- Melakukan switchover dengan cepat, biasanya kurang dari satu menit tergantung beban kerja Anda.

Alur kerja deployment blue/green

Selesaikan langkah-langkah utama berikut saat Anda menggunakan deployment blue/green untuk pembaruan klaster DB Aurora.

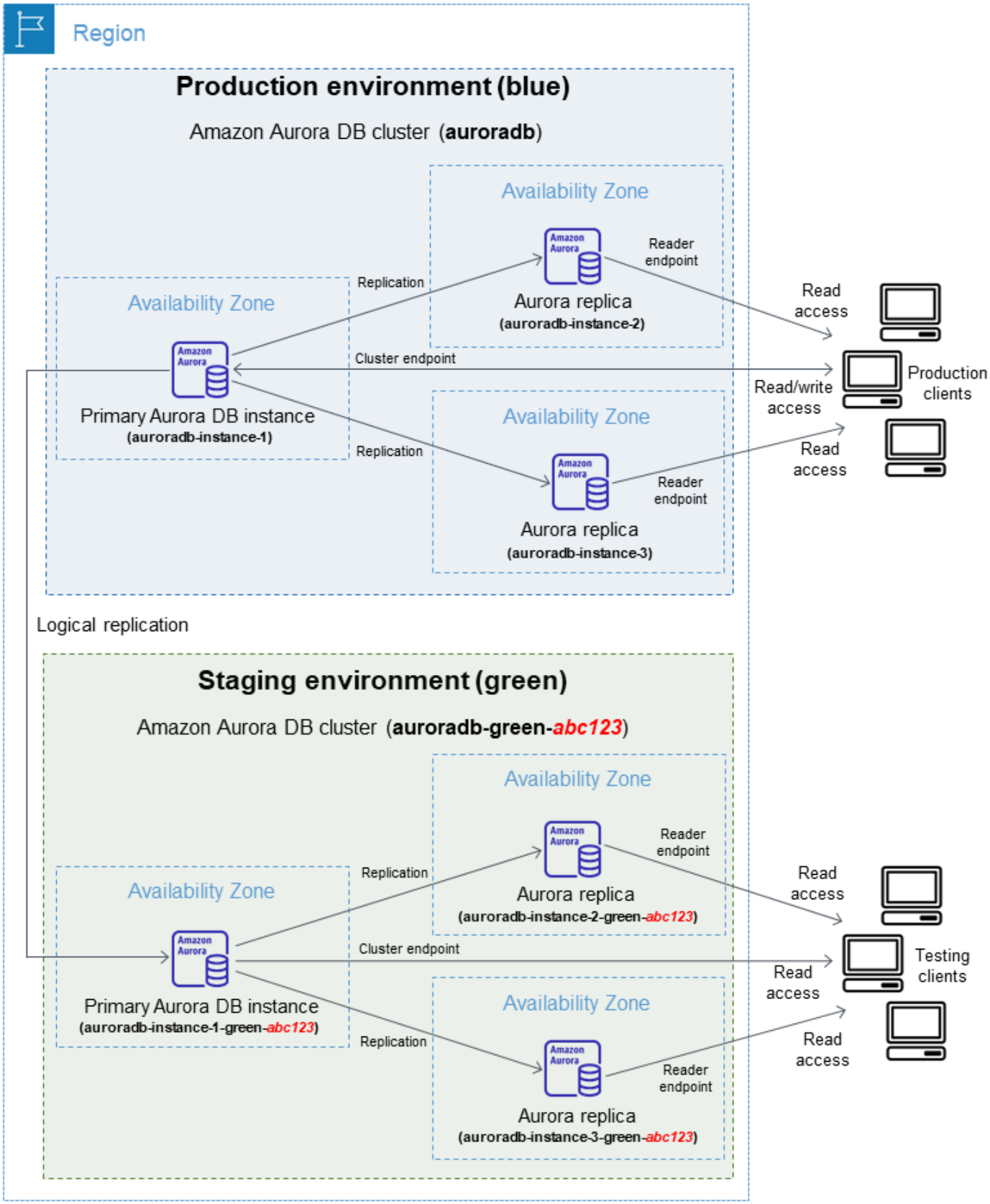
1. Identifikasi klaster DB produksi yang memerlukan pembaruan.

   Gambar berikut menunjukkan contoh klaster DB produksi.



2. Buat deployment blue/green. Untuk petunjuknya, lihat Membuat deployment blue/green.

   Gambar berikut menunjukkan contoh deployment blue/green pada lingkungan produksi dari langkah 1. Saat membuat deployment blue/green, RDS menyalin topologi lengkap dan konfigurasi klaster Aurora DB untuk menciptakan lingkungan hijau. Nama-nama klaster DB dan instans DB yang disalin ditambahkan dengan `-green-`*`random-characters`*. Lingkungan pementasan dalam gambar berisi klaster DB (auroradb-green-*abc123*). Gambar tersebut juga berisi tiga instans DB dalam klaster DB (auroradb-instance1-green-*abc123*, auroradb-instance2-green-*abc123*, dan auroradb-instance3-green-*abc123*).


Region
Production environment (blue)
Amazon Aurora DB cluster (auroradb)
Availability Zone
Amazon Aurora
Reader endpoint
Aurora replica
(auroradb-instance-2)
Replication
Availability Zone
Cluster endpoint
Primary Aurora DB instance
(auroradb-instance-1)
Availability Zone
Replication
Reader endpoint
Aurora replica
(auroradb-instance-3)
Read access
Read/write access
Production clients
Read access
Logical replication
Staging environment (green)
Amazon Aurora DB cluster (auroradb-green-abc123)
Availability Zone
Reader endpoint
Aurora replica
(auroradb-instance-2-green-abc123)
Replication
Availability Zone
Cluster endpoint
Primary Aurora DB instance
(auroradb-instance-1-green-abc123)
Availability Zone
Replication
Reader endpoint
Aurora replica
(auroradb-instance-3-green-abc123)
Read access
Read access
Testing clients
Read access

Saat membuat deployment blue/green, Anda dapat menentukan versi mesin DB yang lebih tinggi dan grup parameter klaster DB yang berbeda untuk klaster DB di lingkungan hijau. Anda juga dapat menentukan grup parameter DB yang berbeda untuk instans DB di klaster DB.

RDS juga mengonfigurasi replikasi dari instans DB primer di lingkungan biru ke instans DB primer di lingkungan hijau.

> **Important**
>
> Untuk Aurora MySQL versi 3, setelah Anda membuat penyebaran biru/hijau, cluster DB di lingkungan hijau memungkinkan operasi tulis secara default. Kami menyarankan Anda membuat cluster DB hanya-baca dengan menyetel `read_only` parameter ke 1 dan me-reboot cluster.

3. Buat perubahan pada lingkungan pementasan.

   Misalnya, Anda dapat membuat perubahan skema pada basis data Anda atau mengubah kelas instans DB yang digunakan oleh satu atau beberapa instans DB di lingkungan hijau.

   Untuk informasi tentang memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

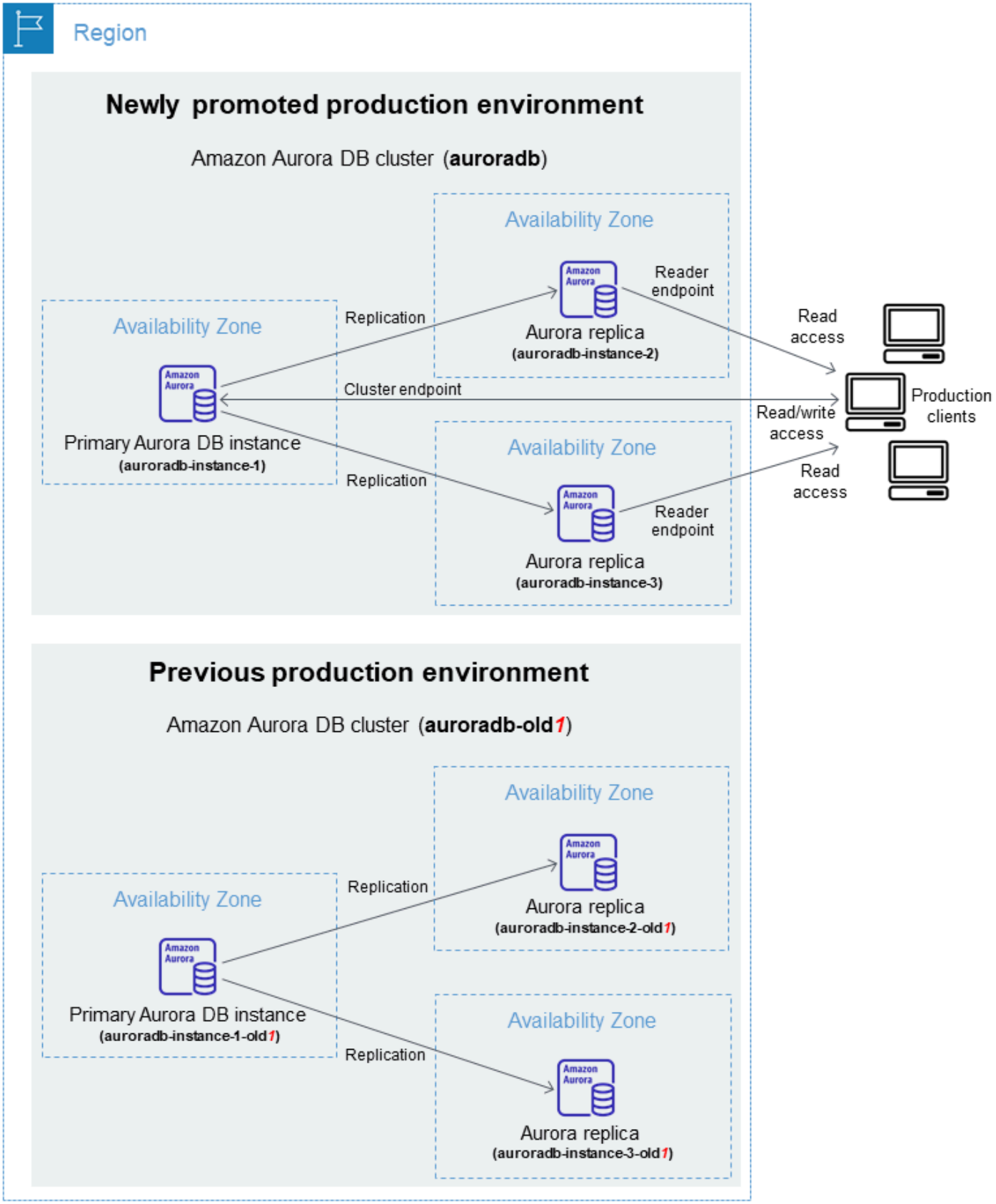
4. Uji lingkungan pementasan Anda.

   Selama pengujian, sebaiknya pertahankan basis data Anda di lingkungan hijau agar hanya dapat dibaca saja. Aktifkan operasi tulis di lingkungan hijau dengan hati-hati karena dapat mengakibatkan konflik replikasi. Hal ini juga dapat menghasilkan data yang tidak diinginkan dalam basis data produksi setelah switchover. Untuk mengaktifkan operasi tulis untuk Aurora MySQL, atur `read_only` parameternya ke`0`, lalu reboot instance DB. Untuk Aurora PostgreSQL, atur parameter ke level sesi. `default_transaction_read_only off`

5. Saat siap, lakukan switchover untuk mempromosikan lingkungan pementasan menjadi lingkungan produksi baru. Untuk petunjuknya, lihat Mengganti deployment blue/green.

   Switchover menyebabkan waktu henti. Waktu henti biasanya kurang dari satu menit, tetapi bisa lebih lama tergantung beban kerja Anda.

   Gambar berikut menunjukkan klaster DB setelah switchover.


Region
Newly promoted production environment
Amazon Aurora DB cluster (auroradb)
Availability Zone
Amazon Aurora
Reader endpoint
Availability Zone
Replication
Aurora replica
(auroradb-instance-2)
Read access
Amazon Aurora
Cluster endpoint
Production clients
Read/write access
Primary Aurora DB instance
(auroradb-instance-1)
Availability Zone
Replication
Read access
Amazon Aurora
Reader endpoint
Aurora replica
(auroradb-instance-3)
Previous production environment
Amazon Aurora DB cluster (auroradb-old1)
Availability Zone
Availability Zone
Replication
Amazon Aurora
Aurora replica
(auroradb-instance-2-old1)
Primary Aurora DB instance
(auroradb-instance-1-old1)
Availability Zone
Replication
Amazon Aurora
Aurora replica
(auroradb-instance-3-old1)

Setelah switchover, klaster DB Aurora di lingkungan hijau menjadi klaster DB produksi baru. Nama dan titik akhir di lingkungan produksi saat ini ditetapkan ke lingkungan produksi yang baru dipromosikan, sehingga Anda tidak perlu melakukan perubahan pada aplikasi. Akibatnya, lalu lintas produksi Anda sekarang mengalir ke lingkungan produksi baru. Klaster DB dan instans DB di lingkungan biru diganti namanya dengan menambahkan `-old`*n* ke nama saat ini, dengan *n* adalah angka. Misalnya, anggap nama instans DB di lingkungan biru adalah `auroradb-instance-1`. Setelah switchover, nama instans DB bisa jadi `auroradb-instance-1-old1`.

Dalam contoh pada gambar, perubahan berikut terjadi selama switchover:

- Klaster DB lingkungan hijau `auroradb-green-abc123` menjadi klaster DB produksi bernama `auroradb`.
- Instans DB lingkungan hijau bernama `auroradb-instance1-green-abc123` menjadi instans DB produksi `auroradb-instance1`.
- Instans DB lingkungan hijau bernama `auroradb-instance2-green-abc123` menjadi instans DB produksi `auroradb-instance2`.
- Instans DB lingkungan hijau bernama `auroradb-instance3-green-abc123` menjadi instans DB produksi `auroradb-instance3`.
- Klaster DB lingkungan biru bernama `auroradb` menjadi `auroradb-old1`.
- Instans DB lingkungan biru bernama `auroradb-instance1` menjadi `auroradb-instance1-old1`.
- Instans DB lingkungan biru bernama `auroradb-instance2` menjadi `auroradb-instance2-old1`.
- Instans DB lingkungan biru bernama `auroradb-instance3` menjadi `auroradb-instance3-old1`.

6. Jika Anda tidak lagi memerlukan deployment blue/green, Anda dapat menghapusnya. Untuk petunjuknya, lihat Menghapus deployment blue/green.

   Setelah switchover, lingkungan produksi sebelumnya tidak dihapus sehingga Anda dapat menggunakannya untuk pengujian regresi, jika perlu.

Mengizinkan akses ke operasi deployment blue/green

Pengguna harus memiliki izin yang diperlukan untuk melakukan operasi yang terkait dengan deployment blue/green. Anda dapat membuat kebijakan IAM yang memberi izin kepada pengguna

dan peran untuk menjalankan operasi API tertentu pada sumber daya yang diperlukan. Anda kemudian dapat melampirkan kebijakan tersebut ke set izin IAM atau peran yang memerlukan izin tersebut. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Amazon Aurora.

Pengguna yang membuat deployment blue/green harus memiliki izin untuk menjalankan operasi RDS berikut:

- `rds:AddTagsToResource`
- `rds:CreateDBCluster`
- `rds:CreateDBInstance`
- `rds:CreateDBClusterEndpoint`

Pengguna yang melakukan switchover pada deployment blue/green harus memiliki izin untuk menjalankan operasi RDS berikut:

- `rds:ModifyDBCluster`
- `rds:PromoteReadReplicaDBCluster`

Pengguna yang menghapus deployment blue/green harus memiliki izin untuk menjalankan operasi RDS berikut:

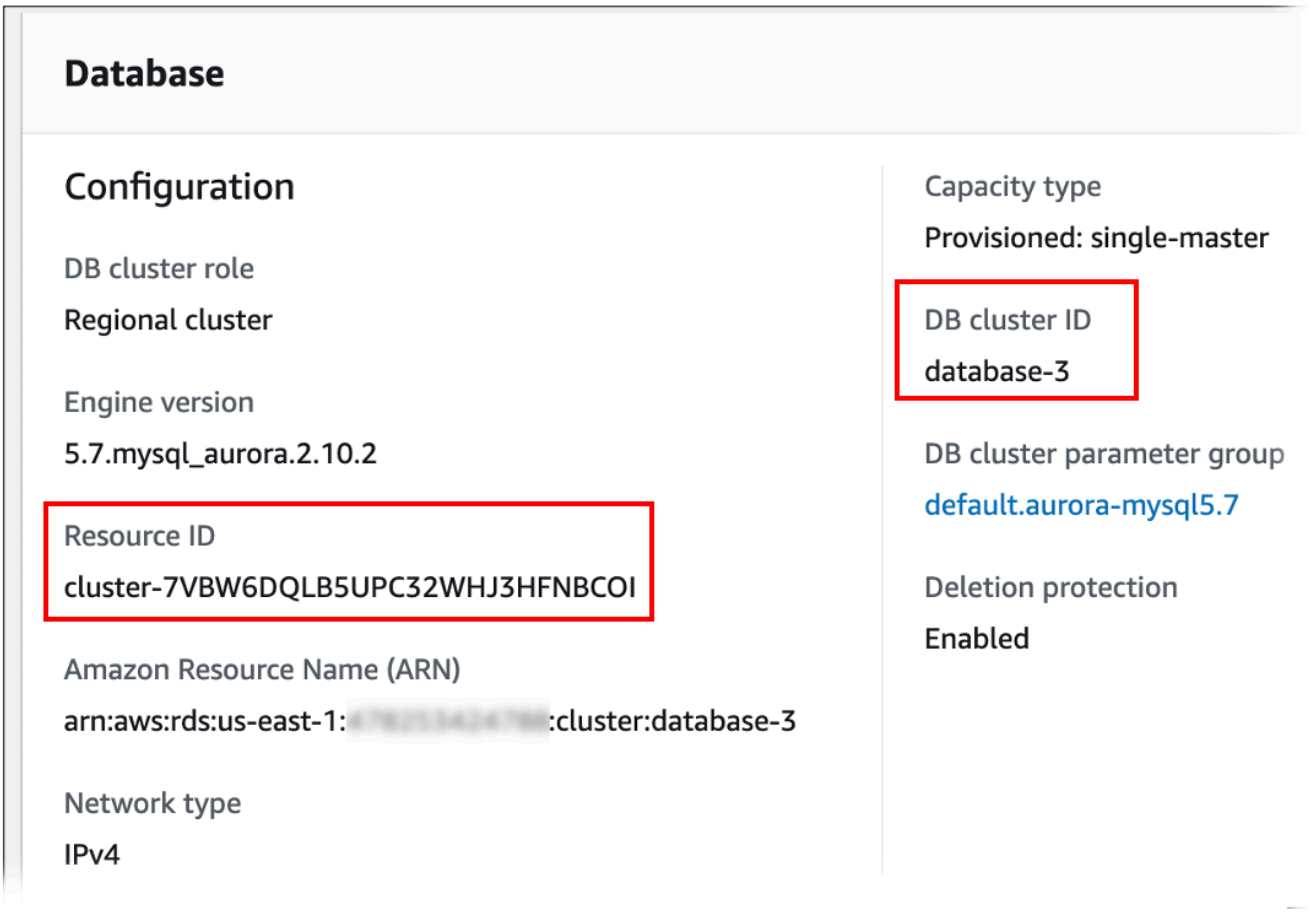
- `rds:DeleteDBCluster`
- `rds:DeleteDBInstance`
- `rds:DeleteDBClusterEndpoint`

Aurora menyediakan dan memodifikasi sumber daya di lingkungan pementasan atas nama Anda. Sumber daya ini mencakup instance DB yang menggunakan konvensi penamaan yang ditentukan secara internal. Oleh karena itu, kebijakan IAM terlampir tidak dapat berisi pola nama sumber daya sebagian sepertimy`-db-prefix-*`. Hanya wildcard (*) yang didukung. Secara umum, sebaiknya gunakan tag sumber daya dan atribut lain yang didukung untuk mengontrol akses ke sumber daya ini, bukan wildcard. Untuk informasi selengkapnya, lihat Kunci tindakan, sumber daya, dan kondisi untuk Amazon RDS.

Pertimbangan untuk deployment blue/green

Amazon RDS melacak sumber daya di deployment blue/green dengan `DbiResourceId` dan `DbClusterResourceId` dari setiap sumber daya. ID sumber daya ini adalah pengenal Wilayah AWS-unik dan tidak dapat diubah untuk sumber daya.

ID sumber daya terpisah dari ID instans DB:



Nama (ID klaster) sumber daya berubah saat Anda switchover deployment blue/green, tetapi setiap sumber daya menyimpan ID sumber daya yang sama. Misalnya, pengidentifikasi klaster DB mungkin adalah `mycluster` di lingkungan biru. Setelah switchover, klaster DB yang sama mungkin diganti namanya menjadi `mycluster-old1`. Namun, ID sumber daya klaster DB tidak berubah selama switchover. Jadi, ketika sumber daya hijau dipromosikan menjadi sumber daya produksi baru, ID sumber dayanya tidak cocok dengan ID sumber daya biru yang sebelumnya diproduksi.

Setelah switchover deployment blue/green, sebaiknya perbarui ID sumber daya ke sumber daya produksi yang baru dipromosikan untuk fitur dan layanan terintegrasi yang Anda gunakan dengan sumber daya produksi. Secara khusus, pertimbangkan pembaruan berikut:

- Jika Anda melakukan pemfilteran menggunakan API RDS dan ID sumber daya, sesuaikan ID sumber daya yang digunakan dalam pemfilteran setelah switchover.
- Jika Anda menggunakan CloudTrail untuk mengaudit sumber daya, sesuaikan konsumen CloudTrail untuk melacak ID sumber daya baru setelah peralihan. Untuk informasi selengkapnya, lihat Memantau panggilan API Amazon Aurora di AWS CloudTrail.
- Jika Anda menggunakan Stream Aktivitas Basis Data untuk sumber daya di lingkungan biru, sesuaikan aplikasi Anda untuk memantau peristiwa basis data aliran baru setelah switchover. Untuk informasi selengkapnya, lihat Aliran aktivitas basis data di Aurora.
- Jika Anda menggunakan API Wawasan Performa, sesuaikan ID sumber daya dalam panggilan ke API setelah switchover. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

  Anda dapat memantau basis data dengan nama yang sama setelah switchover, tetapi basis data tersebut tidak berisi data sebelum switchover.
- Jika menggunakan ID sumber daya dalam kebijakan IAM, pastikan Anda menambahkan ID sumber daya dari sumber daya yang baru dipromosikan jika diperlukan. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Amazon Aurora.
- Jika Anda memiliki peran IAM yang terkait dengan Anda, pastikan untuk mengasosiasikannya kembali setelah peralihan. Peran terlampir tidak secara otomatis disalin ke lingkungan hijau.
- Jika Anda mengautentikasi klaster DB menggunakan autentikasi basis data IAM, pastikan kebijakan IAM yang digunakan untuk akses basis data memiliki basis data biru dan hijau yang tercantum di elemen `Resource` kebijakan. Ini diperlukan agar dapat terhubung ke basis data hijau setelah switchover. Untuk informasi selengkapnya, lihat the section called “Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM”.
- Jika Anda ingin memulihkan snapshot klaster DB manual untuk klaster DB yang merupakan bagian dari deployment blue/green, pastikan Anda memulihkan snapshot klaster DB yang benar dengan memeriksa waktu ketika snapshot diambil. Untuk informasi selengkapnya, lihat Memulihkan dari snapshot klaster DB.
- Amazon Aurora menciptakan lingkungan hijau dengan mengkloning volume penyimpanan Aurora yang mendasarinya di lingkungan biru. Volume klaster hijau hanya menyimpan perubahan tambahan yang dilakukan pada lingkungan hijau. Jika Anda menghapus klaster DB di lingkungan biru, ukuran volume penyimpanan Aurora yang mendasarinya di lingkungan hijau meningkat ke ukuran penuh. Untuk informasi selengkapnya, lihat the section called “Mengkloning volume untuk klaster DB Aurora”.

- Saat Anda menambahkan instans DB ke klaster DB di lingkungan hijau deployment blue/green, instans DB baru tidak akan menggantikan instans DB di lingkungan biru saat Anda switchover. Namun, instans DB baru dipertahankan di klaster DB dan menjadi instans DB di lingkungan produksi baru.
- Saat Anda menghapus instans DB di klaster DB di lingkungan hijau deployment blue/green, Anda tidak dapat membuat instans DB baru untuk menggantikannya dalam deployment blue/green.

  Jika Anda membuat instans DB baru dengan nama dan ARN yang sama dengan instans DB yang dihapus, instans tersebut memiliki `DbiResourceId` yang berbeda, jadi instans tersebut bukan bagian dari lingkungan hijau.

  Perilaku berikut terjadi jika Anda menghapus instans DB di klaster DB di lingkungan hijau:

  - Jika ada instans DB di lingkungan biru dengan nama yang sama, instans tersebut tidak akan switchover ke instans DB di lingkungan hijau. Instans DB ini tidak akan diganti namanya dengan menambahkan `-old`*n* ke nama instans DB.
  - Aplikasi apa pun yang menunjuk ke instans DB di lingkungan biru terus menggunakan instans DB yang sama setelah switchover.

Praktik terbaik untuk deployment blue/green

Berikut ini adalah praktik terbaik untuk deployment blue/green:

Praktik terbaik umum

- Uji klaster DB Aurora secara menyeluruh di lingkungan hijau sebelum switchover.
- Simpan basis data Anda di lingkungan hijau dengan kondisi hanya baca. Sebaiknya Anda mengaktifkan operasi tulis di lingkungan hijau dengan hati-hati karena dapat mengakibatkan konflik replikasi. Hal ini juga dapat menghasilkan data yang tidak diinginkan dalam basis data produksi setelah switchover.
- Saat menggunakan deployment blue/green untuk mengimplementasikan perubahan skema, hanya buat perubahan yang kompatibel dengan replikasi.

  Misalnya, Anda dapat menambahkan kolom baru di akhir tabel, membuat indeks, atau menghapus indeks tanpa mengganggu replikasi dari deployment biru ke deployment hijau. Namun, perubahan skema, seperti penggantian nama kolom atau nama tabel, memecah replikasi ke deployment hijau.

Untuk informasi selengkapnya tentang perubahan yang kompatibel dengan replikasi, lihat Replication with Differing Table Definitions on Source and Replica di dokumentasi MySQL dan Restrictions dalam dokumentasi replikasi logis PostgreSQL.

- Gunakan titik akhir klaster, titik akhir pembaca, atau titik akhir kustom untuk semua koneksi di kedua lingkungan. Jangan gunakan titik akhir instans atau titik akhir kustom dengan daftar statis atau pengecualian.
- Saat Anda mengalihkan deployment blue/green, ikuti praktik terbaik switchover. Untuk informasi selengkapnya, lihat the section called “Praktik terbaik switchover”.

Praktik terbaik Aurora PostgreSQL

- Pantau cache penulisan replikasi logis Aurora PostgreSQL dan buat penyesuaian pada buffer cache jika perlu. Untuk informasi selengkapnya, lihat the section called “Memantau cache penulisan replikasi logis”.
- Jika database Anda memiliki memori freeable yang cukup, tingkatkan nilai parameter `logical_decoding_work_mem` DB di lingkungan biru. Tindakan ini memungkinkan lebih sedikit decoding pada disk, alih-alih menggunakan memori. Anda dapat memantau memori yang dapat dibebaskan dengan `FreeableMemory` CloudWatch metrik. Untuk informasi selengkapnya, lihat the section called “CloudWatch metrik untuk Aurora”.
- Perbarui semua ekstensi PostgreSQL Anda ke versi terbaru sebelum membuat deployment blue/green. Untuk informasi selengkapnya, lihat the section called “Meningkatkan ekstensi PostgreSQL”.
- Jika Anda menggunakan ekstensi `aws_s3`, pastikan Anda memberikan akses klaster DB hijau ke Amazon S3 melalui peran IAM setelah lingkungan hijau dibuat. Hal ini memungkinkan perintah impor dan ekspor untuk terus berfungsi setelah switchover. Untuk petunjuknya, lihat the section called “Menyiapkan akses ke bucket Amazon S3”.

Ketersediaan wilayah dan versi

Ketersediaan fitur dan dukungan bervariasi di seluruh versi spesifik dari setiap mesin basis data, dan di seluruh bagian Wilayah AWS. Untuk informasi selengkapnya tentang versi dan ketersediaan Wilayah dengan Deployment Blue/Green Amazon RDS, lihat Deployment Blue/Green.

Batasan untuk deployment blue/green

Batasan berikut berlaku untuk deployment blue/green.

Topik

- Batasan umum untuk deployment blue/green
- Batasan ekstensi PostgreSQL untuk deployment blue/green
- Batasan untuk perubahan dalam deployment blue/green
- Batasan replikasi logis PostgreSQL untuk deployment blue/green

Batasan umum untuk deployment blue/green

Batasan umum berikut berlaku untuk deployment blue/green:

- Aurora MySQL versi 2.08 dan 2.09 tidak didukung sebagai sumber peningkatan atau versi target.
- Anda tidak dapat berhenti dan memulai klaster yang merupakan bagian dari deployment blue/green.
- Penerapan biru/hijau tidak mendukung pengelolaan kata sandi pengguna utama dengan. AWS Secrets Manager
- Untuk Aurora MySQL, klaster DB sumber tidak dapat berisi basis data yang bernama `tmp`. Basis data dengan nama ini tidak akan disalin ke lingkungan hijau.
- Untuk Aurora PostgreSQL, tabel yang tidak tercatat tidak direplikasi ke lingkungan hijau kecuali parameter `rds.logically_replicate_unlogged_tables` diatur ke 1 pada klaster DB biru. Sebaiknya Anda tidak mengubah nilai parameter ini setelah membuat deployment blue/green untuk menghindari kemungkinan kesalahan replikasi pada tabel yang tidak tercatat.
- Untuk Aurora PostgreSQL RDS untuk PostgreSQLpenerbit) atau replika (pelanggan) yang dikelola sendiri.
- Selama switchover, lingkungan biru dan hijau tidak boleh memiliki integrasi nol-ETL dengan Amazon Redshift. Anda harus menghapus integrasi tersebut terlebih dahulu dan switchover, lalu membuat ulang integrasi.
- Penjadwal Peristiwa (parameter `event_scheduler`) harus dinonaktifkan di lingkungan hijau saat Anda membuat deployment blue/green. Ini mencegah peristiwa dihasilkan di lingkungan hijau dan menyebabkan inkonsistensi.
- Kebijakan Auto Scaling Aurora apa pun yang ditentukan pada klaster DB biru tidak akan disalin ke lingkungan hijau.
- Penerapan biru/hijau tidak mendukung Driver AWS JDBC untuk MySQL. Untuk informasi selengkapnya, lihat Batasan yang Diketahui pada GitHub.

- Deployment blue/green tidak didukung untuk fitur berikut:
  - Proksi Amazon RDS
  - Replika baca Lintas Wilayah
  - Klaster DB Aurora Serverless v1
  - Klaster DB yang merupakan bagian dari basis data global Aurora
  - Babelfish for Aurora PostgreSQL
  - AWS CloudFormation

Batasan ekstensi PostgreSQL untuk deployment blue/green

Batasan berikut berlaku untuk ekstensi PostgreSQL:

- Ekstensi `pg_partman` harus dinonaktifkan di lingkungan biru saat Anda membuat deployment blue/green. Ekstensi tersebut menjalankan operasi DDL seperti `CREATE TABLE`, yang memecah replikasi logis dari lingkungan biru ke lingkungan hijau.
- Ekstensi `pg_cron` harus tetap dinonaktifkan di semua basis data hijau setelah deployment blue/green dibuat. Ekstensi tersebut memiliki pekerja latar belakang yang berjalan sebagai superuser dan melewati pengaturan hanya baca di lingkungan hijau, yang dapat menyebabkan konflik replikasi.
- Ekstensi `apg_plan_mgmt` harus memiliki parameter `apg_plan_mgmt.capture_plan_baselines` yang diatur ke `off` di semua basis data hijau untuk menghindari konflik kunci primer jika rencana yang identik ditangkap di lingkungan biru. Untuk informasi selengkapnya, lihat the section called “Gambaran umum manajemen rencana kueri Aurora PostgreSQL”.

  Jika ingin menangkap rencana eksekusi di Aurora Replicas, Anda harus memberikan titik akhir klaster DB biru saat memanggil fungsi `apg_plan_mgmt.create_replica_plan_capture`. Ini memastikan bahwa pengambilan rencana terus berfungsi setelah switchover. Untuk informasi selengkapnya, lihat the section called “Mengambil rencana eksekusi Aurora PostgreSQL di Replika”.
- Jika klaster DB biru dikonfigurasi sebagai server asing dari ekstensi pembungkus data asing (FDW), Anda harus menggunakan nama titik akhir klaster, alih-alih alamat IP. Hal ini memungkinkan konfigurasi untuk tetap berfungsi setelah switchover.
- Ekstensi `pg_active` dan `pglogical` harus dinonaktifkan di lingkungan biru saat Anda membuat deployment blue/green. Setelah Anda mempromosikan lingkungan hijau menjadi lingkungan

produksi baru, Anda dapat mengaktifkan ekstensi lagi. Selain itu, basis data biru tidak bisa menjadi pelanggan logis dari instans eksternal.

- Jika Anda menggunakan `pgAudit` ekstensi, ekstensi harus tetap berada di pustaka bersama (`shared_preload_libraries`) pada grup parameter DB khusus untuk instance DB biru dan hijau. Untuk informasi selengkapnya, lihat the section called “Menyiapkan ekstensi pgAdit”.

Batasan untuk perubahan dalam deployment blue/green

Berikut ini adalah batasan untuk perubahan dalam deployment blue/green:

- Anda tidak dapat mengubah klaster DB yang tidak terenkripsi menjadi klaster DB yang terenkripsi.
- Anda tidak dapat mengubah klaster DB yang terenkripsi menjadi klaster DB yang tidak terenkripsi.
- Anda tidak dapat mengubah klaster DB lingkungan biru ke versi mesin yang lebih tinggi daripada klaster DB lingkungan hijau yang sesuai.
- Sumber daya di lingkungan biru dan lingkungan hijau harus berada dalam Akun AWS yang sama.
- Jika lingkungan biru berisi kebijakan Aurora Auto Scaling, kebijakan tersebut tidak disalin ke lingkungan hijau. Anda harus menambahkan kembali kebijakan tersebut secara manual ke lingkungan hijau.

Batasan replikasi logis PostgreSQL untuk deployment blue/green

Deployment blue/green menggunakan replikasi logis agar lingkungan pementasan tetap sinkron dengan lingkungan produksi. PostgreSQL memiliki batasan tertentu yang terkait dengan replikasi logis, yang diterjemahkan ke batasan saat membuat deployment blue/green untuk klaster DB Aurora PostgreSQL.

Tabel berikut menjelaskan batasan replikasi logis yang berlaku untuk deployment blue/green Aurora PostgreSQL.

Batasan	Penjelasan
Pernyataan bahasa definisi data (DDL), seperti `CREATE TABLE`	Jika Aurora mendeteksi perubahan DDL di lingkungan biru, basis data hijau Anda memasukkan status Replikasi terdegradasi. Anda menerima peristiwa yang memberi tahu bahwa perubahan DDL di lingkungan biru tidak dapat direplikasi ke lingkungan hijau. Anda harus

Batasan	Penjelasan
dan `CREATE SCHEMA`, tidak direplikasi dari lingkungan biru ke lingkungan hijau.	menghapus deployment blue/green dan semua basis data hijau, lalu buat kembali semuanya. Jika tidak, Anda tidak dapat switchover deployment blue/green.
Operasi `NEXTVAL` pada objek urutan tidak disinkronkan antara lingkungan biru dan lingkungan hijau.	Selama switchover, Aurora menambah nilai urutan di lingkungan hijau agar sesuai dengan yang ada di lingkungan biru. Jika Anda memiliki ribuan urutan, hal ini dapat menunda switchover.
Pembuatan atau perubahan objek besar di lingkungan biru tidak direplikasi ke lingkungan hijau.	Jika Aurora mendeteksi pembuatan atau perubahan objek besar di lingkungan biru yang disimpan dalam tabel sistem `pg_largeobject` , basis data hijau Anda memasukkan status Replikasi terdegradasi. Aurora menghasilkan peristiwa yang memberi tahu Anda bahwa perubahan objek besar di lingkungan biru tidak dapat direplikasi ke lingkungan hijau. Anda harus menghapus deployment blue/green dan semua basis data hijau, lalu buat kembali semuanya. Jika tidak, Anda tidak dapat switchover deployment blue/green.
Tampilan terwujud tidak secara otomatis disegarkan di lingkungan hijau.	Menyegarkan tampilan terwujud di lingkungan biru tidak akan menyegarkannya di lingkungan hijau. Setelah switchover, Anda dapat menjadwalkan penyegaran tampilan terwujud.

Batasan	Penjelasan
Operasi UPDATE dan DELETE tidak diizinkan pada tabel yang tidak memiliki kunci primer.	Sebelum Anda membuat deployment blue/green, pastikan semua tabel di klaster DB memiliki kunci primer.

Untuk informasi selengkapnya, lihat Restrictions di dokumentasi replikasi logis PostgreSQL.

Membuat deployment blue/green

Saat membuat deployment blue/green, Anda menentukan klaster DB yang akan disalin dalam deployment. Klaster DB yang Anda pilih adalah klaster DB produksi, dan menjadi klaster DB di lingkungan biru. RDS menyalin topologi lingkungan biru ke area pementasan, beserta fitur yang dikonfigurasinya. Klaster DB disalin ke lingkungan hijau, dan RDS mengonfigurasi replikasi dari klaster DB di lingkungan biru ke klaster DB di lingkungan hijau. RDS juga menyalin semua instans DB di klaster DB.

Topik

- Mempersiapkan deployment blue/green
- Menentukan perubahan saat membuat deployment blue/green
- Membuat deployment blue/green

Mempersiapkan deployment blue/green

Topik

- Mempersiapkan cluster DB MySQL Aurora untuk penyebaran biru/hijau
- Mempersiapkan cluster DB PostgreSQL Aurora untuk penerapan biru/hijau

Mempersiapkan cluster DB MySQL Aurora untuk penyebaran biru/hijau

Sebelum Anda membuat deployment blue/green untuk klaster DB Aurora MySQL, klaster harus dikaitkan dengan grup parameter klaster DB kustom dengan pencatatan log biner (`binlog_format`) diaktifkan. Pencatatan log biner diperlukan untuk replikasi dari lingkungan biru ke lingkungan hijau. Meskipun format binlog apa pun berfungsi, kami merekomendasikan `ROW` untuk mengurangi risiko inkonsistensi replikasi. Untuk informasi tentang cara membuat grup parameter klaster DB kustom dan mengatur parameter, lihat the section called “Bekerja dengan grup parameter klaster DB”.

> **Note**
>
> Mengaktifkan pencatatan log biner akan meningkatkan jumlah operasi I/O disk tulis untuk klaster DB. Anda dapat memantau penggunaan IOPS dengan `VolumeWriteIOPs` CloudWatch metrik.

Setelah Anda mengaktifkan logging biner, pastikan untuk me-reboot cluster DB sehingga perubahan Anda berlaku. Deployment blue/green mengharuskan instans penulis disinkronkan dengan grup parameter klaster DB, jika tidak, pembuatan akan gagal. Untuk informasi selengkapnya, lihat Mem-boot ulang instans DB dalam klaster Aurora.

Mempersiapkan cluster DB PostgreSQL Aurora untuk penerapan biru/hijau

Sebelum Anda membuat deployment blue/green untuk klaster DB Aurora PostgreSQL, pastikan untuk melakukan hal berikut:

- Kaitkan klaster dengan grup parameter klaster DB kustom yang mengaktifkan replikasi logis (`rds.logical_replication`). Replikasi logika diperlukan untuk replikasi dari lingkungan biru ke lingkungan hijau.

  Saat Anda mengaktifkan replikasi logis, Anda juga perlu menyetel parameter cluster tertentu, sepertimax_replication_slots,max_logical_replication_workers, danmax_worker_processes. Untuk instruksi untuk mengaktifkan replikasi logis dan menyetel parameter ini, lihatthe section called “Menyiapkan replikasi logis”.

  Selain itu, pastikan bahwa parameter `synchronous_commit` diatur ke on.

  Setelah Anda mengonfigurasi parameter yang diperlukan, pastikan untuk mem-boot ulang klaster DB agar perubahan Anda diterapkan. Deployment blue/green mengharuskan instans penulis

disinkronkan dengan grup parameter klaster DB, jika tidak, pembuatan akan gagal. Untuk informasi selengkapnya, lihat Mem-boot ulang instans DB dalam klaster Aurora.

- Pastikan klaster DB Anda menjalankan versi Aurora PostgreSQL yang kompatibel dengan Deployment Blue/Green. Untuk daftar versi yang kompatibel, lihat the section called “Deployment Blue/Green dengan Aurora PostgreSQL”.
- Pastikan bahwa semua tabel di klaster DB memiliki kunci primer. Replikasi logis PostgreSQL tidak mengizinkan operasi UPDATE atau DELETE pada tabel yang tidak memiliki kunci primer.

Menentukan perubahan saat membuat deployment blue/green

Anda dapat membuat perubahan berikut pada klaster DB di lingkungan hijau saat membuat deployment blue/green.

Anda dapat membuat penyesuaian pada klaster dan instans DB-nya di lingkungan hijau setelah di-deploy. Misalnya, Anda dapat membuat perubahan skema pada basis data Anda.

Untuk informasi tentang memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Menentukan versi mesin yang lebih tinggi

Anda dapat menentukan versi mesin yang lebih tinggi jika ingin menguji peningkatan mesin DB. Setelah switchover, basis data ditingkatkan ke versi mesin DB mayor atau minor yang Anda tentukan.

Menentukan grup parameter DB yang berbeda

Tentukan grup parameter klaster DB yang berbeda dari yang digunakan oleh klaster DB. Anda dapat menguji bagaimana perubahan parameter memengaruhi klaster DB di lingkungan hijau atau menentukan grup parameter untuk versi mesin DB mayor baru jika terjadi peningkatan.

Jika Anda menentukan grup parameter klaster DB yang berbeda, grup parameter yang ditentukan dikaitkan dengan klaster DB di lingkungan hijau. Jika Anda tidak menentukan grup parameter klaster DB yang berbeda, klaster DB di lingkungan hijau dikaitkan dengan grup parameter yang sama dengan klaster DB biru.

Membuat deployment blue/green

Anda dapat membuat penerapan biru/hijau menggunakan AWS Management Console, API AWS CLI, atau RDS.

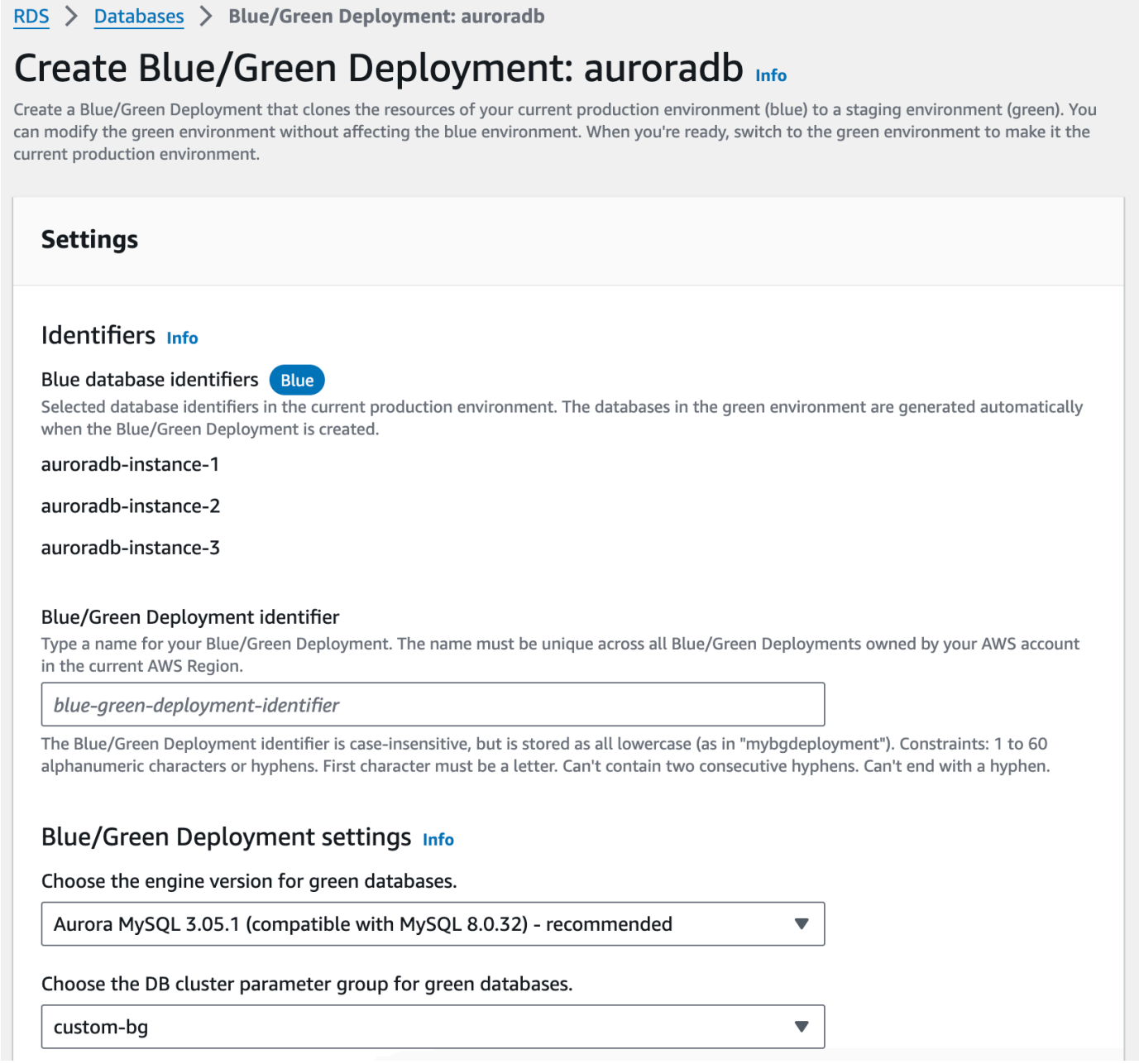
Konsol

Untuk membuat deployment blue/green

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, lalu pilih klaster DB yang ingin disalin ke lingkungan hijau.
3. Pilih Tindakan, Buat Penerapan Biru/Hijau.

   Jika Anda memilih klaster DB Aurora PostgreSQL, tinjau dan konfirmasi batasan replikasi logisnya. Untuk informasi selengkapnya, lihat the section called “Batasan replikasi logis PostgreSQL”.

   Halaman Buat Deployment Blue/Green muncul.



4. Tinjau pengidentifikasi database biru. Pastikan bahwa mereka cocok dengan instans DB yang Anda harapkan di lingkungan biru. Jika tidak, pilih Batalkan.
5. Untuk pengidentifikasi Deployment Blue/Green, masukkan nama untuk deployment blue/green Anda.
6. (Opsional) Untuk pengaturan Deployment Blue/Green, tentukan pengaturan untuk lingkungan hijau:
   - Pilih versi mesin DB jika Anda ingin menguji peningkatan versi mesin DB.
   - Pilih grup parameter klaster DB yang akan dikaitkan dengan klaster DB di lingkungan hijau.

- Pilih grup parameter DB yang akan dikaitkan dengan instans DB di lingkungan hijau.

Anda dapat membuat penyesuaian lain pada basis data di lingkungan hijau setelah di-deploy.

7. Pilih Buat lingkungan pementasan.

AWS CLI

Untuk membuat penyebaran biru/hijau menggunakan AWS CLI, gunakan create-blue-green-deploymentperintah dengan opsi berikut:

- `--blue-green-deployment-name` – Tentukan nama deployment blue/green.
- `--source` – Tentukan ARN dari klaster DB yang ingin Anda salin.
- `--target-engine-version` – Tentukan versi mesin jika Anda ingin menguji peningkatan versi mesin DB di lingkungan hijau. Opsi ini meningkatkan klaster DB di lingkungan hijau ke versi mesin DB yang ditentukan.

  Jika tidak ditentukan, klaster DB di lingkungan hijau dibuat dengan versi mesin yang sama dengan klaster DB di lingkungan biru.
- `--target-db-cluster-parameter-group-name` – Tentukan grup parameter klaster DB yang akan dikaitkan dengan klaster DB di lingkungan hijau.
- `--target-db-parameter-group-name` – Tentukan grup parameter DB yang akan dikaitkan dengan instans DB di lingkungan hijau.

Example Membuat deployment blue/green

Untuk Linux, macOS, atau Unix:

```
aws rds create-blue-green-deployment \
    --blue-green-deployment-name aurora-blue-green-deployment \
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb \
    --target-engine-version 8.0 \
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

Untuk Windows:

```
aws rds create-blue-green-deployment ^
```

```
    --blue-green-deployment-name aurora-blue-green-deployment ^
    --source arn:aws:rds:us-east-2:123456789012:cluster:auroradb ^
    --target-engine-version 8.0 ^
    --target-db-cluster-parameter-group-name mydbclusterparametergroup
```

API RDS

Untuk membuat deployment blue/green menggunakan API Amazon RDS, gunakan operasi `CreateBlueGreenDeployment` dengan parameter berikut:

- `BlueGreenDeploymentName` – Tentukan nama deployment blue/green.
- `Source` – Tentukan ARN dari klaster DB yang ingin Anda salin ke lingkungan hijau.
- `TargetEngineVersion` – Tentukan versi mesin jika Anda ingin menguji peningkatan versi mesin DB di lingkungan hijau. Opsi ini meningkatkan klaster DB di lingkungan hijau ke versi mesin DB yang ditentukan.

  Jika tidak ditentukan, klaster DB di lingkungan hijau dibuat dengan versi mesin yang sama dengan klaster DB di lingkungan biru.
- `TargetDBClusterParameterGroupName` – Tentukan grup parameter klaster DB yang akan dikaitkan dengan klaster DB di lingkungan hijau.
- `TargetDBParameterGroupName` – Tentukan grup parameter DB yang akan dikaitkan dengan instans DB di lingkungan hijau.

Melihat deployment blue/green

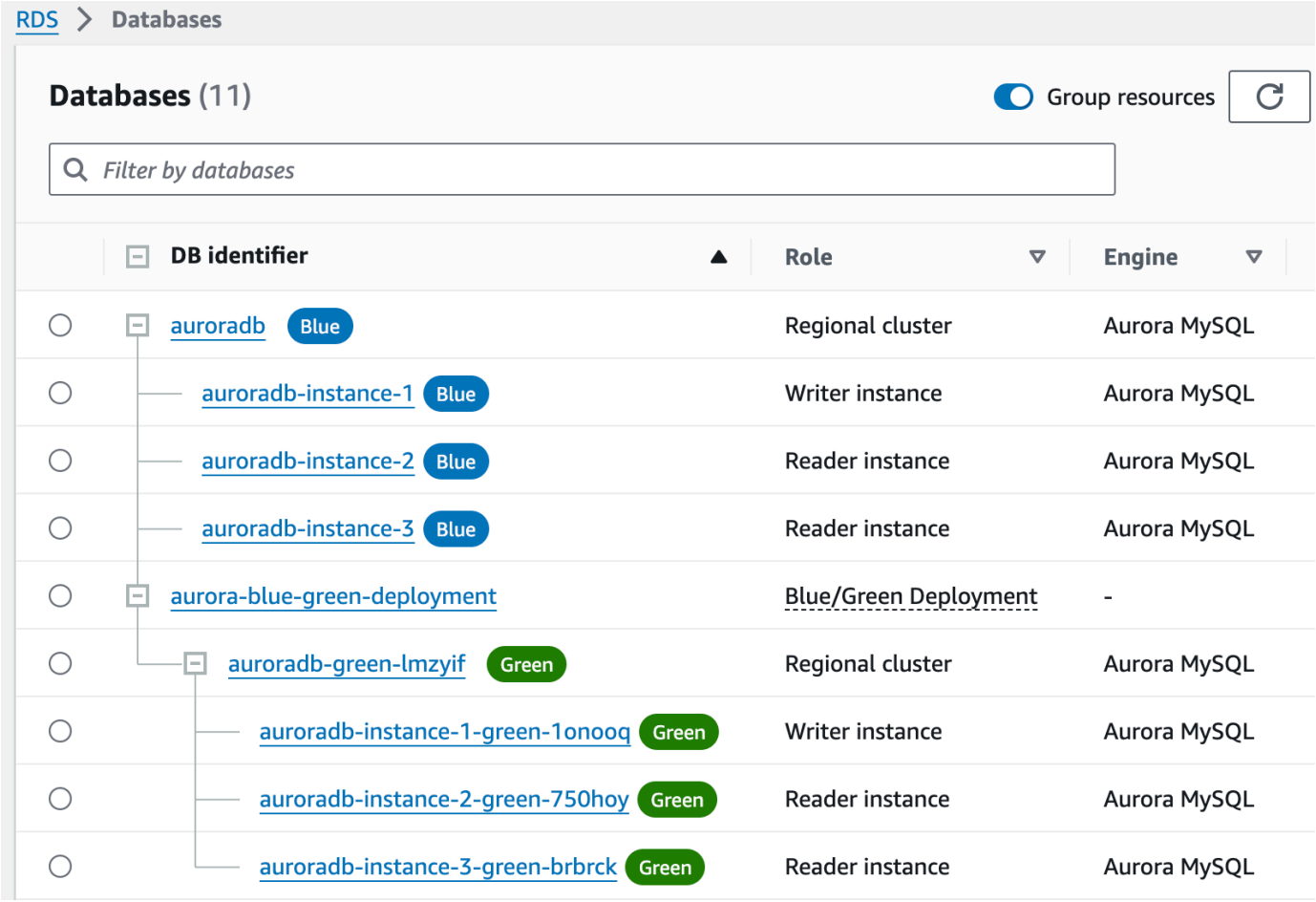
Anda dapat melihat detail tentang deployment blue/green menggunakan AWS Management Console, AWS CLI, atau API RDS.

Anda juga dapat melihat dan berlangganan peristiwa untuk mendapatkan informasi tentang deployment blue/green. Untuk informasi selengkapnya, lihat Peristiwa deployment blue/green.

Konsol

Untuk melihat detail tentang deployment blue/green

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data, lalu temukan deployment blue/green di dalam daftar.

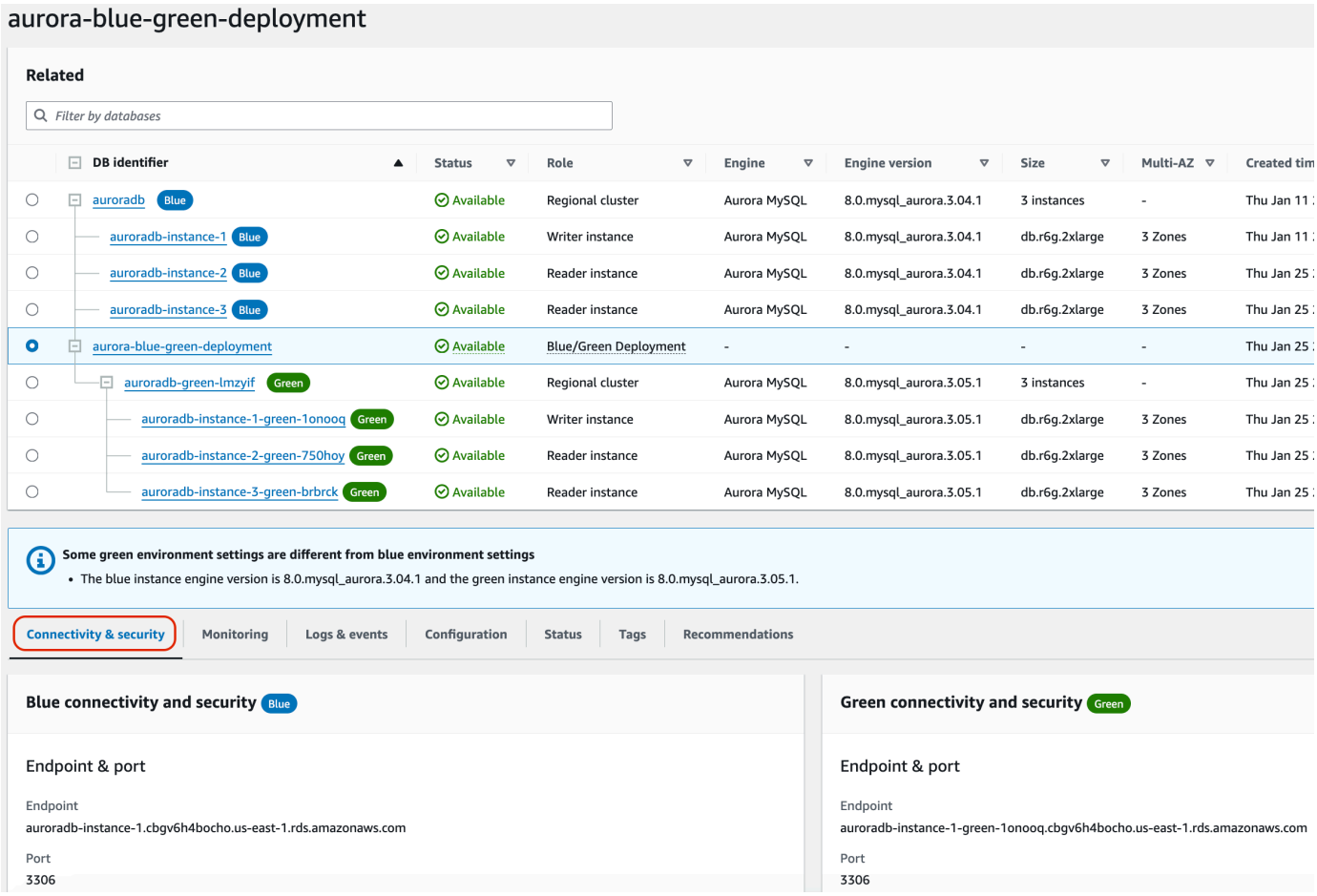


Nilai Peran untuk Deployment adalah Deployment Blue/Green.

3. Pilih nama deployment blue/green hijau yang ingin Anda lihat untuk menampilkan detailnya.

   Setiap tab memiliki bagian untuk deployment biru dan bagian untuk deployment hijau. Misalnya, pada tab Konfigurasi, versi mesin DB mungkin berbeda di lingkungan biru dan di lingkungan hijau jika Anda memutakhirkan versi mesin DB di lingkungan hijau.

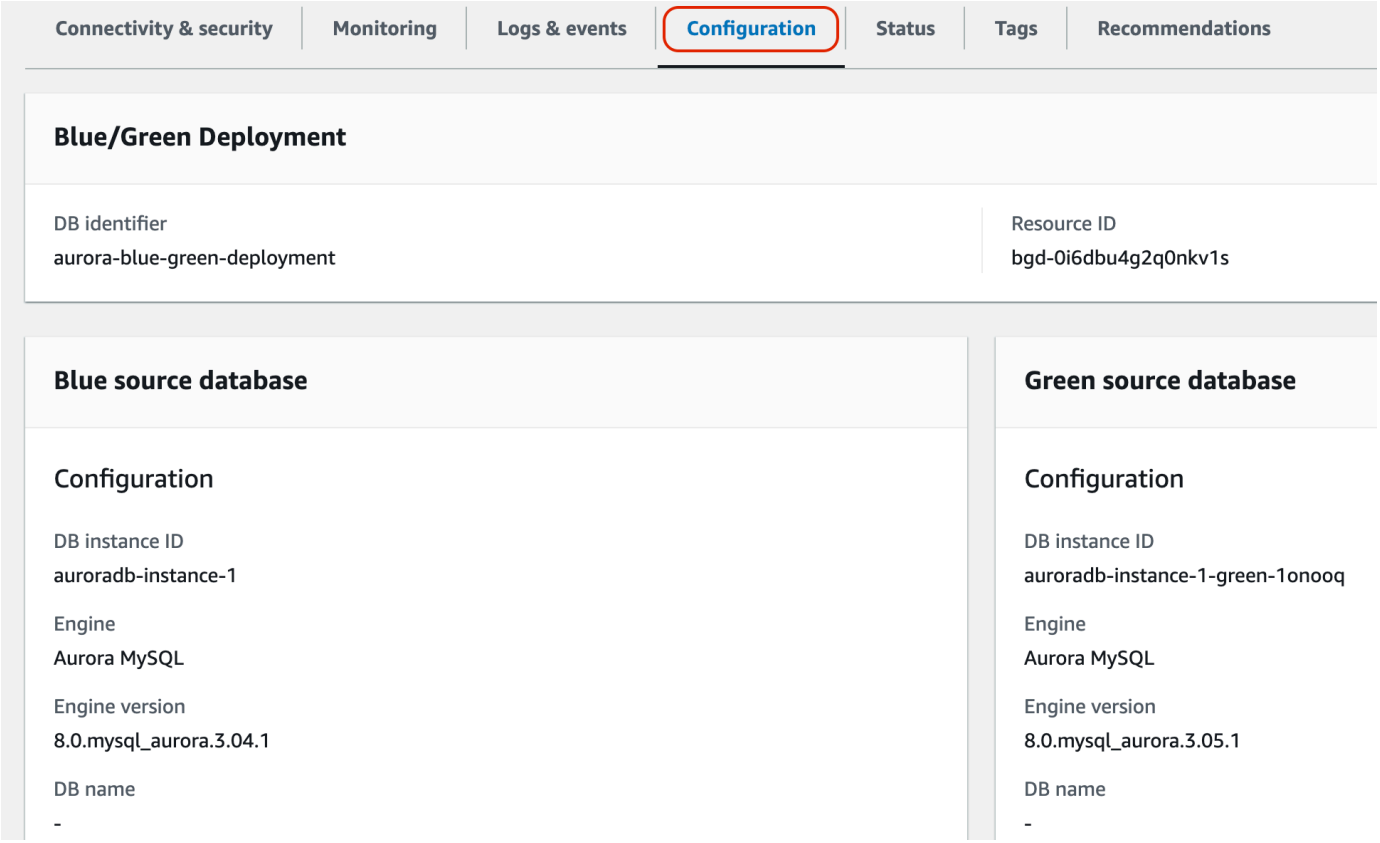
   Gambar berikut menunjukkan contoh tab Konektivitas & keamanan:



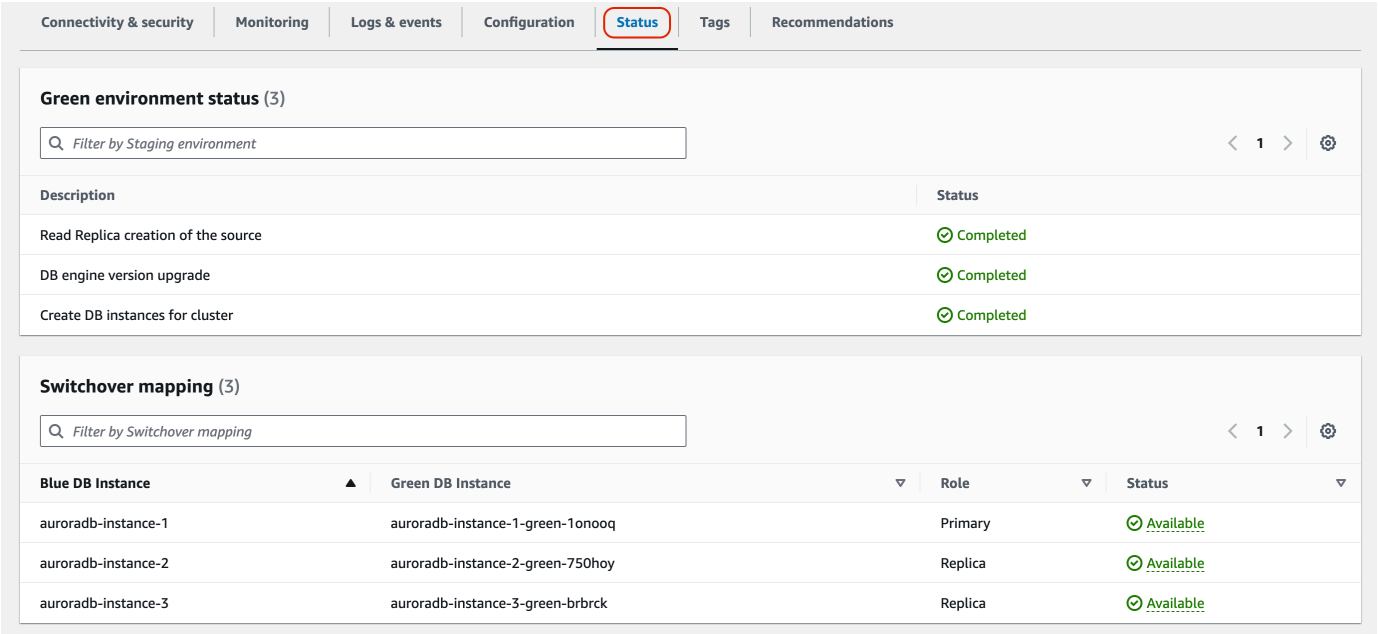
Tab Konektivitas & keamanan juga mencakup bagian yang disebut Replikasi, yang menunjukkan status replikasi logis saat ini dan jeda replika antara lingkungan biru dan hijau. Jika status replikasi adalah `Replicating`, deployment blue/green berhasil direplikasi.

Untuk deployment blue/green Aurora PostgreSQL, status replikasi dapat berubah menjadi `Replication degraded` jika Anda membuat perubahan objek besar atau DDL yang tidak didukung di lingkungan biru. Untuk informasi selengkapnya, lihat the section called “Batasan replikasi logis PostgreSQL”.

Gambar berikut menunjukkan contoh tab Konfigurasi:



Gambar berikut menunjukkan contoh tab Status:



AWS CLI

Untuk melihat detail tentang penyebaran biru/hijau dengan menggunakanAWS CLI, gunakan perintah. describe-blue-green-deployments

Example Melihat detail tentang deployment blue/green dengan memfilter namanya

Saat Anda menggunakan describe-blue-green-deploymentsperintah, Anda dapat memfilter pada file`--blue-green-deployment-name`. Contoh berikut menunjukkan detail untuk deployment blue/green bernama *`my-blue-green-deployment`*.

```
aws rds describe-blue-green-deployments --filters Name=blue-green-deployment-
name,Values=my-blue-green-deployment
```

Example Melihat detail tentang deployment blue/green dengan menentukan pengidentifikasinya

Bila Anda menggunakan describe-blue-green-deploymentsperintah, Anda dapat menentukan`--blue-green-deployment-identifier`. Contoh berikut menunjukkan detail untuk deployment blue/green dengan pengidentifikasi *`bgd-1234567890abcdef`*.

```
aws rds describe-blue-green-deployments --blue-green-deployment-
identifier bgd-1234567890abcdef
```

API RDS

Untuk melihat detail tentang deployment blue/green menggunakan API Amazon RDS, gunakan operasi `DescribeBlueGreenDeployments` dan tentukan `BlueGreenDeploymentIdentifier`.

Mengganti deployment blue/green

Switchover mempromosikan klaster DB, termasuk instans DB-nya, di lingkungan hijau menjadi klaster DB produksi. Sebelum Anda switchover, lalu lintas produksi diarahkan ke klaster di lingkungan biru. Setelah Anda switchover, lalu lintas produksi diarahkan ke klaster DB di lingkungan hijau.

Topik

- Waktu habis switchover
- Pagar pembatas switchover
- Tindakan switchover
- Praktik terbaik switchover

- Memverifikasi CloudWatch metrik sebelum peralihan
- Memantau kelambatan replika sebelum peralihan
- Melakukan switchover pada deployment blue/green
- Setelah switchover

Waktu habis switchover

Anda dapat menentukan periode waktu habis switchover antara 30 detik dan 3.600 detik (satu jam). Jika switchover memakan waktu lebih lama dari durasi yang ditentukan, maka perubahan apa pun akan dikembalikan dan tidak ada perubahan pada lingkungan mana pun. Periode waktu habis default adalah 300 detik (lima menit).

Pagar pembatas switchover

Saat Anda memulai switchover, Amazon RDS menjalankan beberapa pemeriksaan dasar untuk menguji kesiapan lingkungan biru dan hijau untuk switchover. Pemeriksaan ini dikenal sebagai pagar pembatas switchover. Pagar pembatas switchover ini mencegah switchover jika lingkungan belum siap. Oleh karena itu, pagar pembatas tersebut dapat mencegah waktu henti yang lebih lama dari yang diharapkan dan mencegah hilangnya data antara lingkungan biru dan hijau yang mungkin terjadi jika switchover dimulai.

Amazon RDS menjalankan pemeriksaan pagar pembatas berikut pada lingkungan hijau:

- Kondisi replikasi – Memeriksa apakah status replikasi klaster DB hijau berkondisi baik. klaster DB hijau adalah replika dari klaster DB biru.
- Jeda replikasi – Memeriksa apakah jeda replika klaster DB hijau berada dalam batas yang diizinkan untuk switchover. Batas yang diizinkan didasarkan pada periode waktu habis yang ditentukan. Jeda replika menunjukkan seberapa jauh klaster DB tertinggal dari klaster DB biru. Untuk informasi selengkapnya, lihat the section called “Mendiagnosis dan mengatasi jeda di antara replika baca” untuk Aurora MySQL dan the section called “Memantau replikasi” for Aurora PostgreSQL.
- Penulisan aktif – Memastikan tidak ada penulisan aktif pada klaster DB hijau.

Amazon RDS menjalankan pemeriksaan pagar pembatas berikut pada lingkungan biru:

- Replikasi eksternal — Untuk Aurora , pastikan bahwa lingkungan biru bukan sumber logis (penerbit) atau replika (pelanggan) yang dikelola sendiri. Jika ya, kami sarankan Anda melepaskan

slot replikasi dan langganan yang dikelola sendiri di semua database di lingkungan biru, lanjutkan dengan peralihan, lalu buat ulang untuk melanjutkan replikasi. Untuk Aurora MySQL bukan replika binlog eksternal.

- Penulisan aktif yang berjalan lama – Memastikan tidak ada penulisan aktif yang berjalan lama pada klaster DB biru karena dapat meningkatkan jeda replika.
- Pernyataan DDL yang berjalan lama – Memastikan tidak ada pernyataan DDL yang berjalan lama pada instans DB primer biru karena dapat meningkatkan jeda replika.
- Perubahan PostgreSQL yang tidak didukung – Untuk klaster DB Aurora PostgreSQL, memastikan tidak ada perubahan DDL dan tidak ada penambahan atau perubahan objek besar yang dilakukan pada lingkungan biru. Untuk informasi selengkapnya, lihat the section called “Batasan replikasi logis PostgreSQL”.

  Jika Amazon RDS mendeteksi perubahan PostgreSQL yang tidak didukung, status replikasi diubah menjadi `Replication degraded` dan memberi tahu Anda bahwa switchover tidak tersedia untuk deployment blue/green. Untuk melanjutkan switchover, sebaiknya Anda menghapus dan membuat ulang deployment blue/green dan semua basis data hijau. Untuk melakukannya, pilih Tindakan, Hapus dengan basis data hijau.

Tindakan switchover

Saat Anda melakukan switchover pada deployment blue/green, RDS melakukan tindakan berikut:

1. Menjalankan pemeriksaan pagar pembatas untuk memverifikasi apakah lingkungan biru dan hijau siap untuk switchover.
2. Menghentikan operasi tulis baru pada instans DB primer di kedua lingkungan.
3. Memutuskan koneksi ke instans DB di kedua lingkungan dan tidak mengizinkan koneksi baru.
4. Menunggu replikasi untuk mengejar ketertinggalan di lingkungan hijau sehingga lingkungan hijau sinkron dengan lingkungan biru.
5. Mengganti nama DB dan instan DB dan klaster di kedua lingkungan.

   RDS mengganti nama DB instans DB dan klaster di lingkungan hijau agar cocok dengan DB instans DB dan klaster di lingkungan biru. Misalnya, asumsikan nama instans DB di lingkungan biru adalah `mydb`. Asumsikan juga nama instans DB yang sesuai di lingkungan hijau adalah `mydb-green-abc123`. Selama switchover, nama instans DB di lingkungan hijau berubah menjadi `mydb`.

RDS mengganti nama DB instans DB dan klaster di lingkungan biru dengan menambahkan `-old`*n* ke nama saat ini, dengan *n* adalah angka. Misalnya, asumsikan nama instans DB di lingkungan biru adalah `mydb`. Setelah switchover, nama instans DB bisa jadi `mydb-old1`.

RDS juga mengganti nama titik akhir di lingkungan hijau agar sinkron dengan titik akhir yang sesuai di lingkungan biru sehingga perubahan aplikasi tidak diperlukan.

6. Memungkinkan koneksi ke basis data di kedua lingkungan.
7. Memungkinkan operasi tulis pada instans DB primer di lingkungan produksi baru.

   Setelah peralihan, cluster produksi sebelumnya hanya mengizinkan operasi baca Bahkan jika Anda menonaktifkan `read_only` parameter pada cluster DB, itu tetap hanya-baca sampai Anda menghapus penerapan biru/hijau.

Anda dapat memantau status peralihan menggunakan Amazon. EventBridge Untuk informasi selengkapnya, lihat the section called “Peristiwa deployment blue/green”.

Jika tag sudah dikonfigurasi di lingkungan biru, tag tersebut dipindahkan ke lingkungan produksi baru selama switchover. Lingkungan produksi sebelumnya juga mempertahankan tag ini. Untuk informasi selengkapnya tentang tag, lihat Memberi tag pada sumber daya Amazon RDS.

Jika switchover dimulai lalu berhenti sebelum selesai karena alasan apa pun, maka perubahan apa pun akan dikembalikan, dan tidak ada perubahan yang diterapkan pada lingkungan mana pun.

Praktik terbaik switchover

Sebelum melakukan switchover, Anda sangat dianjurkan untuk mengikuti praktik terbaik dengan menyelesaikan tugas-tugas berikut:

- Uji sumber daya secara menyeluruh di lingkungan hijau. Pastikan sumber daya berfungsi dengan baik dan efisien.
- Pantau CloudWatch metrik Amazon yang relevan. Untuk informasi selengkapnya, lihat the section called “Memverifikasi CloudWatch metrik sebelum peralihan”.
- Identifikasi waktu terbaik untuk melakukan switchover.

  Selama switchover, penulisan terputus dari basis data di kedua lingkungan. Identifikasi waktu ketika lalu lintas berada pada titik terendah di lingkungan produksi Anda. Transaksi yang berjalan lama, seperti DDL aktif, dapat meningkatkan waktu switchover Anda, menghasilkan waktu henti yang lebih lama untuk beban kerja produksi.

Jika terdapat banyak koneksi pada klaster DB dan instans DB, pertimbangkan untuk menguranginya secara manual ke jumlah minimum yang diperlukan untuk aplikasi Anda sebelum Anda melakukan switchover pada deployment blue/green. Salah satu cara untuk melakukannya adalah dengan membuat skrip yang memantau status deployment blue/green dan mulai membersihkan koneksi ketika mendeteksi bahwa status telah berubah menjadi `SWITCHOVER_IN_PROGRESS`.

- Pastikan DB instans DB dan klaster di kedua lingkungan berada dalam status `Available`.
- Pastikan klaster DB di lingkungan hijau berkondisi baik dan bereplikasi.
- Pastikan konfigurasi jaringan dan klien Anda tidak meningkatkan cache DNS Time-To-Live (TTL) lebih dari lima detik, yang merupakan default untuk zona DNS Aurora.
  Jika tidak, aplikasi akan terus mengirimkan lalu lintas tulis ke lingkungan biru setelah switchover.
- Anda tidak dapat memutar kembali penerapan biru/hijau setelah peralihan. Untuk beban kerja produksi yang kritis, pertimbangkan untuk menyediakan cluster cadangan sebelum beralih.
- Untuk klaster Aurora PostgreSQL DB untuk instance PostgreSQL DB, lakukan hal berikut:
  - Tinjau batasan replikasi logis dan lakukan tindakan apa pun yang diperlukan sebelum peralihan. Untuk informasi selengkapnya, lihat the section called “Batasan replikasi logis PostgreSQL”.
  - Jalankan operasi `ANALYZE` untuk menyegarkan tabel `pg_statistics`. Ini mengurangi risiko masalah kinerja setelah peralihan.

> **Note**
>
> Selama switchover, Anda tidak dapat mengubah instans DB apa pun yang disertakan dalam switchover.

Memverifikasi CloudWatch metrik sebelum peralihan

Sebelum Anda mengalihkan penerapan biru/hijau, kami sarankan Anda memeriksa nilai metrik berikut di Amazon. CloudWatch

- `DatabaseConnections` – Gunakan metrik ini untuk memperkirakan tingkat aktivitas pada deployment blue/green, dan pastikan nilainya berada pada tingkat yang dapat diterima untuk deployment Anda sebelum Anda switchover. Jika Wawasan Performa diaktifkan, `DBLoad` adalah metrik yang lebih akurat.

- `ActiveTransactions` – Jika `innodb_monitor_enable` diatur ke `all` dalam grup parameter DB untuk salah satu instans DB Anda, gunakan metrik ini untuk melihat apakah ada banyak transaksi aktif yang mungkin memblokir switchover.

Untuk informasi selengkapnya tentang metrik ini, lihat the section called “CloudWatch metrik untuk Aurora”.

Memantau kelambatan replika sebelum peralihan

Sebelum Anda beralih ke penerapan biru/hijau, pastikan bahwa lag replika pada database hijau mendekati nol untuk mengurangi waktu henti.

- Untuk Aurora MySQL, gunakan `AuroraBinlogReplicaLag` CloudWatch metrik untuk mengidentifikasi lag replikasi saat ini pada lingkungan hijau.
- Untuk Aurora PostgreSQL, gunakan kueri SQL berikut:

```
SELECT slot_name,
       confirmed_flush_lsn as flushed,
       pg_current_wal_lsn(),
       (pg_current_wal_lsn() - confirmed_flush_lsn) AS lsn_distance
FROM pg_catalog.pg_replication_slots
WHERE slot_type = 'logical';

slot_name        |    flushed    | pg_current_wal_lsn | lsn_distance
-----------------+---------------+--------------------+------------
logical_replica1 | 47D97/CF32980 | 47D97/CF3BAC8      | 37192
```

`confirmed_flush_lsn`Ini mewakili nomor urutan log terakhir (LSN) yang dikirim ke replika. `pg_current_wall_lsn`Ini mewakili di mana database sekarang. Sebuah `lsn_distance` dari 0 berarti replika tertangkap.

Melakukan switchover pada deployment blue/green

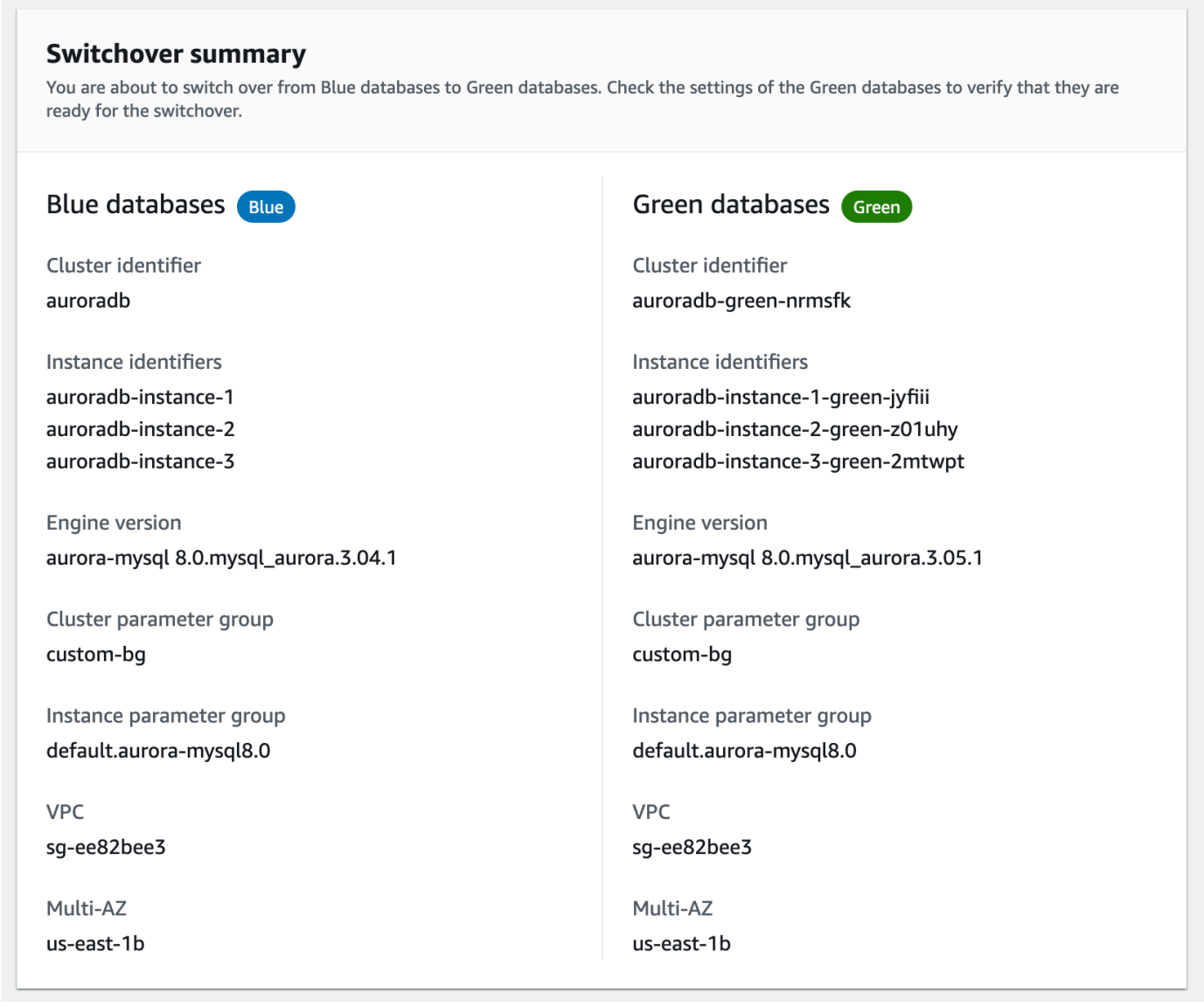
Anda dapat mengalihkan penerapan biru/hijau menggunakan AWS Management Console, API AWS CLI, atau RDS.

Konsol

Untuk melakukan switchover pada deployment blue/green

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih deployment blue/green yang ingin Anda switchover.
3. Untuk Tindakan, pilih Switchover.

   Halaman Switchover muncul.



4. Di halaman Switchover, tinjau ringkasan switchover. Pastikan sumber daya di kedua lingkungan sesuai dengan yang Anda harapkan. Jika tidak, pilih Batalkan.

5. Untuk Pengaturan waktu habis, masukkan batas waktu untuk switchover.
6. Jika klaster menjalankan Aurora PostgreSQL, tinjau dan konfirmasi rekomendasi pra-switchover. Untuk informasi selengkapnya, lihat the section called “Batasan replikasi logis PostgreSQL”.
7. Pilih Switchover.

AWS CLI

Untuk beralih penyebaran biru/hijau dengan menggunakan AWS CLI, gunakan switchover-blue-green-deploymentperintah dengan opsi berikut:

- `--blue-green-deployment-identifier`— Tentukan ID sumber daya dari penerapan biru/hijau.
- `--switchover-timeout` – Tentukan waktu habis untuk switchover, dalam hitungan detik. Angka default-nya adalah 300.

Example Melakukan switchover pada deployment blue/green

Untuk Linux, macOS, atau Unix:

```
aws rds switchover-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --switchover-timeout 600
```

Untuk Windows:

```
aws rds switchover-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --switchover-timeout 600
```

API RDS

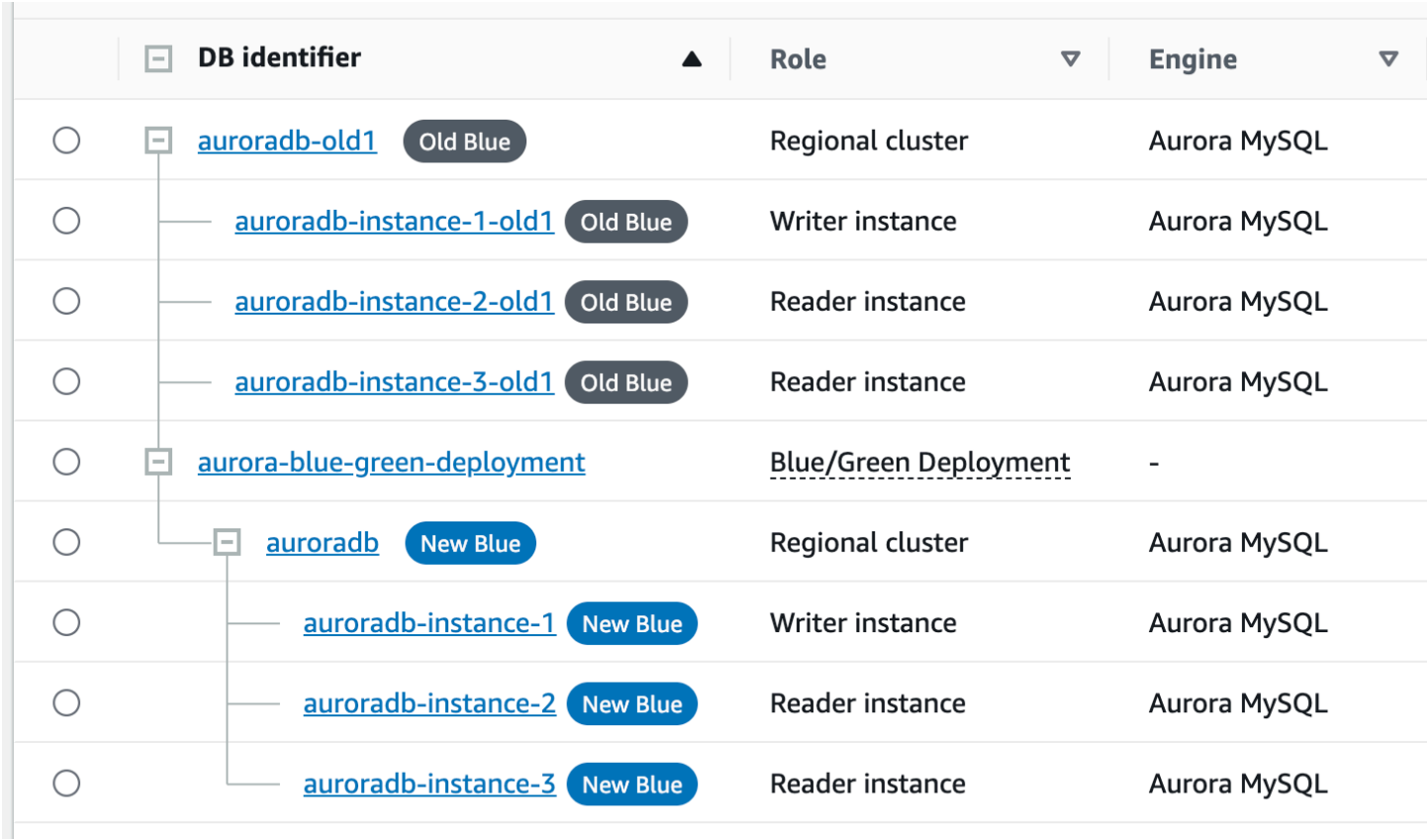
Untuk melakukan switchover pada deployment blue/green menggunakan API Amazon RDS, gunakan operasi `SwitchoverBlueGreenDeployment` dengan parameter berikut:

- `BlueGreenDeploymentIdentifier`— Tentukan ID sumber daya dari penerapan biru/hijau.
- `SwitchoverTimeout` – Tentukan waktu habis untuk switchover, dalam hitungan detik. Angka default-nya adalah 300.

Setelah switchover

Setelah switchover, DB instans DB dan klaster di lingkungan biru sebelumnya akan dipertahankan. Biaya standar berlaku untuk sumber daya ini. Replikasi dan pencatatan log biner antara lingkungan biru dan hijau berhenti.

RDS mengganti nama DB instans DB dan klaster di lingkungan biru dengan menambahkan `-old`*n* ke nama sumber daya saat ini, dengan *n* adalah angka. Cluster DB dipaksa menjadi status hanya-baca. Bahkan jika Anda menonaktifkan `read_only` parameter pada cluster DB, itu tetap hanya-baca sampai Anda menghapus penerapan biru/hijau.



Memperbarui node induk untuk konsumen

Setelah Anda mengalihkan penyebaran biru/hijau, jika cluster DB instans DB memiliki replika eksternal atau konsumen log biner sebelum peralihan, Anda harus memperbarui node induk mereka setelah peralihan untuk mempertahankan kontinuitas replikasi.

Setelah peralihan, instance DB penulis yang sebelumnya berada di lingkungan hijau memancarkan peristiwa yang berisi nama file log master dan posisi log master. Sebagai contoh:

```
aws rds describe-events --output json --source-type db-instance --source-identifier db-
instance-identifier

{
    "Events": [
...
        {
            "SourceIdentifier": "db-instance-identifier",
            "SourceType": "db-instance",
            "Message": "Binary log coordinates in green environment after switchover:
             file mysql-bin-changelog.000003 and position 804",
            "EventCategories": [],
            "Date": "2023-11-10T01:33:41.911Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:db-instance-identifier"
        }
    ]
}
```

Pertama, pastikan bahwa konsumen atau replika telah menerapkan semua log biner dari lingkungan biru tua. Kemudian, gunakan koordinat log biner yang disediakan untuk melanjutkan aplikasi pada konsumen. Misalnya, jika Anda menjalankan replika MySQL di EC2, Anda dapat menggunakan perintah: `CHANGE MASTER TO`

```
CHANGE MASTER TO MASTER_HOST='{new-writer-endpoint}', MASTER_LOG_FILE='mysql-bin-
changelog.000003', MASTER_LOG_POS=804;
```

Menghapus deployment blue/green

Anda dapat menghapus deployment blue/green sebelum atau setelah switchover.

Saat Anda menghapus deployment blue/green sebelum switchover, Amazon RDS secara opsional menghapus klaster DB di lingkungan hijau:

- Jika Anda memilih untuk menghapus klaster DB di lingkungan hijau (`--delete-target`), klaster harus menonaktifkan perlindungan penghapusan.
- Jika Anda tidak menghapus klaster DB di lingkungan hijau (`--no-delete-target`), klaster tersebut dipertahankan, tetapi tidak lagi menjadi bagian dari deployment blue/green. Replikasi berlanjut di antara lingkungan.

Opsi untuk menghapus basis data hijau tidak tersedia di konsol setelah switchover. Saat Anda menghapus penerapan biru/hijau menggunakan AWS CLI, Anda tidak dapat menentukan `--delete-target` opsi jika status penerapannya. `SWITCHOVER_COMPLETED`

> **Important**
>
> Menghapus deployment blue/green tidak memengaruhi lingkungan biru.

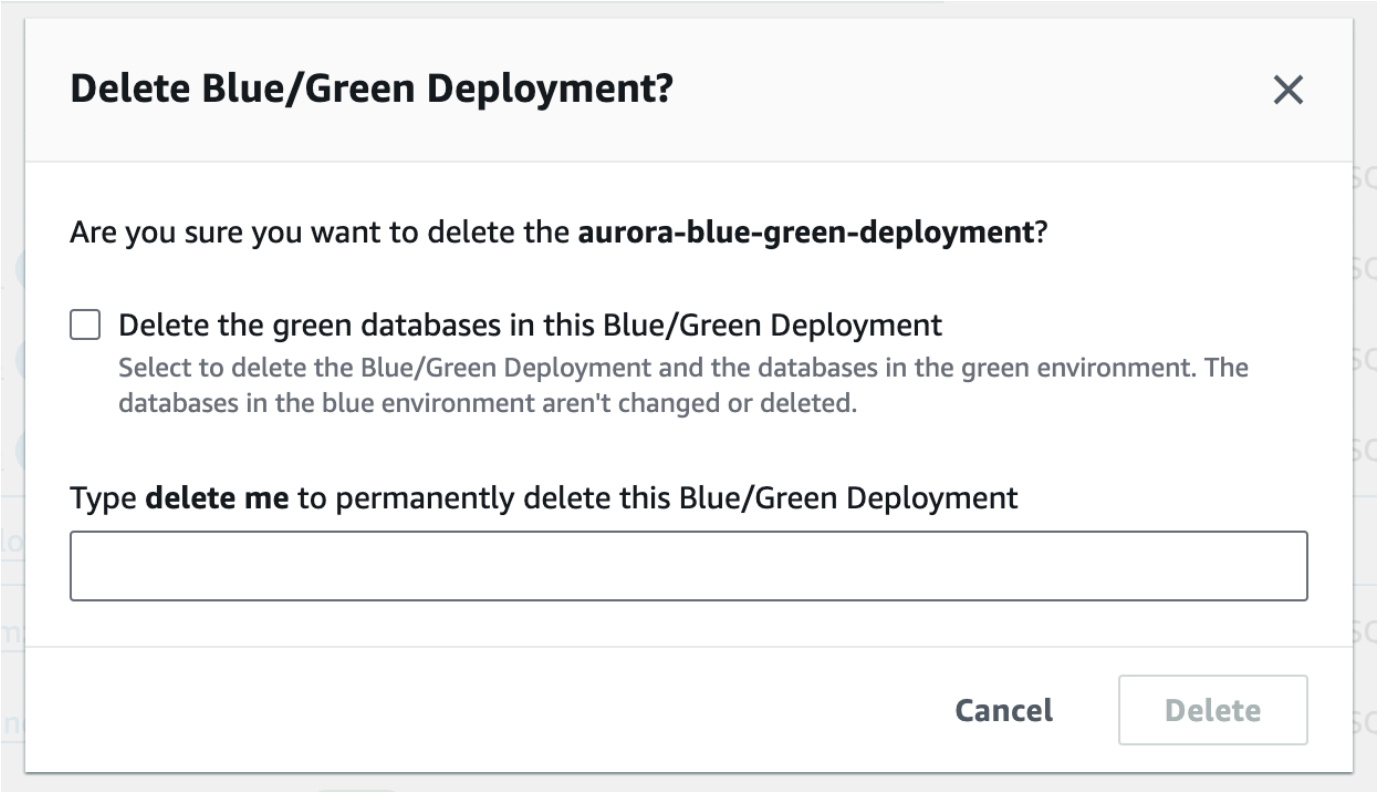
Anda dapat menghapus penerapan biru/hijau menggunakan AWS Management Console, API AWS CLI, atau RDS.

Konsol

Untuk menghapus deployment blue/green

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih deployment blue/green yang ingin dihapus.
3. Untuk Tindakan, pilih Hapus.

   Jendela Hapus Deployment Blue/Green? muncul.



Untuk menghapus basis data hijau, pilih Hapus basis data hijau di Deployment Blue/Green ini.

4. Masukkan **`delete me`** di kotak.
5. Pilih Hapus.

AWS CLI

Untuk menghapus penyebaran biru/hijau dengan menggunakan AWS CLI, gunakan delete-blue-green-deploymentperintah dengan opsi berikut:

- `--blue-green-deployment-identifier`— ID sumber daya dari penyebaran biru/hijau yang akan dihapus.
- `--delete-target` – Menentukan bahwa klaster DB di lingkungan hijau dihapus. Anda tidak dapat menentukan opsi ini jika deployment blue/green memiliki status `SWITCHOVER_COMPLETED`.
- `--no-delete-target` – Menentukan bahwa klaster DB di lingkungan hijau dipertahankan.

Example Menghapus deployment blue/green dan klaster DB di lingkungan hijau

Untuk Linux, macOS, atau Unix:

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --delete-target
```

Untuk Windows:

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --delete-target
```

Example Menghapus deployment blue/green tetapi mempertahankan klaster DB di lingkungan hijau

Untuk Linux, macOS, atau Unix:

```
aws rds delete-blue-green-deployment \
    --blue-green-deployment-identifier bgd-1234567890abcdef \
    --no-delete-target
```

Untuk Windows:

```
aws rds delete-blue-green-deployment ^
    --blue-green-deployment-identifier bgd-1234567890abcdef ^
    --no-delete-target
```

API RDS

Untuk menghapus deployment blue/green menggunakan API Amazon RDS, gunakan operasi `DeleteBlueGreenDeployment` dengan parameter berikut:

- `BlueGreenDeploymentIdentifier`— ID sumber daya dari penyebaran biru/hijau yang akan dihapus.
- `DeleteTarget` – Menentukan `TRUE` untuk menghapus klaster DB di lingkungan hijau atau `FALSE` untuk mempertahankannya. Tidak bisa `TRUE` jika deployment blue/green memiliki status `SWITCHOVER_COMPLETED`.

Mencadangkan dan memulihkan klaster DB Amazon Aurora

Topik ini menyediakan informasi tentang mencadangkan dan memulihkan klaster DB Amazon Aurora.

> **Tip**
>
> Fitur ketersediaan tinggi Aurora dan kapabilitas pencadangan otomatis membantu menjaga data Anda tetap aman tanpa memerlukan pengaturan yang ekstensif dari Anda. Sebelum Anda mengimplementasikan strategi pencadangan, pelajari cara Aurora menyimpan beberapa salinan data Anda dan membantu Anda mengaksesnya di beberapa instans DB dan Wilayah AWS. Lihat perinciannya di Ketersediaan yang tinggi untuk Amazon Aurora.

Topik

- Gambaran umum pencadangan dan pemulihan klaster DB Aurora
- Memahami penggunaan penyimpanan cadangan Amazon Aurora
- Membuat snapshot klaster DB
- Memulihkan dari snapshot klaster DB
- Menyalin snapshot klaster DB
- Berbagi snapshot klaster DB
- Mengekspor data klaster DB ke Amazon S3
- Mengekspor data snapshot klaster DB ke Amazon S3
- Memulihkan klaster DB ke waktu tertentu
- Menghapus snapshot klaster DB
- Tutorial: Memulihkan klaster DB Amazon Aurora dari snapshot klaster DB

Gambaran umum pencadangan dan pemulihan klaster DB Aurora

Topik berikut menjelaskan cadangan Aurora dan cara memulihkan klaster DB Aurora Anda.

Daftar Isi

- Cadangan
  - Menggunakan AWS Backup
- Jendela cadangan
- Mempertahankan cadangan otomatis
  - Periode retensi
  - Melihat cadangan yang dipertahankan
  - Biaya retensi
  - Batasan
  - Menghapus cadangan otomatis yang dipertahankan
- Memulihkan data
- Kloning basis data untuk Aurora
- Backtrack

Cadangan

Aurora mencadangkan volume klaster Anda secara otomatis dan mempertahankan data yang dipulihkan selama periode retensi cadangan. Cadangan otomatis Aurora bersifat kontinu dan inkremental, sehingga Anda dapat memulihkan dengan cepat ke titik mana pun dalam periode retensi cadangan. Tidak ada dampak performa atau gangguan layanan basis data saat data cadangan ditulis. Anda dapat menentukan periode retensi cadangan selama 1–35 hari saat Anda membuat atau memodifikasi klaster DB. Cadangan otomatis disimpan di Amazon S3.

Jika Anda ingin mempertahankan data di luar periode retensi cadangan, Anda dapat mengambil snapshot data di volume klaster Anda. Snapshot klaster DB Aurora tidak kedaluwarsa. Anda dapat membuat klaster DB baru dari snapshot. Untuk informasi selengkapnya, lihat Membuat snapshot klaster DB.

> **Note**
>
> - Untuk klaster DB Amazon Aurora, periode retensi cadangan default adalah satu hari, terlepas dari cara klaster DB dibuat.
> - Anda tidak dapat menonaktifkan cadangan otomatis di Aurora. Periode retensi cadangan untuk Aurora dikelola oleh klaster DB.

Biaya penyimpanan cadangan Anda tergantung pada jumlah cadangan Aurora dan data snapshot yang Anda pertahankan dan berapa lama Anda mempertahankannya. Untuk informasi tentang penyimpanan yang terkait dengan cadangan dan snapshot Aurora, lihat Memahami penggunaan penyimpanan cadangan Amazon Aurora. Untuk informasi harga penyimpanan cadangan Aurora, lihat Harga Amazon RDS for Aurora. Setelah klaster Aurora yang terkait dengan snapshot dihapus, penyimpanan snapshot tersebut akan menimbulkan biaya penyimpanan cadangan standar untuk Aurora.

Menggunakan AWS Backup

Anda dapat menggunakan AWS Backup untuk mengelola cadangan klaster DB Amazon Aurora.

Snapshot yang dikelola oleh AWS Backup dianggap sebagai snapshot klaster DB manual, tetapi tidak diperhitungkan dalam kuota snapshot klaster DB untuk Aurora. Snapshot yang dibuat dengan AWS Backup memiliki nama dengan `awsbackup:job-`*`AWS-Backup-job-number`*. Untuk informasi selengkapnya tentang AWS Backup, lihat Panduan Developer AWS Backup.

Anda juga dapat menggunakan AWS Backup untuk mengelola cadangan otomatis klaster DB Amazon Aurora. Jika cluster DB Anda dikaitkan dengan rencana cadangan diAWS Backup, Anda dapat menggunakan paket cadangan itu untuk point-in-time pemulihan. Pencadangan otomatis (berkelanjutan) yang dikelola oleh AWS Backup memiliki nama dengan `continuous:cluster-`*`AWS-Backup-job-number`*. Untuk informasi selengkapnya, lihat Memulihkan cluster DB ke waktu tertentu menggunakan AWS Backup.

Jendela cadangan

Pencadangan otomatis terjadi setiap hari selama jendela cadangan yang dipilih. Jika pencadangan memerlukan waktu lebih dari yang dialokasikan untuk jendela cadangan, pencadangan akan berlanjut setelah periode berakhir, hingga selesai. Jendela cadangan tidak dapat tumpang tindih dengan jendela pemeliharaan mingguan untuk klaster DB.

Cadangan otomatis Aurora bersifat terus-menerus dan bertahap, namun jendela cadangan digunakan untuk membuat cadangan sistem harian yang dipertahankan dalam periode retensi cadangan. Anda dapat menyalin cadangan untuk mempertahankannya di luar periode retensi.

> **Note**
>
> Ketika Anda membuat klaster DB menggunakan AWS Management Console, Anda tidak dapat menentukan jendela cadangan. Namun, Anda dapat menentukan jendela cadangan ketika Anda membuat klaster DB menggunakan AWS CLI atau API RDS.

Jika Anda tidak menentukan jendela cadangan yang diinginkan saat Anda membuat klaster DB, Aurora akan menetapkan jendela cadangan 30 menit default. Jendela cadangan dipilih secara acak dari blok waktu 8 jam untuk masing-masing Wilayah AWS. Tabel berikut menampilkan daftar blok waktu untuk masing-masing Wilayah AWS tempat periode pencadangan default ditetapkan.

Nama Wilayah	Wilayah	Blok Waktu
AS Timur (Ohio)	us-east-2	03:00–11:00 UTC
AS Timur (Virginia Utara)	us-east-1	03:00–11:00 UTC
AS Barat (California Utara)	us-west-1	06.00–14.00 UTC
AS Barat (Oregon)	us-west-2	06.00–14.00 UTC
Afrika (Cape Town)	af-south-1	03:00–11:00 UTC
Asia Pasifik (Hong Kong)	ap-east-1	06.00–14.00 UTC
Asia Pasifik (Hyderabad)	ap-south-2	06.30–14.30 UTC
Asia Pasifik (Jakarta)	ap-southeast-3	08.00–16.00 UTC

Nama Wilayah	Wilayah	Blok Waktu
Asia Pasifik (Melbourne)	ap-southeast-4	11.00–19.00 UTC
Asia Pasifik (Mumbai)	ap-south-1	16.30–00.30 UTC
Asia Pasifik (Osaka)	ap-northeast-3	00.00–08.00 UTC
Asia Pasifik (Seoul)	ap-northeast-2	13.00–21.00 UTC
Asia Pasifik (Singapura)	ap-southeast-1	14.00–22.00 UTC
Asia Pasifik (Sydney)	ap-southeast-2	12.00–20.00 UTC
Asia Pasifik (Tokyo)	ap-northeast-1	13.00–21.00 UTC
Kanada (Pusat)	ca-central-1	03:00–11:00 UTC
Kanada Barat (Calgary)	ca-west-1	18:00 — 02:00 UTC
Tiongkok (Beijing)	cn-north-1	06.00–14.00 UTC
Tiongkok (Ningxia)	cn-northwest-1	06.00–14.00 UTC
Eropa (Frankfurt)	eu-central-1	20.00–04.00 UTC
Eropa (Irlandia)	eu-west-1	22.00–06.00 UTC
Eropa (London)	eu-west-2	22.00–06.00 UTC
Eropa (Milan)	eu-south-1	02.00–10.00 UTC
Eropa (Paris)	eu-west-3	07.29–14.29 UTC
Eropa (Spanyol)	eu-south-2	02.00–10.00 UTC
Eropa (Stockholm)	eu-north-1	23.00–07.00 UTC
Eropa (Zurich)	eu-central-2	02.00–10.00 UTC

Nama Wilayah	Wilayah	Blok Waktu
Israel (Tel Aviv)	il-central-1	03:00–11:00 UTC
Timur Tengah (Bahrain)	me-south-1	06.00–14.00 UTC
Timur Tengah (UEA)	me-central-1	05.00–13.00 UTC
Amerika Selatan (Sao Paulo)	sa-east-1	23.00–07.00 UTC
AWS GovCloud (AS-Timur)	us-gov-east-1	17.00–01.00 UTC
AWS GovCloud (AS-Barat)	us-gov-west-1	06.00–14.00 UTC

Mempertahankan cadangan otomatis

Saat menghapus klaster DB Aurora Serverless v2 terprovisi, Anda dapat mempertahankan cadangan otomatis. Hal ini memungkinkan Anda memulihkan klaster DB ke titik waktu tertentu dalam periode retensi cadangan, bahkan setelah klaster dihapus.

Cadangan otomatis yang dipertahankan berisi snapshot sistem dan log transaksi dari klaster DB. Cadangan ini juga mencakup properti klaster DB, seperti kelas instans DB, yang diperlukan untuk memulihkannya ke klaster aktif.

Anda dapat memulihkan atau menghapus cadangan otomatis yang dipertahankan menggunakan AWS Management Console, API RDS, dan AWS CLI.

> **Note**
>
> Anda tidak dapat mempertahankan cadangan otomatis untuk klaster DB Aurora Serverless v1.

Topik

- Periode retensi

- Melihat cadangan yang dipertahankan
- Biaya retensi
- Batasan
- Menghapus cadangan otomatis yang dipertahankan

Periode retensi

Snapshot sistem dan log transaksi dalam cadangan otomatis yang dipertahankan akan kedaluwarsa dengan cara yang sama seperti untuk klaster DB sumber. Pengaturan untuk periode retensi klaster sumber juga berlaku untuk cadangan otomatis. Karena tidak ada snapshot atau log baru yang dibuat untuk klaster ini, cadangan otomatis yang dipertahankan pada akhirnya kedaluwarsa sepenuhnya. Setelah periode retensi selesai, Anda terus mempertahankan snapshot klaster DB manual, tetapi semua cadangan otomatis akan kedaluwarsa.

Anda dapat menghapus cadangan otomatis yang dipertahankan menggunakan konsol, AWS CLI, atau API RDS. Untuk informasi selengkapnya, lihat Menghapus cadangan otomatis yang dipertahankan.

Tidak seperti cadangan otomatis yang dipertahankan, snapshot akhir tidak kedaluwarsa. Kami sangat menyarankan agar Anda mengambil snapshot akhir bahkan jika Anda mempertahankan cadangan otomatis karena cadangan otomatis yang dipertahankan pada akhirnya akan kedaluwarsa.

Melihat cadangan yang dipertahankan

Untuk melihat cadangan otomatis yang dipertahankan di konsol RDS, pilih Pencadangan otomatis di panel navigasi, lalu pilih Dipertahankan. Untuk melihat snapshot individual yang terkait dengan cadangan otomatis yang dipertahankan, pilih Snapshot di panel navigasi. Alternatifnya, Anda dapat mendeskripsikan snapshot individual yang terkait dengan cadangan otomatis yang dipertahankan. Dari sana, Anda dapat memulihkan instans DB langsung dari salah satu snapshot tersebut.

Untuk mendeskripsikan cadangan otomatis yang dipertahankan menggunakanAWS CLI, gunakan perintah berikut:

```
aws rds describe-db-cluster-automated-backups --db-cluster-resource-
id DB_cluster_resource_ID
```

Untuk mendeskripsikan cadangan otomatis yang dipertahankan menggunakan API RDS, panggil tindakan `DescribeDBClusterAutomatedBackups` dengan parameter `DbClusterResourceId`.

Biaya retensi

Tidak ada biaya tambahan untuk penyimpanan cadangan hingga 100% dari total penyimpanan basis data Aurora Anda untuk setiap klaster DB Aurora. Selain itu, tidak ada biaya tambahan hingga satu hari ketika Anda mempertahankan cadangan otomatis setelah menghapus klaster DB. Cadangan yang Anda pertahankan selama lebih dari satu hari akan dikenai biaya.

Tidak ada biaya tambahan untuk log transaksi atau metadata instans. Semua aturan penetapan harga lainnya untuk cadangan berlaku untuk klaster yang dapat dipulihkan. Untuk informasi selengkapnya, lihat halaman Harga Amazon Aurora.

Batasan

Batasan berikut berlaku untuk cadangan otomatis yang dipertahankan:

- Jumlah maksimum cadangan otomatis yang dipertahankan di satu Wilayah AWS adalah 40. Ini tidak termasuk dalam kuota untuk klaster DB. Anda dapat memiliki hingga 40 klaster DB yang berjalan, 40 instans DB yang berjalan, dan 40 cadangan otomatis yang dipertahankan untuk klaster DB secara bersamaan.

  Untuk informasi selengkapnya, lihat Kuota dalam Amazon Aurora.
- Cadangan otomatis yang dipertahankan tidak berisi informasi tentang parameter atau grup opsi.
- Anda dapat memulihkan klaster yang dihapus ke titik waktu yang berada dalam periode retensi pada saat penghapusan.
- Anda tidak dapat memodifikasi cadangan otomatis yang dipertahankan karena cadangan ini terdiri dari cadangan sistem, log transaksi, dan properti klaster DB yang ada saat Anda menghapus klaster sumber.

Menghapus cadangan otomatis yang dipertahankan

Anda dapat menghapus cadangan otomatis yang dipertahankan ketika tidak diperlukan lagi.

Konsol

Untuk menghapus cadangan otomatis yang dipertahankan

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

2. Di panel navigasi, pilih Pencadangan otomatis.
3. Pilih tab Dipertahankan.

4. Pilih cadangan otomatis yang dipertahankan yang ingin Anda hapus.
5. Untuk Tindakan, pilih Hapus.
6. Di halaman konfirmasi, masukkan **`delete me`** dan pilih Hapus.

AWS CLI

Anda dapat menghapus cadangan otomatis yang dipertahankan dengan menggunakan AWS CLI perintah delete-db-cluster-automated-backup dengan opsi berikut:

- `--db-cluster-resource-id` – Pengidentifikasi sumber daya untuk klaster DB sumber.

  Anda dapat menemukan pengenal sumber daya untuk cluster DB sumber dari cadangan otomatis yang dipertahankan dengan menjalankan AWS CLI perintah describe-db-cluster-automated - backup.

Example

Contoh ini menghapus cadangan otomatis yang dipertahankan untuk klaster DB sumber yang memiliki ID sumber daya `cluster-123ABCEXAMPLE`.

Untuk Linux, macOS, atau Unix:

```
aws rds delete-db-cluster-automated-backup \
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

Untuk Windows:

```
aws rds delete-db-cluster-automated-backup ^
    --db-cluster-resource-id cluster-123ABCEXAMPLE
```

API RDS

Anda dapat menghapus cadangan otomatis yang dipertahankan dengan menggunakan operasi Amazon RDS API DeleteDB ClusterAutomatedBackup dengan parameter berikut:

- `DbClusterResourceId` – Pengidentifikasi sumber daya untuk klaster DB sumber.

  Anda dapat menemukan pengenal sumber daya untuk instans DB sumber cadangan otomatis yang dipertahankan menggunakan operasi Amazon RDS API DescribeDB. ClusterAutomatedBackups

Memulihkan data

Anda dapat memulihkan data Anda dengan membuat klaster DB Aurora baru dari data cadangan yang dipertahankan Aurora, dari snapshot klaster DB yang telah Anda simpan, atau dari cadangan otomatis yang dipertahankan. Anda dapat dengan cepat memulihkan salinan baru klaster DB yang dibuat dari data cadangan ke titik waktu mana pun selama periode retensi cadangan Anda. Karena cadangan Aurora bersifat kontinu dan inkremental selama periode retensi cadangan, Anda tidak perlu sering mengambil snapshot data untuk meningkatkan waktu pemulihan.

Waktu pemulihan terbaru untuk klaster DB adalah titik terbaru yang dapat Anda untuk memulihkan klaster DB Anda. Titik ini biasanya 5 menit dari waktu saat ini untuk klaster DB aktif, atau 5 menit dari waktu penghapusan klaster untuk cadangan otomatis yang dipertahankan.

Opsi Waktu pemulihan paling awal menentukan seberapa jauh ke belakang dalam periode retensi cadangan yang dapat Anda tentukan untuk memulihkan volume klaster Anda.

Untuk menentukan waktu pemulihan terbaru atau paling awal untuk klaster DB, cari `Latest restorable time` atau `Earliest restorable time` pada konsol RDS. Untuk informasi tentang melihat nilai-nilai ini, lihat Melihat cadangan yang dipertahankan.

Anda dapat menentukan kapan pemulihan klaster DB selesai dengan memeriksa nilai `Latest restorable time` dan `Earliest restorable time`. Nilai-nilai ini menampilkan NULL sampai operasi pemulihan selesai. Anda tidak dapat meminta operasi pencadangan atau pemulihan jika `Latest restorable time` atau `Earliest restorable time` menampilkan NULL.

Untuk informasi tentang memulihkan klaster DB ke waktu yang ditentukan, lihat Memulihkan klaster DB ke waktu tertentu.

Kloning basis data untuk Aurora

Anda juga dapat menggunakan kloning basis data untuk mengkloning basis data klaster DB Aurora Anda ke klaster DB baru, alih-alih memulihkan snapshot klaster DB. Basis data klon hanya menggunakan ruang tambahan yang minimal saat pertama kali dibuat. Data disalin hanya seiring perubahannya, misalnya pada basis data sumber atau basis data klon. Anda dapat membuat beberapa klon dari klaster DB yang sama, atau membuat klon tambahan bahkan dari klon lain. Untuk informasi selengkapnya, lihat Mengkloning volume untuk klaster DB Amazon Aurora.

Backtrack

Aurora MySQL sekarang mendukung tindakan "memundurkan" klaster DB ke waktu tertentu, tanpa memulihkan data dari cadangan. Untuk informasi selengkapnya, lihat Melakukan backtracking klaster DB Aurora.

Memahami penggunaan penyimpanan cadangan Amazon Aurora

Amazon Aurora memelihara dua jenis cadangan: cadangan otomatis (berkelanjutan) dan snapshot.

Penyimpanan cadangan otomatis

Cadangan otomatis (berkelanjutan) untuk klaster secara inkremental menyimpan semua perubahan basis data dalam periode retensi tertentu agar dapat memulihkan ke titik waktu mana pun dalam periode retensi tersebut. Periode retensi dapat berkisar 1–35 hari. Cadangan otomatis bersifat inkremental dan ditagih berdasarkan jumlah penyimpanan yang diperlukan untuk memulihkan ke waktu mana pun dalam periode retensi.

Aurora juga menyediakan jumlah penggunaan cadangan gratis. Jumlah penggunaan gratis ini sama dengan ukuran volume klaster terbaru (seperti yang ditunjukkan oleh metrik `VolumeBytesUsed` Amazon CloudWatch). Jumlah ini dikurangi dari penggunaan cadangan otomatis yang dihitung. Selan itu, tidak ada biaya untuk cadangan otomatis yang periode retensinya hanya 1 hari.

Misalnya, cadangan otomatis Anda memiliki periode retensi 7 hari, dan Anda ingin memulihkan klaster Anda ke statusnya dari empat hari yang lalu. Aurora menggunakan data inkremental yang tersimpan dalam cadangan otomatis untuk membuat kembali status klaster pada waktu yang persis empat hari lalu.

Cadangan otomatis menyimpan semua informasi yang diperlukan agar dapat memulihkan klaster ke waktu mana pun dalam periode retensi. Itu berarti bahwa cadangan otomatis menyimpan semua perubahan selama periode retensi, termasuk penulisan informasi baru atau penghapusan informasi yang ada. Untuk basis data yang memiliki banyak perubahan yang terjadi, ukuran cadangan otomatis akan tumbuh dari waktu ke waktu. Setelah basis data berhenti mengalami perubahan, Anda dapat mendapati bahwa ukuran cadangan otomatis berkurang karena perubahan tersimpan sebelumnya berada di luar periode retensi.

Total penggunaan yang ditagih untuk cadangan otomatis tidak pernah melebihi ukuran volume klaster kumulatif selama periode retensi. Misalnya, jika periode retensi Anda adalah 7 hari, dan volume klaster Anda 100 GB setiap hari, maka penggunaan cadangan otomatis yang ditagih tidak pernah melebihi 700 GB (100 GB * 7).

Penyimpanan snapshot

Snapshot klaster DB selalu merupakan cadangan penuh yang ukurannya adalah volume klaster pada saat snapshot diambil. Snapshot, baik diambil secara manual oleh pengguna maupun secara

otomatis oleh rencana AWS Backup, diperlakukan sebagai snapshot manual. Aurora menyediakan penyimpanan gratis tanpa batas untuk semua snapshot yang berada dalam periode retensi cadangan otomatis. Setelah berada di luar periode retensi, snapshot manual ditagih per GB-bulan. Setiap snapshot sistem otomatis tidak pernah dikenai biaya kecuali jika disalin dan dipertahankan melewati periode retensi.

Untuk informasi umum tentang cadangan Aurora, lihat Cadangan. Untuk informasi harga penyimpanan cadangan Aurora, lihat halaman harga Amazon Aurora.

Metrik Amazon CloudWatch untuk penyimpanan cadangan Aurora

Anda dapat memantau klaster Aurora dan membuat laporan menggunakan metrik Amazon CloudWatch melalui konsol CloudWatch. Anda dapat menggunakan metrik CloudWatch berikut untuk meninjau dan memantau jumlah penyimpanan yang digunakan oleh cadangan Aurora Anda. Metrik ini dihitung secara independen untuk setiap klaster Aurora DB.

- `BackupRetentionPeriodStorageUsed` – Merupakan jumlah penyimpanan cadangan yang digunakan, dalam byte, untuk menyimpan cadangan otomatis pada saat ini.
  - Nilainya tergantung pada ukuran volume klaster dan jumlah perubahan (penulisan dan pembaruan) yang dibuat untuk klaster DB selama periode retensi. Hal ini karena cadangan otomatis harus menyimpan semua perubahan inkremental yang dilakukan pada klaster agar dapat memulihkan ke titik waktu mana pun.
  - Metrik ini tidak mengurangi tingkat gratis penggunaan cadangan yang disediakan Aurora.
  - Metrik ini menghasilkan satu titik data harian untuk penggunaan cadangan otomatis yang dicatat pada hari itu.
- `SnapshotStorageUsed` – Merupakan jumlah penyimpanan cadangan yang digunakan, dalam byte, untuk menyimpan snapshot manual di luar periode retensi cadangan otomatis.
  - Nilainya tergantung pada jumlah snapshot yang dipertahankan di luar periode retensi cadangan otomatis dan ukuran masing-masing snapshot.
  - Ukuran masing-masing snapshot adalah ukuran volume klaster pada saat Anda mengambil snapshot.
  - Snapshot adalah cadangan penuh, bukan inkremental.
  - Metrik ini menghasilkan satu titik data harian untuk setiap snapshot yang diisi. Untuk mengetahui total penggunaan snapshot harian Anda, gunakan jumlah metrik ini selama 1 hari.
- `TotalBackupStorageBilled` – Merupakan metrik untuk semua penggunaan cadangan yang ditagih, dalam byte, untuk klaster yang ditentukan:

`BackupRetentionPeriodStorageUsed + SnapshotStorageUsed - free tier`

- Metrik ini menghasilkan satu titik data harian untuk nilai `BackupRetentionPeriodStorageUsed` dikurangi tingkat gratis penggunaan cadangan yang disediakan Aurora. Tingkat gratis ini sama dengan ukuran volume klaster DB yang terakhir dicatat. Titik data ini merepresentasikan penggunaan sebenarnya yang ditagih untuk cadangan otomatis.
- Metrik ini menghasilkan titik data harian individu untuk semua nilai `SnapshotStorageUsed`.
- Untuk mengetahui total penggunaan cadangan harian yang ditagih, gunakan jumlah metrik ini selama 1 hari. Metrik ini menjumlahkan semua penggunaan snapshot yang ditagih dengan penggunaan cadangan otomatis yang ditagih, untuk menghasilkan total penggunaan cadangan yang ditagih.

Untuk informasi selengkapnya tentang cara menggunakan metrik CloudWatch, lihat Ketersediaan metrik Aurora di konsol Amazon RDS.

Menghitung penggunaan penyimpanan cadangan

Penggunaan untuk cadangan otomatis dihitung dengan melihat semua catatan inkremental yang harus disimpan agar dapat memulihkan ke titik waktu mana pun dalam periode retensi cadangan.

Misalnya, Anda memiliki cadangan otomatis dengan periode retensi 7 hari. Ukuran volume klaster Anda tepat sebelum periode retensi adalah 100 GB. Dengan demikian, itulah jumlah paling sedikit yang perlu disimpan Aurora. Kemudian, Anda memiliki aktivitas di bawah selama 7 hari ke depan, dengan ukuran catatan inkremental yang merupakan jumlah penyimpanan yang diperlukan untuk menyimpan catatan perubahan yang berasal dari penulisan dan pembaruan basis data Anda.

Hari	Ukuran catatan inkremental (GB)
1	10
2	15
3	25
4	20
5	10

Hari	Ukuran catatan inkremental (GB)
6	25
7	30
Total	135

Data ini menunjukkan bahwa penggunaan cadangan otomatis yang dihitung untuk cadangan Anda adalah sebagai berikut:

```
100 GB (volume size before retention period) + 135 GB (size of incremental records) =
 235 GB total backup usage
```

Kemudian, kurangi tingkat penggunaan gratis dengan penggunaan yang ditagih. Asumsikan bahwa ukuran terbaru volume Anda adalah 200 GB:

```
235 GB total backup usage - 200 GB (latest volume size) = 35 GB billed backup usage
```

FAQ

Kapan saya ditagih untuk snapshot?

Anda ditagih untuk snapshot manual yang berada di luar (lebih lama dari) periode retensi cadangan otomatis.

Apa itu snapshot manual?

Snapshot manual adalah snapshot yang memenuhi ketentuan berikut ini:

- Diminta secara manual oleh Anda
- Diambil oleh layanan cadangan otomatis seperti AWS Backup
- Disalin dari snapshot sistem otomatis untuk mempertahankannya di luar periode retensi

Apa yang terjadi pada snapshot manual saya jika saya menghapus klaster DB saya?

Snapshot manual tidak kedaluwarsa sampai Anda menghapusnya.

Saat Anda menghapus klaster DB Anda, snapshot manual yang sebelumnya Anda ambil terus ada. Jika snapshot ini sebelumnya tidak ditagih karena mereka berada dalam periode retensi

cadangan otomatis, sekarang snapshot tersebut tidak tercakup lagi dan semuanya akan mulai ditagih menurut ukuran penuh untuk penggunaannya.

Bagaimana cara mengurangi biaya penyimpanan cadangan saya?

Ada beberapa cara untuk mengurangi biaya terkait penggunaan cadangan:

- Hapus snapshot manual yang berada di luar periode penyimpanan cadangan otomatis Anda. Hal ini termasuk snapshot yang Anda ambil, serta snapshot yang mungkin diambil oleh rencana AWS Backup Anda. Pastikan untuk memeriksa rencana AWS Backup untuk memastikan layanan ini tidak menyimpan snapshot di luar periode retensi yang tidak Anda harapkan.
- Evaluasi penulisan dan pembaruan ke basis data Anda untuk melihat apakah Anda dapat mengurangi jumlah perubahan yang Anda buat. Karena cadangan otomatis kami menyimpan semua perubahan inkremental dalam periode retensi, mengurangi jumlah pembaruan yang Anda buat juga mengurangi biaya cadangan otomatis Anda.
- Evaluasi apakah mengurangi periode retensi cadangan otomatis Anda layak untuk dilakukan. Mengurangi periode retensi berarti bahwa cadangan menyimpan data inkremental dalam lebih sedikit hari, yang dapat mengurangi biaya cadangan secara keseluruhan. Namun, mengurangi periode retensi ini juga dapat menyebabkan beberapa snapshot mulai ditagih karena snapshot tersebut sekarang berada di luar periode retensi. Pastikan untuk memeriksa semua biaya snapshot tambahan yang mungkin Anda keluarkan sebelum memutuskan apakah ini tindakan yang tepat untuk Anda.

Bagaimana penyimpanan cadangan ditagih?

Penyimpanan cadangan ditagih berdasarkan GB-bulan.

Hal ini berarti bahwa penggunaan penyimpanan cadangan ditagih sebagai penggunaan rata-rata berbobot selama bulan tertentu. Berikut adalah beberapa contoh untuk bulan 30 hari:

- Penggunaan cadangan yang ditagih adalah 100 GB untuk total 30 hari dalam sebulan. Biaya Anda adalah sebagai berikut:

```
(100 GB * 30) / 30 = 100 GB-month
```

- Penggunaan cadangan yang ditagih adalah 100 GB untuk 15 hari pertama dalam sebulan, kemudian 0 GB untuk 15 terakhir. Biaya Anda adalah sebagai berikut:

```
(100 GB * 15 + 0 GB * 15) / 30 = 50 GB-month
```

- Penggunaan cadangan yang ditagih adalah 50 GB untuk 10 hari pertama dalam sebulan, 100 GB untuk 10 hari ke depan, kemudian 150 GB untuk 10 hari terakhir. Biaya Anda adalah sebagai berikut:

  ```
  (50 GB * 10 + 100 GB * 10 + 150 GB * 10) / 30 = 100 GB-month
  ```

Bagaimana pengaturan pelacakan mundur untuk klaster DB saya memengaruhi penggunaan penyimpanan cadangan?

Pengaturan pelacakan mundur untuk klaster Aurora DB tidak memengaruhi volume data cadangan untuk klaster tersebut. Amazon menagih penyimpanan untuk pelacakan mundur data secara terpisah. Untuk informasi harga pelacakan mundur Aurora, lihat halaman harga Amazon Aurora.

Bagaimana biaya penyimpanan berlaku untuk snapshot yang dibagikan?

Jika Anda berbagi snapshot dengan pengguna lain, Anda masih menjadi pemilik snapshot tersebut. Biaya penyimpanan berlaku untuk pemilik snapshot. Jika Anda menghapus snapshot yang dibagikan yang Anda miliki, tidak ada yang dapat mengaksesnya.

Untuk mempertahankan akses ke snapshot yang dibagikan yang dimiliki orang lain, Anda dapat menyalin snapshot tersebut. Dengan begitu, Anda menjadi pemilik snapshot baru. Setiap biaya penyimpanan untuk snapshot yang disalin berlaku untuk akun Anda.

Untuk informasi selengkapnya tentang berbagi snapshot, lihat Berbagi snapshot klaster DB. Untuk informasi selengkapnya tentang menyalin snapshot, lihat Menyalin snapshot klaster DB.

Membuat snapshot klaster DB

Amazon RDS membuat snapshot volume penyimpanan klaster DB Anda, mencadangkan seluruh klaster DB, bukan hanya basis data individu. Saat Anda membuat snapshot klaster DB, Anda perlu mengidentifikasi klaster DB yang akan dicadangkan, dan kemudian memberikan nama snapshot klaster DB Anda agar Anda dapat memulihkannya di lain waktu. Jumlah waktu yang diperlukan untuk membuat snapshot klaster DB bervariasi sesuai ukuran basis data Anda. Karena snapshot menyertakan seluruh volume penyimpanan, ukuran file, seperti file sementara, juga memengaruhi jumlah waktu yang diperlukan untuk membuat snapshot.

> **Note**
>
> Klaster DB Anda harus dalam status `available` untuk mengambil snapshot klaster DB.

Tidak seperti pencadangan otomatis, snapshot manual tidak bergantung pada periode retensi pencadangan. Snapshots tidak kedaluwarsa.

Untuk cadangan jangka sangat panjang, sebaiknya Anda mengekspor data snapshot ke Amazon S3. Jika versi utama mesin DB Anda tidak didukung lagi, Anda tidak dapat memulihkan ke versi tersebut dari snapshot. Untuk informasi selengkapnya, lihat Mengekspor data snapshot klaster DB ke Amazon S3.

Anda dapat membuat snapshot cluster DB menggunakan AWS Management Console API, AWS CLI, atau RDS.

Konsol

Untuk membuat snapshot klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.

   Daftar Snapshot manual akan muncul.
3. Pilih Ambil snapshot.

   Jendela Ambil snapshot DB akan muncul.
4. Untuk jenis Snapshot, pilih DB cluster.

5. Pilih cluster DB yang ingin Anda ambil snapshot.
6. Masukkan nama Snapshot.
7. Pilih Ambil snapshot.

   Daftar snapshot Manual muncul, dengan status snapshot cluster DB baru ditampilkan sebagai. `Creating` Setelah statusnya adalah `Available`, Anda dapat melihat waktu pembuatannya.

AWS CLI

Saat Anda membuat snapshot cluster DB menggunakan AWS CLI, Anda perlu mengidentifikasi cluster DB mana yang akan Anda cadangkan, dan kemudian beri nama snapshot cluster DB Anda sehingga Anda dapat memulihkannya nanti. Anda dapat melakukan ini dengan menggunakan AWS CLI `create-db-cluster-snapshot`perintah dengan parameter berikut:

- `--db-cluster-identifier`
- `--db-cluster-snapshot-identifier`

Dalam contoh ini, Anda membuat snapshot klaster DB yang bernama *`mydbclustersnapshot`* untuk klaster DB yang disebut *`mydbcluster`*.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster-snapshot \
    --db-cluster-identifier mydbcluster \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

Untuk Windows:

```
aws rds create-db-cluster-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

RDS API

Saat membuat snapshot klaster DB menggunakan Amazon RDS API, Anda perlu mengidentifikasi klaster DB yang akan dicadangkan, lalu memberi nama snapshot klaster DB Anda sehingga Anda

dapat memulihkannya di lain waktu. Anda dapat melakukannya dengan menggunakan perintah Amazon RDS API `CreateDBClusterSnapshot` dengan parameter berikut:

- DB ClusterIdentifier
- DB ClusterSnapshotIdentifier

Menentukan apakah snapshot klaster DB tersedia

Anda dapat memeriksa apakah snapshot cluster DB tersedia dengan melihat di bawah Snapshots pada tab Maintenance & backup pada halaman detail untuk cluster di AWS Management Console, dengan menggunakan perintah `describe-db-cluster-snapshots`CLI, atau dengan menggunakan tindakan API. `DescribeDBClusterSnapshots`

Anda juga dapat menggunakan perintah `wait db-cluster-snapshot-available` CLI untuk melakukan polling API setiap 30 detik hingga snapshot tersedia.

Memulihkan dari snapshot klaster DB

Amazon RDS membuat snapshot volume penyimpanan instans basis data Anda, sehingga mencadangkan seluruh klaster DB dan bukan hanya masing-masing basis data. Anda dapat membuat klaster DB baru dengan memulihkannya dari snapshot DB. Anda memberikan nama snapshot klaster DB untuk memulihkannya, kemudian memberikan nama klaster DB baru yang dibuat dari pemulihan tersebut. Anda tidak dapat memulihkan dari snapshot klaster DB ke klaster DB yang sudah ada; klaster DB baru dibuat saat Anda memulihkan.

Anda dapat menggunakan klaster DB yang dipulihkan segera setelah statusnya `available`.

Anda dapat menggunakan AWS CloudFormation untuk memulihkan cluster DB dari snapshot cluster DB. Untuk informasi selengkapnya, lihat AWS::RDS::DBCluster dalam Panduan Pengguna AWS CloudFormation .

> **Note**
>
> Berbagi snapshot cluster DB manual, baik terenkripsi atau tidak terenkripsi, memungkinkan AWS akun resmi untuk secara langsung memulihkan cluster DB dari snapshot alih-alih mengambil salinannya dan memulihkannya. Untuk informasi selengkapnya, lihat Berbagi snapshot klaster DB.

Untuk informasi tentang memulihkan klaster Aurora DB atau klaster global dengan versi RDS Extended Support, lihat. Aurora DB atau cluster global dengan Amazon RDS Extended Support

Pertimbangan grup parameter

Kami sarankan agar Anda mempertahankan grup parameter DB dan grup parameter klaster DB untuk snapshot klaster DB yang Anda buat, sehingga Anda dapat menautkan klaster DB yang dipulihkan dengan grup parameter yang benar.

Grup parameter DB dan grup parameter klaster DB default dikaitkan dengan klaster yang dipulihkan, kecuali jika Anda memilih yang berbeda. Tidak ada pengaturan parameter kustom yang tersedia di grup parameter default.

Anda dapat menentukan grup parameter saat memulihkan klaster DB.

Untuk informasi selengkapnya tentang grup parameter DB dan grup parameter klaster DB, lihat Bekerja dengan grup parameter.

Pertimbangan grup keamanan

Saat Anda memulihkan klaster DB, cloud privat virtual (VPC) default, grup subnet DB, dan grup keamanan VPC akan dikaitkan dengan instans yang dipulihkan, kecuali jika Anda memilih yang berbeda.

- Jika Anda menggunakan konsol Amazon RDS, Anda dapat menentukan grup keamanan VPC kustom untuk ditautkan dengan klaster atau membuat grup keamanan VPC baru.
- Jika Anda menggunakan AWS CLI, Anda dapat menentukan grup keamanan VPC kustom untuk diasosiasikan dengan cluster dengan menyertakan `--vpc-security-group-ids` opsi dalam perintah. `restore-db-cluster-from-snapshot`
- Jika Anda menggunakan API Amazon RDS, Anda dapat menyertakan `VpcSecurityGroupIds.VpcSecurityGroupId.N` di dalam tindakan `RestoreDBClusterFromSnapshot`.

Segera setelah pemulihan selesai dan klaster DB baru Anda tersedia, Anda juga dapat mengubah pengaturan VPC dengan memodifikasi klaster DB. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora.

Pertimbangan Amazon Aurora

Dengan Aurora, Anda dapat memulihkan snapshot klaster DB ke klaster DB.

Dengan Aurora MySQL dan Aurora PostgreSQL, Anda juga dapat memulihkan snapshot klaster DB ke klaster DB Aurora Serverless. Untuk informasi selengkapnya, lihat Memulihkan klaster DB Aurora Serverless v1.

Dengan Aurora MySQL, Anda dapat memulihkan snapshot klaster DB dari klaster tanpa kueri paralel ke klaster dengan kueri paralel. Karena kueri paralel biasanya digunakan dengan tabel yang sangat besar, mekanisme snapshot adalah cara tercepat untuk menyerap data dalam volume besar ke klaster Aurora MySQL yang memiliki kueri paralel aktif. Untuk informasi selengkapnya, lihat Bekerja dengan kueri paralel untuk Amazon Aurora MySQL.

Memulihkan dari snapshot

Anda dapat memulihkan klaster DB dari snapshot klaster DB menggunakan AWS Management Console, AWS CLI, atau API RDS.

Konsol

Untuk memulihkan klaster DB dari snapshot klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih snapshot klaster DB untuk memulihkan klaster.
4. Untuk Tindakan, pilih Pulihkan snapshot.

   Halaman Pulihkan snapshot ditampilkan.
5. Pilih versi mesin DB yang Anda inginkan untuk memulihkan klaster DB.

   Secara default, snapshot dipulihkan ke versi mesin DB yang sama dengan klaster DB sumber, jika versi tersebut tersedia.
6. Untuk Pengidentifikasi instans DB, masukkan nama klaster DB Anda yang dipulihkan.
7. Tentukan pengaturan lain, seperti konfigurasi penyimpanan klaster DB.

   Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.
8. Pilih Pulihkan klaster DB.

AWS CLI

Untuk mengembalikan cluster DB dari snapshot cluster DB, gunakan AWS CLI perintah restore-db-cluster-from-snapshot.

Dalam contoh ini, Anda memulihkan dari snapshot klaster DB yang dibuat sebelumnya yang bernama `mydbclustersnapshot`. Anda memulihkan ke klaster DB baru yang bernama `mynewdbcluster`.

Anda dapat menentukan pengaturan lain, seperti versi mesin DB. Jika Anda tidak menentukan versi mesin, klaster DB dipulihkan ke versi mesin default.

Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine aurora-mysql|aurora-postgresql
```

Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mynewdbcluster ^
    --snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql|aurora-postgresql
```

Setelah klaster DB dipulihkan, Anda harus menambahkannya ke grup keamanan yang digunakan oleh klaster DB ini untuk membuat snapshot klaster DB jika Anda menginginkan fungsi yang sama dengan yang digunakan pada klaster DB sebelumnya.

> **Important**
>
> Jika Anda menggunakan konsol untuk memulihkan klaster DB, maka Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan AWS CLI untuk memulihkan klaster DB, Anda harus secara eksplisit membuat instans primer untuk klaster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB. Jika Anda tidak membuat instans DB primer, titik akhir klaster DB tetap dalam status `creating`.
>
> Panggil create-db-instance AWS CLI perintah untuk membuat instance utama untuk cluster DB Anda. Sertakan nama klaster DB sebagai `--db-cluster-identifier` nilai pilihan.

API RDS

Untuk memulihkan cluster DB dari snapshot cluster DB, panggil operasi RDS API RestoreDB ClusterFromSnapshot dengan parameter berikut:

- `DBClusterIdentifier`
- `SnapshotIdentifier`

Important

Jika Anda menggunakan konsol untuk memulihkan klaster DB, maka Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan API RDS untuk memulihkan klaster DB, Anda harus secara eksplisit membuat instans primer untuk klaster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB. Jika Anda tidak membuat instans DB primer, titik akhir klaster DB tetap dalam status `creating`.

Panggil operasi API RDS CreateDBInstance untuk membuat instans primer untuk klaster DB Anda. Sertakan nama klaster DB sebagai nilai parameter `DBClusterIdentifier`.

Menyalin snapshot klaster DB

Dengan Amazon Aurora, Anda dapat menyalin cadangan otomatis atau snapshot klaster DB manual. Setelah Anda menyalin snapshot, salinannya disebut snapshot manual. Anda dapat membuat beberapa salinan cadangan otomatis atau snapshot manual, tetapi setiap salinan harus memiliki pengidentifikasi unik.

Anda dapat menyalin snapshot dalam Wilayah AWS yang sama dan di seluruh Wilayah AWS. Anda juga dapat menyalin snapshot bersama.

Anda tidak dapat menyalin snapshot klaster DB di seluruh Wilayah dan akun dalam satu langkah. Lakukan satu langkah untuk setiap tindakan penyalinan ini. Sebagai alternatif penyalinan, Anda juga dapat berbagi snapshot manual dengan akun AWS lain. Untuk informasi selengkapnya, lihat Berbagi snapshot klaster DB.

> **Note**
>
> Amazon menagih Anda berdasarkan jumlah data snapshot dan cadangan Amazon Aurora yang Anda simpan dan jangka waktu penyimpanannya. Untuk informasi tentang penyimpanan yang terkait dengan cadangan dan snapshot Aurora, lihat Memahami penggunaan penyimpanan cadangan Amazon Aurora. Untuk informasi harga terkait penyimpanan Aurora, lihat Harga Amazon RDS for Aurora.

Batasan

Berikut adalah beberapa batasan saat Anda menyalin snapshot:

- Anda tidak dapat menyalin snapshot ke atau dari Wilayah AWS berikut:
  - Tiongkok (Beijing)
  - China (Ningxia)
- Anda dapat menyalin snapshot antara AWS GovCloud (AS-Timur) dan AWS GovCloud (AS-Barat). Namun, Anda tidak dapat menyalin snapshot di antara Wilayah AWS GovCloud (US) ini dan Wilayah AWS komersial.
- Jika Anda menghapus snapshot sumber sebelum snapshot target tersedia, penyalinan snapshot mungkin gagal. Verifikasi bahwa snapshot target telah berstatus `AVAILABLE` sebelum Anda menghapus snapshot sumber.

- Anda dapat memiliki maksimum lima permintaan penyalinan snapshot yang sedang berlangsung ke satu Wilayah tujuan per akun.
- Bila Anda meminta beberapa salinan snapshot untuk instans DB sumber yang sama, salinan snapshot tersebut akan berada dalam antrean internal. Salinan yang diminta nanti tidak akan dimulai hingga salinan snapshot sebelumnya selesai. Untuk informasi selengkapnya, lihat Mengapa pembuatan snapshot EC2 AMI atau EBS saya lambat? di Pusat Pengetahuan AWS.
- Tergantung pada Wilayah AWS yang terlibat dan jumlah data yang akan disalin, mungkin perlu waktu berjam-jam untuk menyelesaikan penyalinan snapshot lintas Wilayah. Dalam beberapa kasus, mungkin ada sejumlah besar permintaan penyalinan snapshot lintas Wilayah dari Wilayah sumber tertentu. Dalam kasus seperti itu, Amazon RDS mungkin menempatkan permintaan penyalinan lintas Wilayah baru dari Wilayah sumber tersebut ke dalam antrean hingga beberapa penyalinan yang sedang berlangsung selesai. Tidak ada informasi progres yang ditampilkan tentang permintaan penyalinan saat permintaan tersebut ada di dalam antrean. Informasi progres akan ditampilkan saat penyalinan dimulai.

Retensi snapshot

Amazon RDS menghapus cadangan otomatis dalam beberapa situasi:

- Pada akhir periode retensi.
- Jika Anda menonaktifkan pencadangan otomatis untuk klaster DB.
- Jika Anda menghapus klaster DB.

Jika Anda ingin menyimpan cadangan otomatis untuk jangka waktu yang lebih lama, salin cadangan otomatis tersebut untuk membuat snapshot manual, yang akan dipertahankan hingga Anda menghapusnya. Biaya penyimpanan Amazon RDS mungkin berlaku untuk snapshot manual jika melebihi ruang penyimpanan default.

Untuk informasi selengkapnya tentang biaya penyimpanan cadangan, lihat Harga Amazon RDS.

Menyalin snapshot bersama

Anda dapat menyalin snapshot yang dibagikan kepada Anda oleh akun AWS lain. Dalam beberapa kasus, Anda dapat menyalin snapshot terenkripsi yang telah dibagikan dari akun AWS lain. Dalam kasus seperti ini, Anda harus memiliki akses ke AWS KMS key yang digunakan untuk mengenkripsi snapshot tersebut.

Anda hanya dapat menyalin snapshot klaster DB bersama, baik yang dienkripsi atau tidak, di Wilayah AWS yang sama. Untuk informasi selengkapnya, lihat Berbagi snapshot terenkripsi.

Menangani enkripsi

Anda dapat menyalin snapshot yang telah dienkripsi menggunakan kunci KMS. Jika Anda menyalin snapshot terenkripsi, salinan snapshot tersebut juga harus dienkripsi. Jika Anda menyalin snapshot terenkripsi dalam Wilayah AWS yang sama, Anda dapat mengenkripsi salinan tersebut dengan kunci KMS yang sama seperti snapshot asli. Atau Anda dapat menentukan kunci KMS yang berbeda.

Jika Anda menyalin snapshot terenkripsi di seluruh Wilayah, Anda harus menentukan kunci KMS yang valid di Wilayah AWS tujuan. Kunci ini dapat berupa kunci KMS khusus Wilayah, atau kunci multi-Wilayah. Untuk informasi selengkapnya tentang kunci KMS multi-Wilayah, lihat Menggunakan kunci multi-Wilayah di AWS KMS.

Snapshot sumber tetap terenkripsi selama proses penyalinan. Untuk informasi selengkapnya, lihat .

> **Note**
>
> Untuk snapshot klaster DB Amazon Aurora, Anda tidak dapat mengenkripsi snapshot klaster DB yang tidak terenkripsi saat Anda menyalin snapshot.

Penyalinan snapshot inkremental

Aurora tidak mendukung penyalinan snapshot inkremental. Salinan snapshot klaster DB Aurora selalu merupakan salinan lengkap. Salinan snapshot lengkap berisi semua data dan metadata yang diperlukan untuk memulihkan klaster DB.

Penyalinan snapshot lintas Wilayah

Anda dapat menyalin snapshot klaster DB di seluruh Wilayah AWS. Namun, ada kendala dan pertimbangan tertentu dalam penyalinan snapshot lintas Wilayah.

Tergantung pada Wilayah AWS yang terlibat dan jumlah data yang akan disalin, mungkin perlu waktu berjam-jam untuk menyelesaikan penyalinan snapshot lintas Wilayah.

Dalam beberapa kasus, mungkin ada sejumlah besar permintaan penyalinan snapshot lintas Wilayah dari Wilayah AWS sumber tertentu. Dalam kasus tersebut, Amazon RDS mungkin menempatkan

permintaan penyalinan lintas Wilayah baru dari Wilayah AWS sumber tersebut ke dalam antrean hingga beberapa penyalinan yang sedang berlangsung selesai. Tidak ada informasi progres yang ditampilkan tentang permintaan penyalinan saat permintaan tersebut ada di dalam antrean. Informasi progres akan ditampilkan saat penyalinan dimulai.

Pertimbangan grup parameter

Saat Anda menyalin snapshot lintas Wilayah, salinannya tidak menyertakan grup parameter yang digunakan oleh klaster DB asli. Saat Anda memulihkan snapshot untuk membuat klaster DB baru, klaster DB tersebut mendapatkan grup parameter default untuk Wilayah AWS tempatnya dibuat. Untuk memberi klaster DB baru parameter yang sama seperti aslinya, Anda harus melakukan hal berikut:

1. Di Wilayah AWS tujuan, buat grup parameter klaster DB dengan pengaturan yang sama seperti klaster DB aslinya. Jika sudah ada grup parameter klaster DB di Wilayah AWS baru, Anda dapat menggunakannya.
2. Setelah Anda memulihkan snapshot di Wilayah AWS tujuan, ubah klaster DB baru dan tambahkan grup parameter baru atau yang sudah ada dari langkah sebelumnya.

Menyalin snapshot klaster DB

Gunakan prosedur dalam topik ini untuk menyalin snapshot klaster DB. Jika mesin basis data sumber Anda adalah Aurora, maka snapshot Anda adalah snapshot klaster DB.

Untuk setiap akun AWS, Anda dapat menyalin hingga lima snapshot klaster DB sekaligus dari satu Wilayah AWS ke yang lainnya. Menyalin snapshot klaster DB terenkripsi maupun tidak terenkripsi didukung. Jika Anda menyalin snapshot klaster DB ke Wilayah AWS lain, berarti Anda membuat snapshot klaster DB manual yang dipertahankan dalam Wilayah AWS tersebut. Menyalin snapshot klaster DB dari Wilayah AWS sumber akan menimbulkan biaya transfer data Amazon RDS.

Untuk informasi selengkapnya tentang biaya transfer data, lihat Harga Amazon RDS.

Setelah salinan snapshot klaster DB dibuat di Wilayah AWS baru, salinan snapshot klaster DB itu akan berperilaku sama seperti semua snapshot klaster DB lainnya dalam Wilayah AWS tersebut.

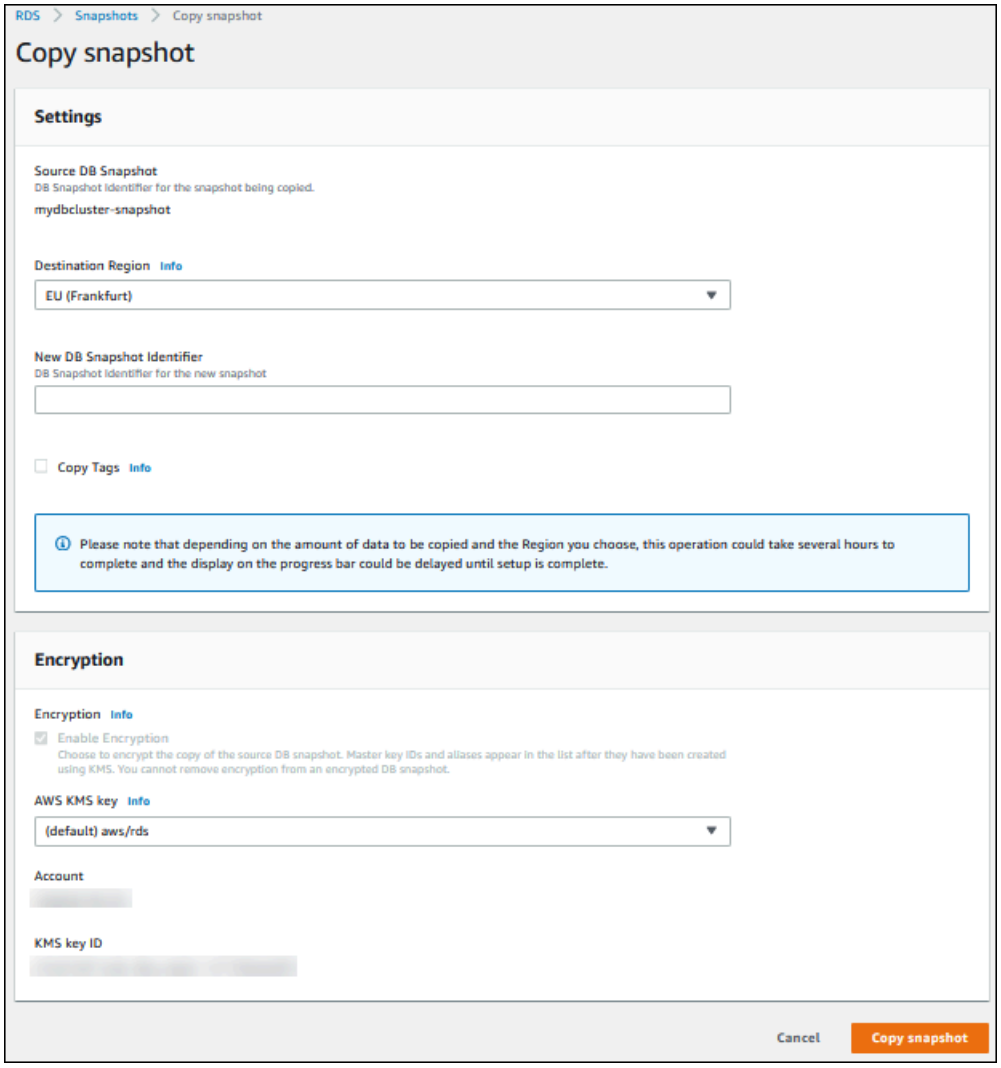
Konsol

Prosedur ini berfungsi untuk menyalin snapshot klaster DB terenkripsi atau tidak terenkripsi, di Wilayah AWS yang sama atau di seluruh Wilayah.

Untuk membatalkan operasi penyalinan setelah berlangsung, hapus snapshot klaster DB target saat snapshot klaster DB tersebut dalam status menyalin.

Untuk menyalin snapshot klaster DB

1. Masuk ke AWS Management Console, lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih snapshot cluster DB yang ingin Anda salin.
4. Untuk Tindakan, pilih Salin snapshot. Halaman Salin snapshot akan ditampilkan.



5. (Opsional) Untuk menyalin snapshot klaster DB ke Wilayah AWS lain, pilih Wilayah AWS tersebut untuk Wilayah Tujuan.
6. Masukkan nama salinan snapshot klaster DB dalam Pengidentifikasi Snapshot DB Baru.

7. Untuk menyalin tanda dan nilai dari snapshot ke salinan snapshot, pilih Salin Tanda.
8. Pilih Salin Snapshot.

Menyalin snapshot klaster DB yang tidak terenkripsi menggunakan AWS CLI atau API Amazon RDS

Gunakan prosedur di bagian berikut untuk menyalin snapshot klaster DB yang tidak terenkripsi dengan menggunakan AWS CLI atau API Amazon RDS.

Untuk membatalkan operasi penyalinan setelah dijalankan, hapus snapshot klaster DB target yang diidentifikasi oleh `--target-db-cluster-snapshot-identifier` atau `TargetDBClusterSnapshotIdentifier` saat snapshot klaster DB tersebut berstatus menyalin.

AWS CLI

Untuk menyalin snapshot klaster DB, gunakan perintah AWS CLI copy-db-cluster-snapshot. Jika Anda menyalin snapshot ke Wilayah AWS lain, jalankan perintah tersebut di Wilayah AWS tempat snapshot akan disalin.

Opsi berikut digunakan untuk menyalin snapshot klaster DB yang tidak terenkripsi:

- `--source-db-cluster-snapshot-identifier` – Pengidentifikasi untuk snapshot klaster DB yang akan disalin. Jika Anda menyalin snapshot ke Wilayah AWS lain, pengidentifikasi ini harus dalam format ARN untuk Wilayah AWS sumber.
- `--target-db-cluster-snapshot-identifier` – Pengidentifikasi untuk salinan baru snapshot klaster DB.

Kode berikut membuat salinan snapshot klaster DB `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805` bernama `myclustersnapshotcopy` di Wilayah AWS tempat perintah dijalankan. Setelah salinan dibuat, semua tanda di snapshot asli akan disalin ke salinan snapshot.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 \
```

```
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
  --copy-tags
```

Untuk Windows:

```
aws rds copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 ^
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --copy-tags
```

API RDS

Untuk menyalin snapshot cluster DB, gunakan operasi Amazon RDS API ClusterSnapshotCopyDB. Jika Anda menyalin snapshot ke Wilayah AWS lain, lakukan tindakan tersebut di Wilayah AWS tempat snapshot akan disalin.

Parameter berikut digunakan untuk menyalin snapshot klaster DB yang tidak terenkripsi:

- `SourceDBClusterSnapshotIdentifier` – Pengidentifikasi untuk snapshot klaster DB yang akan disalin. Jika Anda menyalin snapshot ke Wilayah AWS lain, pengidentifikasi ini harus dalam format ARN untuk Wilayah AWS sumber.
- `TargetDBClusterSnapshotIdentifier` – Pengidentifikasi untuk salinan baru snapshot klaster DB.

Kode berikut membuat salinan snapshot `arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805` bernama `myclustersnapshotcopy` di Wilayah AS Barat (California Utara). Setelah salinan dibuat, semua tanda di snapshot asli akan disalin ke salinan snapshot.

Example

```
https://rds.us-west-1.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-east-1%3A123456789012%3Acluster-snapshot%3Aaurora-cluster1-snapshot-20130805
```

```
    &TargetDBSnapshotIdentifier=myclustersnapshotcopy
    &Version=2013-09-09
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20140429/us-west-1/rds/aws4_request
    &X-Amz-Date=20140429T175351Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

Menyalin snapshot klaster DB terenkripsi menggunakan AWS CLI atau API Amazon RDS

Gunakan prosedur di bagian berikut untuk menyalin snapshot klaster DB terenkripsi menggunakan AWS CLI atau API Amazon RDS.

Untuk membatalkan operasi penyalinan setelah dijalankan, hapus snapshot klaster DB target yang diidentifikasi oleh `--target-db-cluster-snapshot-identifier` atau `TargetDBClusterSnapshotIdentifier` saat snapshot klaster DB tersebut berstatus menyalin.

AWS CLI

Untuk menyalin snapshot klaster DB, gunakan perintah AWS CLI copy-db-cluster-snapshot. Jika Anda menyalin snapshot ke Wilayah AWS lain, jalankan perintah tersebut di Wilayah AWS tempat snapshot akan disalin.

Opsi berikut digunakan untuk menyalin snapshot klaster DB terenkripsi:

- `--source-db-cluster-snapshot-identifier` – Pengidentifikasi untuk snapshot klaster DB terenkripsi yang akan disalin. Jika Anda menyalin snapshot ke Wilayah AWS lain, pengidentifikasi ini harus dalam format ARN untuk Wilayah AWS sumber.
- `--target-db-cluster-snapshot-identifier` – Pengidentifikasi untuk salinan baru snapshot klaster DB terenkripsi.
- `--kms-key-id` – Pengidentifikasi kunci KMS untuk kunci yang akan digunakan untuk mengenkripsi salinan snapshot klaster DB.

  Anda dapat menggunakan opsi ini jika snapshot klaster DB terenkripsi. Salin snapshot tersebut di Wilayah AWS yang sama, dan sebaiknya tentukan kunci KMS baru yang akan digunakan untuk mengenkripsi salinan. Jika tidak, salinan snapshot klaster DB akan dienkripsi dengan kunci KMS yang sama seperti snapshot klaster DB sumber.

Anda harus menggunakan opsi ini jika snapshot klaster DB terenkripsi dan Anda menyalin snapshot ke Wilayah AWS lain. Dalam hal ini, Anda harus menentukan kunci KMS untuk Wilayah AWS tujuan.

Contoh kode berikut menyalin snapshot klaster DB terenkripsi dari Wilayah AS Barat (Oregon) ke Wilayah AS Timur (Virginia Utara). Perintah ini dipanggil di Wilayah AS Timur (Virginia Utara).

Example

Untuk Linux, macOS, atau Unix:

```
aws rds copy-db-cluster-snapshot \
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20161115 \
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy \
  --kms-key-id my-us-east-1-key
```

Untuk Windows:

```
aws rds copy-db-cluster-snapshot ^
  --source-db-cluster-snapshot-identifier arn:aws:rds:us-west-2:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20161115 ^
  --target-db-cluster-snapshot-identifier myclustersnapshotcopy ^
  --kms-key-id my-us-east-1-key
```

`--source-region`Parameter diperlukan saat Anda menyalin snapshot cluster DB terenkripsi antara Wilayah AWS GovCloud (AS-Timur) dan AWS GovCloud (AS-Barat). Untuk `--source-region`, tentukan Wilayah AWS instans DB sumber. Wilayah AWS yang ditentukan di `source-db-cluster-snapshot-identifier` harus cocok dengan Wilayah AWS yang ditentukan untuk`--source-region`.

Jika `--source-region` tidak ditentukan, tentukan nilai `--pre-signed-url`. URL yang telah ditandatangani adalah URL yang berisi permintaan bertanda tangan Signature Versi 4 untuk perintah `copy-db-cluster-snapshot` yang dipanggil di Wilayah AWS sumber. Untuk mempelajari lebih lanjut tentang `pre-signed-url` opsi, lihat copy-db-cluster-snapshotdi Referensi AWS CLI Perintah.

API RDS

Untuk menyalin snapshot cluster DB, gunakan operasi Amazon RDS API ClusterSnapshotCopyDB. Jika Anda menyalin snapshot ke Wilayah AWS lain, lakukan tindakan tersebut di Wilayah AWS tempat snapshot akan disalin.

Parameter berikut digunakan untuk menyalin snapshot klaster DB terenkripsi:

- `SourceDBClusterSnapshotIdentifier` – Pengidentifikasi untuk snapshot klaster DB terenkripsi yang akan disalin. Jika Anda menyalin snapshot ke Wilayah AWS lain, pengidentifikasi ini harus dalam format ARN untuk Wilayah AWS sumber.
- `TargetDBClusterSnapshotIdentifier` – Pengidentifikasi untuk salinan baru snapshot klaster DB terenkripsi.
- `KmsKeyId` – Pengidentifikasi kunci KMS untuk kunci yang akan digunakan untuk mengenkripsi salinan snapshot klaster DB.

  Anda dapat menggunakan parameter ini secara opsional jika snapshot klaster DB terenkripsi. Salin snapshot di Wilayah AWS yang sama, lalu tentukan kunci KMS baru yang akan digunakan untuk mengenkripsi salinan tersebut. Jika tidak, salinan snapshot klaster DB akan dienkripsi dengan kunci KMS yang sama seperti snapshot klaster DB sumber.

  Anda harus menggunakan parameter ini jika snapshot klaster DB terenkripsi dan Anda menyalin snapshot ke Wilayah AWS lain. Dalam hal ini, Anda harus menentukan kunci KMS untuk Wilayah AWS tujuan.
- `PreSignedUrl` – Jika Anda menyalin snapshot ke Wilayah AWS lain, Anda harus menentukan parameter `PreSignedUrl`. Nilai `PreSignedUrl` harus berupa URL yang berisi permintaan bertanda tangan Signature Versi 4 untuk tindakan `CopyDBClusterSnapshot` yang akan dipanggil di Wilayah AWS sumber tempat snapshot klaster DB disalin. Untuk mempelajari selengkapnya tentang menggunakan URL yang telah ditetapkan sebelumnya, lihat ClusterSnapshotCopyDB.

Contoh kode berikut menyalin snapshot klaster DB terenkripsi dari Wilayah AS Barat (Oregon) ke Wilayah AS Timur (Virginia Utara). Tindakan ini dipanggil di Wilayah AS Timur (Virginia Utara).

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CopyDBClusterSnapshot
```

```
    &KmsKeyId=my-us-east-1-key
    &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCopyDBClusterSnapshot
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526SourceDBClusterSnapshotIdentifier%253Darn%25253Aaws%25253Ards
%25253Aus-west-2%25253A123456789012%25253Acluster-snapshot%25253Aaurora-cluster1-
snapshot-20161115
         %2526SignatureMethod%253DHmacSHA256
         %2526SignatureVersion%253D4
         %2526Version%253D2014-10-31
         %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
         %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-west-2%252Frds
%252Faws4_request
         %2526X-Amz-Date%253D20161117T215409Z
         %2526X-Amz-Expires%253D3600
         %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-amz-
content-sha256%253Bx-amz-date
         %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &SourceDBClusterSnapshotIdentifier=arn%3Aaws%3Ards%3Aus-
west-2%3A123456789012%3Acluster-snapshot%3Aaurora-cluster1-snapshot-20161115
    &TargetDBClusterSnapshotIdentifier=myclustersnapshotcopy
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
    &X-Amz-Date=20161117T221704Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=da4f2da66739d2e722c85fcfd225dc27bba7e2b8dbea8d8612434378e52adccf
```

`PreSignedUrl`Parameter diperlukan saat Anda menyalin snapshot cluster DB terenkripsi antara Wilayah AWS GovCloud (AS-Timur) dan AWS GovCloud (AS-Barat). Nilai `PreSignedUrl` harus berupa URL yang berisi permintaan bertanda tangan Signature Versi 4 untuk operasi `CopyDBClusterSnapshot` yang akan dipanggil di Wilayah AWS sumber tempat snapshot klaster DB disalin. Untuk mempelajari selengkapnya tentang menggunakan URL yang telah ditetapkan sebelumnya, lihat CopyDB ClusterSnapshot di Referensi API Amazon RDS.

Untuk membuat URL yang sudah ditandatangani secara otomatis ketimbang secara manual, gunakan perintah AWS CLI copy-db-cluster-snapshot dengan opsi `--source-region`.

Menyalin snapshot klaster DB di seluruh akun

Anda dapat mengaktifkan akun AWS lain untuk menyalin snapshot klaster DB yang Anda tentukan dengan menggunakan `ModifyDBClusterSnapshotAttribute` API Amazon RDS dan tindakan `CopyDBClusterSnapshot`. Anda hanya dapat menyalin snapshot klaster DB ke seluruh akun di Wilayah AWS yang sama. Proses penyalinan lintas akun berfungsi sebagai berikut: Akun A menyediakan snapshot untuk disalin, dan Akun B menyalinnya.

1. Dengan menggunakan Akun A, panggil `ModifyDBClusterSnapshotAttribute`, dengan menentukan **`restore`** untuk parameter `AttributeName`, dan ID untuk Akun B untuk parameter `ValuesToAdd`.
2. (Jika snapshot terenkripsi) Dengan menggunakan Akun A, perbarui kebijakan kunci untuk kunci KMS, dengan pertama-tama menambahkan ARN Akun B sebagai `Principal`, lalu mengizinkan tindakan `kms:CreateGrant`.
3. (Jika snapshot terenkripsi) Dengan menggunakan Akun B, pilih atau buat pengguna, lalu lampirkan kebijakan IAM untuk pengguna tersebut, yang akan memungkinkannya menyalin snapshot klaster DB terenkripsi menggunakan kunci KMS Anda.
4. Dengan menggunakan Akun B, panggil `CopyDBClusterSnapshot` dan gunakan parameter `SourceDBClusterSnapshotIdentifier` untuk menentukan ARN snapshot klaster DB yang akan disalin, yang harus menyertakan ID untuk Akun A.

Untuk mencantumkan semua AWS akun yang diizinkan untuk memulihkan snapshot cluster DB, gunakan operasi DescribeDB SnapshotAttributes atau DescribeDB API. ClusterSnapshotAttributes

Untuk menghapus izin berbagi untuk akun AWS, gunakan tindakan `ModifyDBSnapshotAttribute` atau `ModifyDBClusterSnapshotAttribute` dengan `AttributeName` diatur ke `restore` dan ID akun yang akan dihapus di parameter `ValuesToRemove`.

Menyalin snapshot klaster DB tidak terenkripsi ke akun lain

Gunakan prosedur berikut untuk menyalin snapshot klaster DB yang tidak terenkripsi ke akun lain di Wilayah AWS yang sama.

1. Di akun sumber untuk snapshot klaster DB, panggil `ModifyDBClusterSnapshotAttribute`, dengan menentukan **`restore`** untuk parameter `AttributeName`, dan ID akun target untuk parameter `ValuesToAdd`.

Menjalankan contoh berikut menggunakan akun 987654321 akan mengizinkan dua pengidentifikasi akun AWS, 123451234512 dan 123456789012, untuk memulihkan snapshot klaster DB bernama manual-snapshot1.

```
https://rds.us-west-2.amazonaws.com/
 ?Action=ModifyDBClusterSnapshotAttribute
 &AttributeName=restore
 &DBClusterSnapshotIdentifier=manual-snapshot1
 &SignatureMethod=HmacSHA256&SignatureVersion=4
 &ValuesToAdd.member.1=123451234512
 &ValuesToAdd.member.2=123456789012
 &Version=2014-10-31
 &X-Amz-Algorithm=AWS4-HMAC-SHA256
 &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
 &X-Amz-Date=20150922T220515Z
 &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
 &X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
```

2. Di akun target, panggil CopyDBClusterSnapshot dan gunakan parameter SourceDBClusterSnapshotIdentifier untuk menentukan ARN snapshot klaster DB yang akan disalin, yang harus menyertakan ID untuk akun sumber.

   Menjalankan contoh berikut menggunakan akun 123451234512 akan menyalin snapshot klaster DB aurora-cluster1-snapshot-20130805 dari akun 987654321 dan membuat snapshot klaster DB bernama dbclustersnapshot1.

```
https://rds.us-west-2.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
```

```
   &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

Menyalin snapshot klaster DB terenkripsi ke akun lain

Gunakan prosedur berikut untuk menyalin snapshot klaster DB terenkripsi ke akun lain di Wilayah AWS yang sama.

1. Di akun sumber untuk snapshot klaster DB, panggil `ModifyDBClusterSnapshotAttribute`, dengan menentukan **`restore`** untuk parameter `AttributeName`, dan ID akun target untuk parameter `ValuesToAdd`.

   Menjalankan contoh berikut menggunakan akun `987654321` akan mengizinkan dua pengidentifikasi akun AWS, `123451234512` dan `123456789012`, untuk memulihkan snapshot klaster DB bernama `manual-snapshot1`.

   ```
   https://rds.us-west-2.amazonaws.com/
    ?Action=ModifyDBClusterSnapshotAttribute
    &AttributeName=restore
    &DBClusterSnapshotIdentifier=manual-snapshot1
    &SignatureMethod=HmacSHA256&SignatureVersion=4
    &ValuesToAdd.member.1=123451234512
    &ValuesToAdd.member.2=123456789012
    &Version=2014-10-31
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
    &X-Amz-Date=20150922T220515Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36dddbb3
   ```

2. Di akun sumber untuk snapshot cluster DB, buat kunci KMS kustom Wilayah AWS sama dengan snapshot cluster DB terenkripsi. Saat membuat kunci yang dikelola pelanggan, Anda memberikan akses ke sana untuk targetAkun AWS. Untuk informasi selengkapnya, lihat Buat kunci yang dikelola pelanggan dan berikan akses ke sana.
3. Salin dan bagikan snapshot ke targetAkun AWS. Untuk informasi selengkapnya, lihat Salin dan bagikan snapshot dari akun sumber.
4. Di akun target, panggil `CopyDBClusterSnapshot` dan gunakan parameter `SourceDBClusterSnapshotIdentifier` untuk menentukan ARN snapshot klaster DB yang akan disalin, yang harus menyertakan ID untuk akun sumber.

Menjalankan contoh berikut menggunakan akun 123451234512 akan menyalin snapshot klaster DB aurora-cluster1-snapshot-20130805 dari akun 987654321 dan membuat snapshot klaster DB bernama dbclustersnapshot1.

```
https://rds.us-west-2.amazonaws.com/
   ?Action=CopyDBClusterSnapshot
   &CopyTags=true
   &SignatureMethod=HmacSHA256
   &SignatureVersion=4
   &SourceDBClusterSnapshotIdentifier=arn:aws:rds:us-west-2:987654321:cluster-
snapshot:aurora-cluster1-snapshot-20130805
   &TargetDBClusterSnapshotIdentifier=dbclustersnapshot1
   &Version=2013-09-09
   &X-Amz-Algorithm=AWS4-HMAC-SHA256
   &X-Amz-Credential=AKIADQKE4SARGYLE/20150922/us-west-2/rds/aws4_request
   &X-Amz-Date=20140429T175351Z
   &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-
date
   &X-Amz-
Signature=9164337efa99caf850e874a1cb7ef62f3cea29d0b448b9e0e7c53b288ddffed2
```

Berbagi snapshot klaster DB

Dengan Amazon RDS, Anda dapat berbagi snapshot klaster DB manual dengan cara berikut:

- Berbagi snapshot klaster DB manual, baik yang dienkripsi maupun tidak, sehingga memungkinkan akun AWS terotorisasi menyalin snapshot tersebut.
- Dengan berbagi snapshot klaster DB manual, baik yang dienkripsi maupun tidak, akun AWS terotorisasi akan dapat memulihkan klaster DB secara langsung dari snapshot ini, bukan membuat salinannya dan memulihkan dari salinan tersebut.

> **Note**
>
> Untuk berbagi snapshot klaster DB otomatis, buat klaster DB manual dengan menyalin snapshot otomatis, lalu membagikannya. Proses ini juga berlaku untuk sumber daya yang dihasilkan oleh AWS Backup.

Untuk informasi selengkapnya tentang cara menyalin snapshot, lihat Menyalin snapshot klaster DB. Untuk informasi selengkapnya tentang memulihkan instans DB dari snapshot klaster DB, lihat Memulihkan dari snapshot klaster DB.

Untuk informasi selengkapnya tentang memulihkan klaster DB dari snapshot klaster DB, lihat Gambaran umum pencadangan dan pemulihan klaster DB Aurora.

Anda dapat berbagi snapshot manual dengan hingga 20 lainnya Akun AWS.

Batasan berikut berlaku saat berbagi foto manual dengan yang lain Akun AWS:

- Saat memulihkan kluster DB dari snapshot bersama menggunakan AWS Command Line Interface (AWS CLI) atau Amazon RDS API, Anda harus menentukan Nama Sumber Daya Amazon (ARN) dari snapshot bersama sebagai pengenal snapshot.

Daftar Isi

- Berbagi snapshot
- Berbagi snapshot publik
  - Melihat snapshot publik yang dimiliki oleh orang lain Akun AWS
  - Melihat snapshot publik Anda sendiri

  - Berbagi snapshot publik dari versi mesin DB yang tidak digunakan lagi
- Berbagi snapshot terenkripsi
  - Buat kunci yang dikelola pelanggan dan berikan akses ke sana
  - Salin dan bagikan snapshot dari akun sumber
  - Salin snapshot bersama di akun target
- Menghentikan berbagi snapshot

Berbagi snapshot

Anda dapat membagikan snapshot cluster DB menggunakan AWS Management Console API, AWS CLI, atau RDS.

Konsol

Menggunakan konsol Amazon RDS, Anda dapat membagikan snapshot cluster DB manual hingga 20. Akun AWS Anda juga dapat menggunakan konsol untuk berhenti berbagi snapshot manual dengan satu atau beberapa akun.

Untuk berbagi snapshot klaster DB manual dengan menggunakan konsol Amazon RDS

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih snapshot manual yang ingin Anda bagikan.
4. Untuk Tindakan, pilih Bagikan snapshot.
5. Pilih salah satu opsi berikut untuk Visibilitas snapshot DB.

   - Jika sumbernya tidak terenkripsi, pilih Publik untuk mengizinkan semua Akun AWS memulihkan cluster DB dari snapshot cluster DB manual Anda, atau pilih Private untuk mengizinkan hanya Akun AWS yang Anda tentukan untuk memulihkan cluster DB dari snapshot cluster DB manual Anda.

     > ⚠ Warning
     >
     > Jika Anda menyetel visibilitas snapshot DB ke Publik, semua Akun AWS dapat memulihkan cluster DB dari snapshot cluster DB manual Anda dan memiliki akses ke

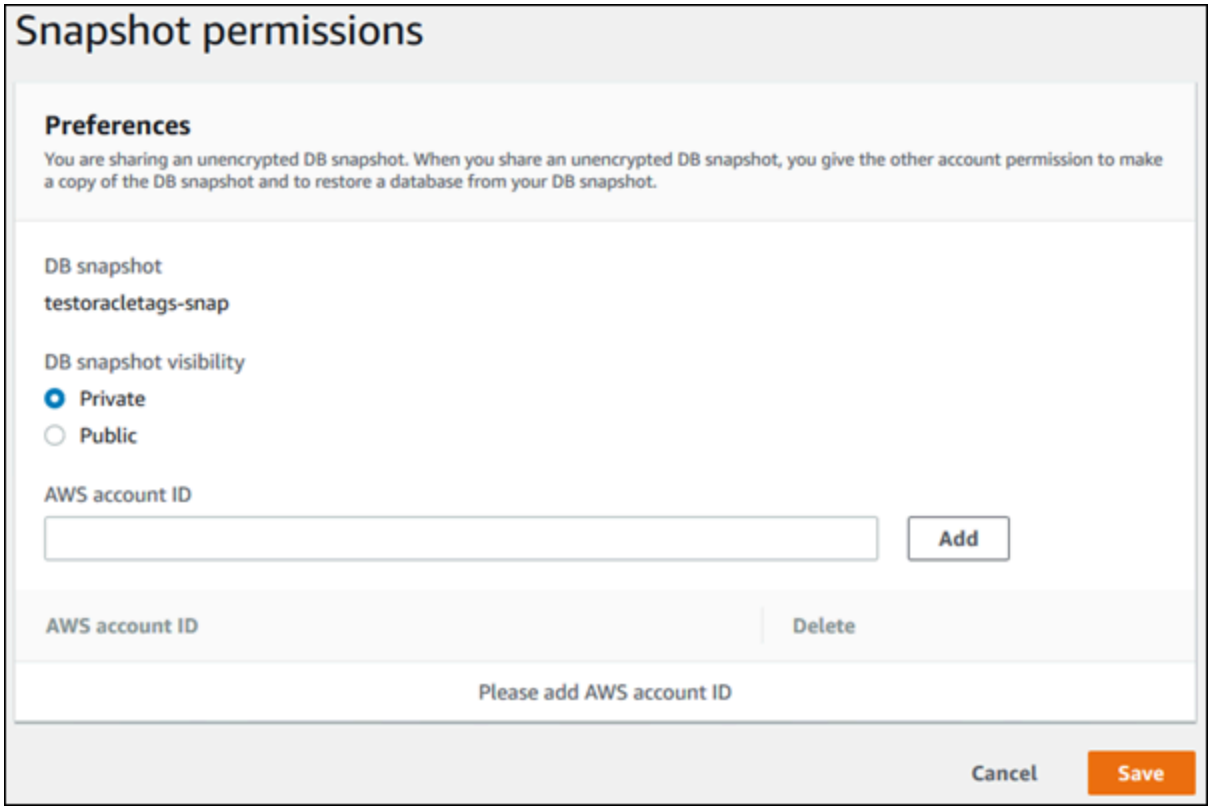
data Anda. Jangan berbagi snapshot klaster DB manual apa pun yang berisi informasi pribadi sebagai Publik.
Untuk informasi selengkapnya, lihat Berbagi snapshot publik.

- Jika sumbernya dienkripsi, Visibilitas snapshot DB ditetapkan sebagai Privat karena snapshot terenkripsi tidak dapat dibagikan sebagai publik.

> **Note**
>
> Snapshot yang telah dienkripsi dengan default tidak AWS KMS key dapat dibagikan. Untuk informasi tentang cara mengatasi masalah ini, lihatBerbagi snapshot terenkripsi.

6. Untuk ID AWS Akun, masukkan Akun AWS pengenal untuk akun yang ingin Anda izinkan untuk memulihkan kluster DB dari snapshot manual Anda, lalu pilih Tambah. Ulangi untuk menyertakan Akun AWS pengidentifikasi tambahan, hingga 20 Akun AWS.

   Jika Anda membuat kesalahan saat menambahkan Akun AWS pengenal ke daftar akun yang diizinkan, Anda dapat menghapusnya dari daftar dengan memilih Hapus di sebelah kanan Akun AWS pengidentifikasi yang salah.



7. Setelah Anda menambahkan pengenal untuk semua Akun AWS yang ingin Anda izinkan untuk memulihkan snapshot manual, pilih Simpan untuk menyimpan perubahan Anda.

AWS CLI

Untuk berbagi snapshot klaster DB, gunakan perintah `aws rds modify-db-cluster-snapshot-attribute`. Gunakan parameter `--values-to-add` untuk menambahkan daftar ID untuk Akun AWS yang diotorisasi untuk memulihkan snapshot manual.

Example berbagi snapshot dengan satu akun

Contoh berikut memungkinkan Akun AWS identifier `123456789012` untuk mengembalikan snapshot cluster DB bernama. `cluster-3-snapshot`

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier cluster-3-snapshot \
--attribute-name restore \
--values-to-add 123456789012
```

Untuk Windows:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier cluster-3-snapshot ^
--attribute-name restore ^
--values-to-add 123456789012
```

Example berbagi snapshot dengan beberapa akun

Contoh berikut memungkinkan dua Akun AWS pengidentifikasi, `111122223333` dan`444455556666`, untuk mengembalikan snapshot cluster DB bernama. `manual-cluster-snapshot1`

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-add {"111122223333","444455556666"}
```

Untuk Windows:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-add "[\"111122223333\",\"444455556666\"]"
```

> **Note**
>
> Saat menggunakan command prompt Windows, Anda harus meng-escape tanda kutip ganda (") dalam kode JSON dengan memberikan garis miring terbalik (\) di depannya.

Untuk membuat daftar yang Akun AWS diaktifkan untuk memulihkan snapshot, gunakan `describe-db-cluster-snapshot-attributes` AWS CLI perintah.

API RDS

Anda juga dapat berbagi snapshot cluster DB manual dengan yang lain Akun AWS dengan menggunakan Amazon RDS API. Untuk melakukannya, panggil operasi `ModifyDBClusterSnapshotAttribute`. Tentukan `restore` untuk`AttributeName`, dan gunakan `ValuesToAdd` parameter untuk menambahkan daftar ID untuk Akun AWS yang berwenang untuk mengembalikan snapshot manual.

Untuk membuat snapshot manual publik dan dapat dipulihkan oleh semua orang Akun AWS, gunakan nilainya. `all` Namun, berhati-hatilah untuk tidak menambahkan `all` nilai untuk setiap snapshot manual yang berisi informasi pribadi yang Anda tidak ingin tersedia untuk semua Akun AWS. Selain itu, jangan tentukan `all` untuk snapshot terenkripsi karena menjadikan snapshot tersebut berstatus publik tidak didukung.

Untuk mencantumkan semua yang Akun AWS diizinkan untuk memulihkan snapshot, gunakan operasi `DescribeDBClusterSnapshotAttributes`API.

Berbagi snapshot publik

Anda dapat membagikan snapshot manual yang tidak terenkripsi sebagai publik, yang membuat snapshot tersedia untuk semua. Akun AWS Pastikan saat berbagi snapshot sebagai publik bahwa tidak ada informasi pribadi yang dimasukkan ke dalam snapshot publik.

Ketika snapshot dibagikan secara publik, ia memberikan semua Akun AWS izin untuk menyalin snapshot dan membuat cluster DB darinya.

Anda tidak ditagih untuk penyimpanan cadangan snapshot publik yang dimiliki oleh akun lain. Anda hanya ditagih untuk snapshot yang Anda miliki.

Jika Anda menyalin snapshot publik, Anda memiliki salinannya. Anda ditagih untuk penyimpanan cadangan salinan snapshot Anda. Jika Anda membuat klaster DB dari snapshot publik, Anda ditagih untuk klaster DB tersebut. Untuk informasi harga Amazon Aurora, lihat halaman harga Aurora.

Anda hanya dapat menghapus snapshot publik yang Anda miliki. Untuk menghapus snapshot bersama atau publik, pastikan untuk masuk ke Akun AWS yang memiliki snapshot.

Melihat snapshot publik yang dimiliki oleh orang lain Akun AWS

Anda dapat melihat snapshot publik yang dimiliki oleh akun lain secara khusus Wilayah AWS di tab Publik halaman Snapshots di konsol Amazon RDS. Snapshot Anda (yang dimiliki oleh akun Anda) tidak muncul di tab ini.

Untuk melihat snapshot publik

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih tab Publik.

   Snapshot publik muncul. Anda dapat melihat akun mana yang memiliki snapshot publik di kolom Pemilik.

   > **Note**
   >
   > Anda mungkin harus mengubah preferensi halaman, dengan memilih ikon roda gigi di kanan atas daftar Snapshot publik, untuk melihat kolom ini.

Melihat snapshot publik Anda sendiri

Anda dapat menggunakan AWS CLI perintah berikut (hanya Unix) untuk melihat snapshot publik yang dimiliki oleh Anda Akun AWS secara khusus. Wilayah AWS

```
aws rds describe-db-cluster-snapshots --snapshot-type public --include-public |
 grep account_number
```

Output yang ditampilkan mirip dengan contoh berikut jika Anda memiliki snapshot publik:

```
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot1",
"DBClusterSnapshotArn": "arn:aws:rds:us-west-2:123456789012:cluster-
snapshot:myclustersnapshot2",
```

Berbagi snapshot publik dari versi mesin DB yang tidak digunakan lagi

Memulihkan atau menyalin snapshot publik dari versi mesin DB yang tidak digunakan lagi tidak didukung. Agar snapshot publik yang tidak didukung tersedia untuk dipulihkan atau disalin, lakukan langkah-langkah berikut:

1. Tandai snapshot sebagai pribadi.
2. Pulihkan snapshot yang telah disalin.
3. Tingkatkan cluster DB yang dipulihkan ke versi mesin yang didukung.
4. Buat snapshot baru.
5. Bagikan kembali snapshot secara publik.

Berbagi snapshot terenkripsi

Anda dapat berbagi snapshot klaster DB yang telah dienkripsi "saat diam" menggunakan algoritma enkripsi AES-256, sebagaimana dijelaskan dalam Mengenkripsi sumber daya Amazon Aurora.

Pembatasan berikut berlaku untuk berbagi snapshot terenkripsi:

- Anda tidak dapat berbagi snapshot terenkripsi sebagai publik.
- Anda tidak dapat membagikan snapshot yang telah dienkripsi menggunakan kunci KMS default dari Akun AWS yang membagikan snapshot.

Untuk mengatasi masalah kunci KMS default, lakukan tugas-tugas berikut:

1. Buat kunci yang dikelola pelanggan dan berikan akses ke sana.
2. Salin dan bagikan snapshot dari akun sumber.
3. Salin snapshot bersama di akun target.

Buat kunci yang dikelola pelanggan dan berikan akses ke sana

Pertama, Anda membuat kunci KMS kustom Wilayah AWS sama dengan snapshot cluster DB terenkripsi. Saat membuat kunci yang dikelola pelanggan, Anda memberikan akses ke sana untuk yang lain Akun AWS.

Untuk membuat kunci yang dikelola pelanggan dan memberikan akses ke sana

1. Masuk ke AWS Management Console dari sumbernya Akun AWS.
2. Buka AWS KMS konsol di https://console.aws.amazon.com/kms.
3. Untuk mengubah Wilayah AWS, gunakan pemilih Wilayah di sudut kanan atas halaman.
4. Di panel navigasi, pilih Kunci yang dikelola pelanggan.
5. Pilih Buat kunci.
6. Pada halaman tombol Konfigurasi:
    a. Untuk Key type, pilih Symmetric.
    b. Untuk penggunaan Kunci, pilih Enkripsi dan dekripsi.
    c. Perluas Opsi lanjutan.
    d. Untuk asal bahan utama, pilih KMS.
    e. Untuk Regionalitas, pilih kunci Wilayah Tunggal.
    f. Pilih Berikutnya.
7. Pada halaman Tambahkan label:
    a. Untuk Alias. masukkan nama tampilan untuk kunci KMS Anda, misalnya. **`share-snapshot`**
    b. (Opsional) Masukkan deskripsi untuk kunci KMS Anda.
    c. (Opsional) Tambahkan tag ke kunci KMS Anda.
    d. Pilih Berikutnya.
8. Pada halaman Tentukan izin administratif kunci, pilih Berikutnya.
9. Pada halaman Tentukan izin penggunaan kunci:
    a. Untuk Lainnya Akun AWS, pilih Tambahkan yang lain Akun AWS.
    b. Masukkan ID yang Akun AWS ingin Anda berikan aksesnya.

        Anda dapat memberikan akses ke beberapa Akun AWS.
    c. Pilih Berikutnya.

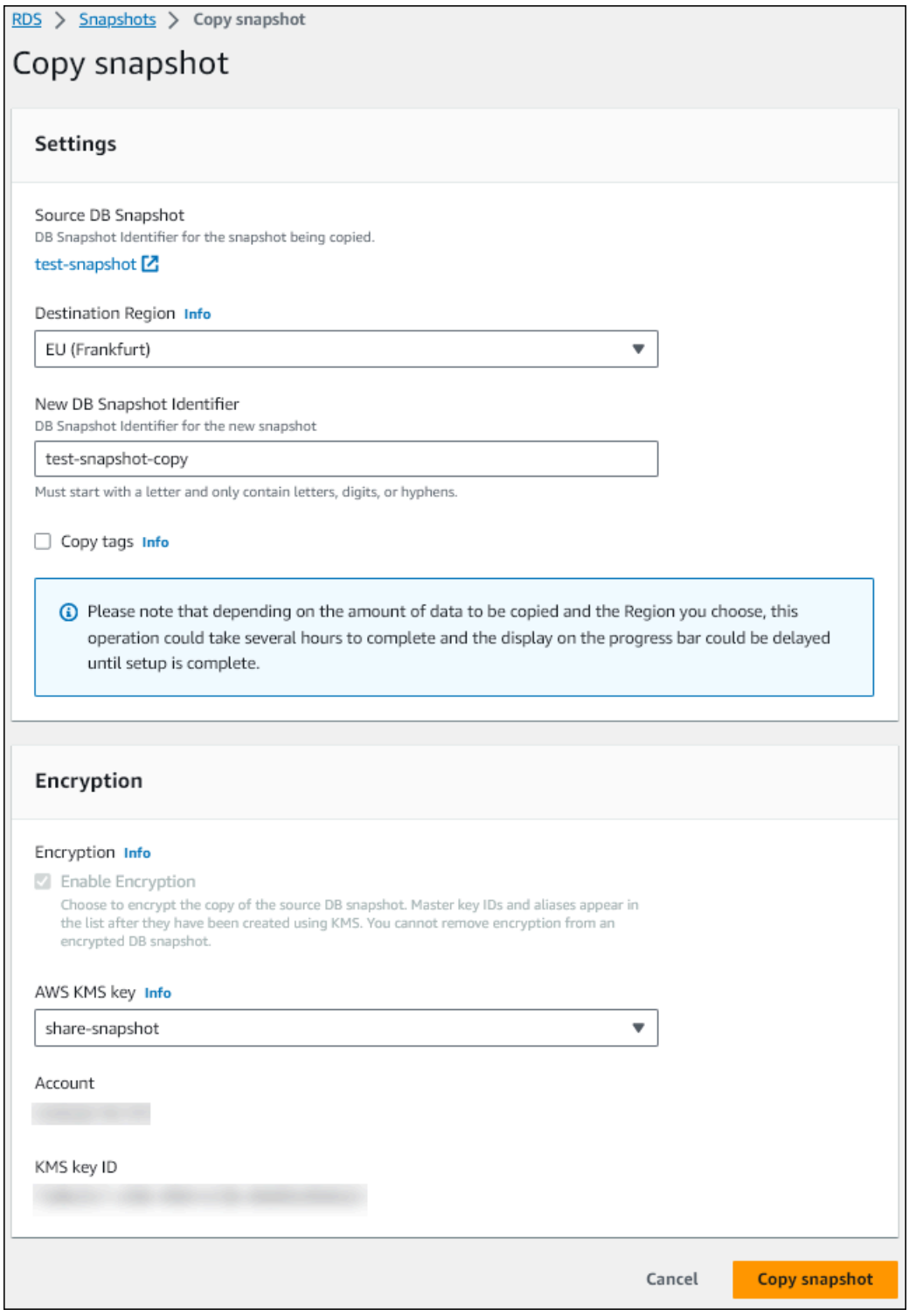
10. Tinjau kunci KMS Anda, lalu pilih Selesai.

Salin dan bagikan snapshot dari akun sumber

Selanjutnya Anda menyalin snapshot cluster DB sumber ke snapshot baru menggunakan kunci yang dikelola pelanggan. Kemudian Anda membagikannya dengan target Akun AWS.

Untuk menyalin dan membagikan snapshot

1. Masuk ke AWS Management Console dari sumbernya Akun AWS.
2. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/
3. Di panel navigasi, pilih Snapshot.
4. Pilih snapshot cluster DB yang ingin Anda salin.
5. Untuk Tindakan, pilih Salin snapshot.
6. Pada halaman Salin snapshot:
    a. Untuk Wilayah Tujuan, pilih Wilayah AWS tempat Anda membuat kunci yang dikelola pelanggan di prosedur sebelumnya.
    b. Masukkan nama salinan snapshot klaster DB dalam Pengidentifikasi Snapshot DB Baru.
    c. Untuk AWS KMS key, pilih kunci yang dikelola pelanggan yang Anda buat.



d. Pilih Salin snapshot.

7. Ketika salinan snapshot tersedia, pilih.
8. Untuk Tindakan, pilih Bagikan snapshot.
9. Pada halaman izin Snapshot:

a. Masukkan Akun AWS ID yang Anda gunakan untuk membagikan salinan snapshot, lalu pilih Tambah.

b. Pilih Simpan.

Snapshot dibagikan.

Salin snapshot bersama di akun target

Sekarang Anda dapat menyalin snapshot bersama di target Akun AWS.

Untuk menyalin snapshot bersama

1. Masuk ke AWS Management Console dari target Akun AWS.
2. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/
3. Di panel navigasi, pilih Snapshot.
4. Pilih tab Dibagikan dengan saya.
5. Pilih snapshot bersama.
6. Untuk Tindakan, pilih Salin snapshot.
7. Pilih pengaturan Anda untuk menyalin snapshot seperti pada prosedur sebelumnya, tetapi gunakan AWS KMS key yang termasuk dalam akun target.

   Pilih Salin snapshot.

Menghentikan berbagi snapshot

Untuk berhenti membagikan snapshot cluster DB, Anda menghapus izin dari target Akun AWS.

Konsol

Untuk berhenti berbagi snapshot cluster DB manual dengan Akun AWS

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih snapshot manual yang pembagiannya ingin Anda hentikan.

4. Pilih Tindakan, lalu pilih Bagikan snapshot.
5. Untuk menghapus izin untuk Akun AWS, pilih Hapus untuk pengenal AWS akun untuk akun tersebut dari daftar akun resmi.
6. Pilih Simpan untuk menyimpan perubahan Anda.

CLI

Untuk menghapus Akun AWS pengenal dari daftar, gunakan `--values-to-remove` parameter.

Example menghentikan berbagi snapshot

Contoh berikut mencegah Akun AWS ID 444455556666 memulihkan snapshot.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-remove 444455556666
```

Untuk Windows:

```
aws rds modify-db-cluster-snapshot-attribute ^
--db-cluster-snapshot-identifier manual-cluster-snapshot1 ^
--attribute-name restore ^
--values-to-remove 444455556666
```

API RDS

Untuk menghapus izin berbagi untuk sebuah Akun AWS, gunakan `ModifyDBClusterSnapshotAttribute`operasi dengan `AttributeName` set to `restore` dan `ValuesToRemove` parameter. Untuk menandai snapshot manual sebagai privat, hapus nilai `all` dari daftar nilai untuk atribut `restore`.

Mengekspor data klaster DB ke Amazon S3

Anda dapat mengekspor data dari klaster DB Amazon Aurora langsung ke bucket Amazon S3. Proses ekspor berjalan di latar belakang dan tidak memengaruhi performa klaster DB aktif Anda.

Secara default, semua data dalam klaster DB diekspor. Namun, Anda dapat memilih untuk mengekspor set basis data, skema, atau tabel tertentu.

Amazon Aurora mengklona klaster DB, mengekstrak data dari klona, dan menyimpan data dalam bucket Amazon S3. Data disimpan dalam format Apache Parquet yang dikompresi dan konsisten. Setiap file Parquet biasanya berukuran 1–10 MB.

Performa lebih cepat yang bisa Anda dapatkan dengan mengekspor data snapshot untuk Aurora MySQL versi 2 dan versi 3 tidak berlaku untuk mengekspor data klaster DB. Untuk informasi selengkapnya, lihat Mengekspor data snapshot klaster DB ke Amazon S3.

Anda dikenakan biaya untuk mengekspor seluruh klaster DB, baik Anda mengekspor semua atau sebagian data. Untuk informasi selengkapnya, lihat Halaman Harga Amazon Aurora.

Setelah data diekspor, Anda dapat menganalisis data yang diekspor secara langsung melalui alat seperti Amazon Athena atau Amazon Redshift Spectrum. Untuk informasi lebih lanjut tentang menggunakan Athena untuk membaca data Parket, lihat Parket di Panduan Pengguna SerDe Amazon Athena. Untuk informasi selengkapnya tentang cara menggunakan Redshift Spectrum untuk membaca data Parquet, lihat COPY dari format data berkolom dalam Panduan Developer Basis Data Amazon Redshift.

Ketersediaan dan dukungan fitur bervariasi di semua versi spesifik setiap mesin basis data dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang versi dan ketersediaan Wilayah pengeksporan data klaster DB ke S3, lihat Mengekspor data klaster ke Amazon S3.

Topik

- Batasan
- Ikhtisar pengeksporan data klaster DB
- Menyiapkan akses ke bucket Amazon S3
- Mengekspor data klaster DB ke bucket Amazon S3
- Memantau tugas ekspor klaster DB
- Membatalkan tugas ekspor klaster DB

- Pesan kegagalan untuk tugas ekspor Amazon S3
- Memecahkan masalah kesalahan izin PostgreSQL
- Konvensi penamaan file
- Konversi data dan format penyimpanan

Batasan

Mengekspor data klaster DB ke Amazon S3 memiliki batasan sebagai berikut:

- Anda tidak dapat menjalankan beberapa tugas ekspor untuk klaster DB yang sama secara bersamaan. Ini berlaku untuk ekspor penuh dan sebagian.
- Klaster DB Aurora Serverless v1 tidak mendukung ekspor ke S3.
- Aurora MySQL dan Aurora PostgreSQL mendukung ekspor ke S3 hanya untuk mode mesin yang disediakan.
- Ekspor ke S3 tidak mendukung awalan S3 yang berisi titik dua (:).
- Karakter berikut di jalur file S3 akan diubah menjadi garis bawah (_) selama ekspor berlangsung:

```
\ ` " (space)
```

- Jika basis data, skema, atau tabel memiliki karakter dalam namanya selain yang berikut ini, maka ekspor parsial tidak didukung. Namun, Anda dapat mengekspor seluruh klaster DB.
  - Huruf latin (A–Z)
  - Digit (0–9)
  - Simbol dolar ($)
  - Garis bawah (_)
- Spasi ( ) dan karakter-karakter tertentu tidak didukung dalam nama kolom tabel basis data. Tabel yang nama kolomnya berisi karakter berikut akan dilewati selama ekspor berlangsung:

```
, ; { } ( ) \n \t = (space)
```

- Tabel yang namanya berisi garis miring (/) akan dilewati selama ekspor berlangsung.
- Tabel Aurora PostgreSQL yang bersifat sementara dan tidak tercatat akan dilewati selama ekspor berlangsung.
- Jika data berisi objek besar, seperti BLOB atau CLOB, yang berukuran mendekati atau lebih dari 500 MB, maka ekspornya akan gagal.

- Jika suatu tabel berisi baris besar yang berukuran mendekati atau lebih dari 2 GB, maka tabel tersebut akan dilewati selama ekspor berlangsung.
- Sebaiknya Anda menggunakan nama unik untuk setiap tugas ekspor. Jika tidak menggunakan nama tugas yang unik, Anda mungkin menerima pesan kesalahan berikut:

  ExportTaskAlreadyExistsFault: Terjadi kesalahan (ExportTaskAlreadyExists) saat memanggil StartExportTask operasi: Tugas ekspor dengan ID *xxxxx* sudah ada.
- Karena beberapa tabel mungkin dilewati, sebaiknya Anda memverifikasi jumlah baris dan tabel dalam data setelah ekspor.

Ikhtisar pengeksporan data klaster DB

Anda menggunakan proses berikut untuk mengekspor data klaster DB ke bucket Amazon S3. Untuk detail selengkapnya, lihat bagian berikut.

1. Identifikasi klaster DB yang datanya ingin Anda ekspor.
2. Atur akses ke bucket Amazon S3.

   Bucket adalah kontainer untuk objek atau file Amazon S3. Untuk memberikan informasi untuk mengakses bucket, lakukan langkah-langkah berikut:

   a. Identifikasi bucket S3 tempat data klaster DB akan diekspor. Bucket S3 harus berada di Wilayah AWS yang sama dengan klaster DB. Untuk informasi selengkapnya, lihat Mengidentifikasi bucket Amazon S3 untuk ekspor.

   b. Buat peran AWS Identity and Access Management (IAM) yang memberikan akses tugas ekspor klaster DB ke bucket S3. Untuk informasi selengkapnya, lihat Memberikan akses ke bucket Amazon S3 menggunakan peran IAM.
3. Buat enkripsi simetris AWS KMS key untuk enkripsi sisi server. Kunci KMS digunakan oleh tugas ekspor cluster untuk mengatur enkripsi AWS KMS sisi server saat menulis data ekspor ke S3.

   Kebijakan kunci KMS harus menyertakan izin `kms:DescribeKey` dan `kms:CreateGrant`. Untuk informasi selengkapnya tentang cara menggunakan kunci KMS di Amazon Aurora, lihat Manajemen AWS KMS key.

   Jika Anda memiliki pernyataan penolakan dalam kebijakan kunci KMS Anda, pastikan untuk secara eksplisit mengecualikan prinsipal layanan. AWS `export.rds.amazonaws.com`

Anda dapat menggunakan kunci KMS dalam AWS akun Anda, atau Anda dapat menggunakan kunci KMS lintas akun. Untuk informasi selengkapnya, lihat Menggunakan cross-account AWS KMS key.

4. Ekspor klaster DB ke Amazon S3 menggunakan konsol atau perintah CLI `start-export-task`. Untuk informasi selengkapnya, lihat Mengekspor data klaster DB ke bucket Amazon S3.
5. Untuk mengakses data Anda yang diekspor di bucket Amazon S3 lihat Mengunggah, mengunduh, dan mengelola objek dalam Panduan Pengguna Amazon Simple Storage Service.

Menyiapkan akses ke bucket Amazon S3

Anda mengidentifikasi bucket Amazon S3, lalu memberikan izin tugas ekspor klaster DB untuk mengaksesnya.

Topik

- Mengidentifikasi bucket Amazon S3 untuk ekspor
- Memberikan akses ke bucket Amazon S3 menggunakan peran IAM
- Menggunakan bucket Amazon S3 lintas akun

Mengidentifikasi bucket Amazon S3 untuk ekspor

Identifikasi bucket Amazon S3 untuk mengekspor data klaster DB tujuan. Gunakan bucket S3 yang ada atau buat bucket S3 baru.

> **Note**
>
> Bucket S3 harus berada di AWS Region yang sama dengan cluster DB.

Untuk informasi selengkapnya tentang cara bekerja dengan bucket Amazon S3, lihat informasi berikut dalam Panduan Pengguna Amazon Simple Storage Service:

- Bagaimana cara melihat properti untuk bucket S3?
- Bagaimana cara mengaktifkan enkripsi default untuk bucket Amazon S3?
- Bagaimana cara membuat bucket S3?

Memberikan akses ke bucket Amazon S3 menggunakan peran IAM

Sebelum Anda mengekspor data klaster DB ke Amazon S3, berikan izin akses tulis tugas ekspor ke bucket Amazon S3.

Untuk memberikan izin ini, buat kebijakan IAM yang memberikan akses ke bucket, lalu buat peran IAM dan lampirkan kebijakan ke peran tersebut. Setelah itu, Anda dapat menetapkan peran IAM ke tugas ekspor klaster DB.

> **Important**
>
> Jika Anda berencana untuk menggunakan klaster AWS Management Console untuk mengekspor DB Anda, Anda dapat memilih untuk membuat kebijakan IAM dan peran secara otomatis ketika Anda mengekspor cluster DB. Untuk petunjuknya, lihat Mengekspor data klaster DB ke bucket Amazon S3.

Untuk memberikan akses tugas ke Amazon S3

1. Buat kebijakan IAM. Kebijakan ini memberikan bucket dan izin objek yang memungkinkan tugas ekspor klaster DB Anda mengakses Amazon S3.

   Dalam kebijakan tersebut, sertakan tindakan yang diperlukan berikut untuk mengizinkan transfer file dari Amazon Aurora ke bucket S3:

   - `s3:PutObject*`
   - `s3:GetObject*`
   - `s3:ListBucket`
   - `s3:DeleteObject*`
   - `s3:GetBucketLocation`

   Dalam kebijakan tersebut, sertakan sumber daya berikut untuk mengidentifikasi bucket S3 dan objek dalam bucket. Daftar sumber daya berikut menunjukkan format Amazon Resource Name (ARN) untuk mengakses Amazon S3.

   - `arn:aws:s3:::`*`your-s3-bucket`*
   - `arn:aws:s3:::`*`your-s3-bucket`*`/*`

Untuk mengetahui informasi selengkapnya tentang cara membuat kebijakan IAM untuk Amazon Aurora, lihat Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM. Lihat juga Tutorial: Membuat dan melampirkan kebijakan yang dikelola pelanggan pertama Anda di Panduan Pengguna IAM.

AWS CLI Perintah berikut membuat kebijakan IAM bernama `ExportPolicy` dengan opsi ini. Perintah ini akan memberikan akses ke bucket bernama `your-s3-bucket`.

> **Note**
>
> Setelah Anda membuat kebijakan, catat ARN kebijakan tersebut. Anda memerlukan ARN ini untuk langkah berikutnya saat Anda melampirkan kebijakan tersebut pada peran IAM.

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::your-s3-bucket",
                "arn:aws:s3:::your-s3-bucket/*"
            ]
        }
    ]
}'
```

2. Buat peran IAM agar Aurora dapat mengambil peran IAM ini atas nama Anda untuk mengakses bucket Amazon S3 Anda. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin kepada pengguna IAM dalam Panduan Pengguna IAM.

Contoh berikut menunjukkan menggunakan AWS CLI perintah untuk membuat peran bernama`rds-s3-export-role`.

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
           "Service": "export.rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole"
      }
    ]
  }'
```

3. Lampirkan kebijakan IAM yang Anda buat pada peran IAM yang Anda buat.

   AWS CLI Perintah berikut melampirkan kebijakan yang dibuat sebelumnya ke peran bernama`rds-s3-export-role`. Ganti *`your-policy-arn`* dengan ARN kebijakan yang Anda catat di langkah sebelumnya.

   ```
   aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
   export-role
   ```

Menggunakan bucket Amazon S3 lintas akun

Anda dapat menggunakan bucket S3 di seluruh AWS akun. Untuk informasi selengkapnya, lihat Menggunakan bucket Amazon S3 lintas akun.

Mengekspor data klaster DB ke bucket Amazon S3

Anda dapat memiliki hingga lima tugas ekspor klaster DB bersamaan yang sedang berlangsung per Akun AWS.

> **Note**
>
> Mengekspor data klaster DB dapat memakan waktu, tergantung jenis dan ukuran basis data Anda. Tugas ekspor mengklona dan menskalakan seluruh basis data terlebih dahulu

sebelum mengekstrak data ke Amazon S3. Progres tugas selama fase ini ditampilkan sebagai Memulai. Saat tugas beralih menjadi mengekspor data ke S3, progres akan ditampilkan sebagai Sedang berlangsung.
Waktu yang diperlukan untuk menyelesaikan ekspor tergantung pada data yang disimpan di basis data. Misalnya, tabel dengan kunci primer numerik atau kolom indeks terdistribusi dengan baik mengekspor paling cepat. Tabel yang tidak berisi kolom yang sesuai untuk partisi dan tabel dengan hanya satu indeks pada kolom berbasis string membutuhkan waktu lebih lama karena pengeksporan menggunakan proses single-threaded yang lebih lambat.

Anda dapat mengekspor data cluster DB ke Amazon S3 menggunakan AWS Management Console, API AWS CLI, atau RDS.

Jika Anda menggunakan fungsi Lambda untuk mengekspor klaster DB, tambahkan tindakan `kms:DescribeKey` terhadap kebijakan fungsi Lambda. Untuk informasi selengkapnya, lihat izin AWS Lambda.

Konsol

Opsi konsol Ekspor ke Amazon S3 hanya muncul untuk klaster DB yang dapat diekspor ke Amazon S3. Klaster DB mungkin tidak tersedia untuk diekspor karena alasan berikut:

- Mesin DB tidak didukung untuk ekspor S3.
- Versi klaster DB tidak didukung untuk ekspor S3.
- Ekspor S3 tidak didukung di AWS Wilayah tempat cluster DB dibuat.

Untuk mengekspor data klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB yang datanya ingin Anda ekspor.
4. Untuk Tindakan, pilih Ekspor ke Amazon S3.

   Jendela Ekspor ke Amazon S3 akan muncul.
5. Untuk Pengidentifikasi ekspor, masukkan nama untuk mengidentifikasi tugas ekspor. Nilai ini juga akan digunakan untuk nama file yang dibuat di bucket S3.

6. Pilih data yang akan diekspor:

   - Pilih Semua untuk mengekspor semua data dalam klaster DB.
   - Pilih Parsial untuk mengekspor bagian tertentu dari klaster DB. Untuk mengidentifikasi bagian klaster mana yang akan diekspor, masukkan satu atau beberapa basis data, skema, atau tabel untuk Pengidentifikasi, dipisahkan dengan spasi.

     Gunakan format berikut:

     ```
     database[.schema][.table] database2[.schema2][.table2] ... databasen[.schemaN]
     [.tablen]
     ```

     Contohnya:

     ```
     mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
      mydatabase2.myschema2.mytable2
     ```

7. Untuk Bucket S3, pilih bucket yang akan dijadikan tujuan ekspor.

   Untuk menetapkan data yang diekspor ke jalur folder dalam bucket S3, masukkan jalur opsional untuk Prefiks S3.

8. Untuk Peran IAM, pilih peran yang memberi Anda akses tulis ke bucket S3 yang Anda pilih, atau buat peran baru.

   - Jika Anda membuat peran dengan mengikuti langkah-langkah di Memberikan akses ke bucket Amazon S3 menggunakan peran IAM, pilih peran tersebut.
   - Jika Anda tidak membuat peran yang memberi Anda akses tulis ke bucket S3 yang Anda pilih, pilih Buat peran baru untuk membuat peran secara otomatis. Selanjutnya, masukkan nama untuk peran dalam Nama peran IAM.

9. Untuk KMS key, masukkan ARN untuk kunci yang akan digunakan untuk mengenkripsi data yang diekspor.

10. Pilih Ekspor ke Amazon S3.

AWS CLI

Untuk mengekspor data cluster DB ke Amazon S3 menggunakan AWS CLI, gunakan start-export-taskperintah dengan opsi yang diperlukan berikut:

- `--export-task-identifier`
- `--source-arn` – Amazon Resource Name (ARN) klaster DB
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

Dalam contoh berikut, tugas ekspor diberi nama *`my-cluster-export`*, yang mengekspor data ke bucket S3 bernama. *`my-export-bucket`*

Example

Untuk Linux, macOS, atau Unix:

```
aws rds start-export-task \
    --export-task-identifier my-cluster-export \
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster \
    --s3-bucket-name my-export-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

Untuk Windows:

```
aws rds start-export-task ^
    --export-task-identifier my-DB-cluster-export ^
    --source-arn arn:aws:rds:us-west-2:123456789012:cluster:my-cluster ^
    --s3-bucket-name my-export-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

Berikut adalah contoh output.

```
{
    "ExportTaskIdentifier": "my-cluster-export",
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:my-cluster",
    "S3Bucket": "my-export-bucket",
    "IamRoleArn": "arn:aws:iam:123456789012:role/ExportTest",
    "KmsKeyId": "my-key",
    "Status": "STARTING",
    "PercentProgress": 0,
    "TotalExtractedDataInGB": 0,
```

```
}
```

Untuk menyediakan jalur folder di bucket S3 untuk ekspor cluster DB, sertakan `--s3-prefix` opsi dalam start-export-taskperintah.

API RDS

Untuk mengekspor data cluster DB ke Amazon S3 menggunakan Amazon RDS API, gunakan StartExportTaskoperasi dengan parameter yang diperlukan berikut:

- `ExportTaskIdentifier`
- `SourceArn` – ARN klaster DB
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

Memantau tugas ekspor klaster DB

Anda dapat memantau ekspor cluster DB menggunakan AWS Management Console, AWS CLI, atau RDS API.

Konsol

Untuk memantau ekspor klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Ekspor di Amazon S3.

   Ekspor klaster DB ditunjukkan di kolom Tipe sumber. Status ekspor ditampilkan di kolom Status.
3. Untuk melihat informasi detail tentang ekspor klaster khusus, pilih tugas ekspor.

AWS CLI

Untuk memantau tugas ekspor cluster DB menggunakan AWS CLI, gunakan describe-export-tasksperintah.

Contoh berikut menunjukkan cara menampilkan informasi saat ini tentang semua ekspor klaster DB Anda.

Example

```
aws rds describe-export-tasks

{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2022-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "S3Bucket": "examplebucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:parameter-groups-
test"
        },
{
            "Status": "COMPLETE",
            "TaskStartTime": "2022-10-31T20:58:06.998Z",
            "TaskEndTime": "2022-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "S3Bucket": "examplebucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2022-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket examplebucket2 isn't located in the current
 AWS Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "S3Bucket": "examplebucket2",
```

```
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "SourceArn": "arn:aws:rds:us-
west-2:123456789012:cluster:example-1-2019-10-30-06-45"
        }
    ]
}
```

Untuk menampilkan informasi tentang ekspor khusus, sertakan opsi `--export-task-identifier` dengan perintah `describe-export-tasks`. Sertakan opsi `--Filters` untuk memfilter output. Untuk opsi lainnya, lihat describe-export-tasksperintah.

API RDS

Untuk menampilkan informasi tentang ekspor cluster DB menggunakan Amazon RDS API, gunakan operasi. DescribeExportTasks

Untuk melacak penyelesaian alur kerja ekspor atau memulai alur kerja lainnya, Anda dapat berlangganan topik Amazon Simple Notification Service. Untuk mengetahui informasi selengkapnya tentang Amazon SNS, lihat Bekerja dengan pemberitahuan peristiwa Amazon RDS.

Membatalkan tugas ekspor klaster DB

Anda dapat membatalkan tugas ekspor klaster DB menggunakan AWS Management Console, API AWS CLI, atau RDS.

> **Note**
>
> Membatalkan tugas ekspor tidak menghapus data apa pun yang diekspor ke Amazon S3. Untuk mengetahui informasi tentang cara menghapus data menggunakan konsol, lihat Bagaimana cara menghapus objek dari bucket S3? Untuk menghapus data menggunakan CLI, gunakan perintah delete-object.

Konsol

Untuk membatalkan tugas ekspor klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Ekspor di Amazon S3.

   Ekspor klaster DB ditunjukkan di kolom Tipe sumber. Status ekspor ditampilkan di kolom Status.
3. Pilih tugas ekspor yang ingin Anda batalkan.
4. Pilih Batalkan.
5. Pilih Batalkan tugas ekspor di halaman konfirmasi.

AWS CLI

Untuk membatalkan tugas ekspor menggunakan AWS CLI, gunakan cancel-export-taskperintah. Perintah tersebut memerlukan opsi `--export-task-identifier`.

Example

```
aws rds cancel-export-task --export-task-identifier my-export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "S3Bucket": "examplebucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:us-west-2:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my-export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "SourceArn": "arn:aws:rds:us-west-2:123456789012:cluster:export-example-1"
}
```

API RDS

Untuk membatalkan tugas ekspor menggunakan Amazon RDS API, gunakan CancelExportTaskoperasi dengan `ExportTaskIdentifier` parameter.

Pesan kegagalan untuk tugas ekspor Amazon S3

Tabel berikut menjelaskan pesan yang akan ditampilkan jika tugas ekspor Amazon S3 gagal.

Pesan kegagalan	Deskripsi
Gagal menemukan atau mengakses klaster DB sumber: [nama klaster]	Klaster DB sumber tidak dapat diklona.
Terjadi kesalahan internal yang tidak diketahui.	Tugas telah gagal karena kesalahan yang tidak diketahui, pengecualian, atau kegagalan.
Terjadi kesalahan internal yang tidak diketahui saat menulis metadata tugas ekspor ke bucket S3 [nama bucket].	Tugas telah gagal karena kesalahan yang tidak diketahui, pengecualian, atau kegagalan.
Ekspor RDS gagal menulis metadata tugas ekspor karena tidak dapat mengambil peran IAM [peran ARN].	Tugas ekspor mengambil peran IAM Anda untuk memvalidasi apakah tugas tersebut diperbolehkan menulis metadata ke bucket S3 Anda. Jika tidak dapat mengambil peran IAM Anda, berarti tugas tersebut gagal.
Ekspor RDS gagal menulis metadata tugas ekspor ke bucket S3 [nama bucket] menggunakan peran IAM [ARN peran] dengan kunci KMS [ID kunci]. Kode kesalahan: [kode kesalahan]	Satu atau beberapa izin tidak ada, sehingga tugas ekspor tidak dapat mengakses bucket S3. Pesan kegagalan ini muncul saat menerima salah satu kode kesalahan berikut: • `AWSSecurityTokenServiceException` dengan kode kesalahan `AccessDenied` • `AmazonS3Exception` dengan kode kesalahan `NoSuchBucket`, `AccessDenied`, `KMS.KMSInvalidStateException`, `403 Forbidden`, atau `KMS.DisabledException` Kode kesalahan ini menunjukkan bahwa pengaturan salah dikonfigurasi untuk peran IAM, bucket S3, atau kunci KMS.

Pesan kegagalan	Deskripsi
Peran IAM [ARN peran] tidak diizinkan untuk memanggil [tindakan S3] pada bucket S3 [nama bucket]. Tinjau izin Anda dan coba lagi ekspor.	Kebijakan IAM salah dikonfigurasi. Izin untuk tindakan S3 tertentu pada bucket S3 tidak ada, sehingga menyebabkan tugas ekspor gagal.
Pemeriksaan kunci KMS gagal. Periksa kredensial pada kunci KMS Anda, lalu coba lagi.	Pemeriksaan kredensial kunci KMS gagal.
Pemeriksaan kredensial S3 gagal. Periksa izin pada bucket S3 dan kebijakan IAM Anda.	Pemeriksaan kredensial S3 gagal.
Bucket S3 [nama bucket] tidak valid. Bucket tersebut tidak berada di Wilayah AWS saat ini atau tidak ada. Tinjau nama bucket S3 Anda, lalu coba lagi ekspor.	Bucket S3 tidak valid.
Bucket S3 [nama bucket] tidak berada di Wilayah AWS saat ini. Tinjau nama bucket S3 Anda, lalu coba lagi ekspor.	Ember S3 salah Wilayah AWS.

Memecahkan masalah kesalahan izin PostgreSQL

Saat mengekspor basis data PostgreSQL ke Amazon S3, Anda mungkin melihat kesalahan `PERMISSIONS_DO_NOT_EXIST` yang menyatakan bahwa tabel tertentu dilewati. Kesalahan ini biasanya terjadi saat superuser, yang Anda tetapkan saat membuat klaster DB, tidak memiliki izin untuk mengakses tabel tersebut.

Untuk memperbaiki kesalahan ini, jalankan perintah berikut:

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

Untuk informasi selengkapnya tentang hak pengguna super, lihat Hak akses akun pengguna master.

Konvensi penamaan file

Data yang diekspor untuk tabel tertentu disimpan dalam format *base_prefix*/*files*, dengan prefiks dasar sebagai berikut:

```
export_identifier/database_name/schema_name.table_name/
```

Contohnya:

```
export-1234567890123-459/rdststcluster/mycluster.DataInsert_7ADB5D19965123A2/
```

File output menggunakan konvensi penamaan berikut, di mana *partition_index* bersifat alfanumerik:

```
partition_index/part-00000-random_uuid.format-based_extension
```

Sebagai contoh:

```
1/part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
a/part-00000-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
```

Konvensi penamaan file dapat berubah sewaktu-waktu. Oleh karena itu, ketika membaca tabel target, sebaiknya baca segala sesuatu di dalam prefiks dasar untuk tabel.

Konversi data dan format penyimpanan

Saat Anda mengekspor klaster DB ke bucket Amazon S3, Amazon Aurora mengonversi data ke, mengekspor data dalam, dan menyimpan data dalam format Parquet. Untuk informasi selengkapnya, lihat Konversi data saat mengekspor ke bucket Amazon S3.

Mengekspor data snapshot klaster DB ke Amazon S3

Anda dapat mengekspor data snapshot klaster DB ke bucket Amazon S3. Proses ekspor berjalan di latar belakang dan tidak memengaruhi performa klaster DB aktif Anda.

Saat Anda mengekspor snapshot klaster DB, Amazon Aurora akan mengekstrak data dari snapshot tersebut dan menyimpannya di bucket Amazon S3. Anda dapat mengekspor snapshot manual dan snapshot sistem otomatis. Secara default, semua data dalam snapshot akan diekspor. Namun, Anda dapat memilih untuk mengekspor set basis data, skema, atau tabel tertentu.

Data disimpan dalam format Apache Parquet yang dikompresi dan konsisten. Setiap file Parquet biasanya berukuran 1–10 MB.

Setelah data diekspor, Anda dapat menganalisis data yang diekspor secara langsung melalui alat seperti Amazon Athena atau Amazon Redshift Spectrum. Untuk informasi lebih lanjut tentang menggunakan Athena untuk membaca data Parket, lihat Parket di Panduan Pengguna SerDe Amazon Athena. Untuk informasi selengkapnya tentang cara menggunakan Redshift Spectrum untuk membaca data Parquet, lihat COPY dari format data berkolom dalam Panduan Developer Basis Data Amazon Redshift.

Ketersediaan dan dukungan fitur bervariasi di semua versi spesifik setiap mesin basis data dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang ketersediaan versi dan Wilayah untuk mengekspor data snapshot klaster DB ke S3, lihat Mengekspor data snapshot ke Amazon S3.

Topik

- Batasan
- Ringkasan pengeksporan data snapshot
- Menyiapkan akses ke bucket Amazon S3
- Mengekspor snapshot ke bucket Amazon S3
- Performa ekspor di Aurora MySQL
- Memantau ekspor snapshot
- Membatalkan tugas ekspor snapshot
- Pesan kegagalan untuk tugas ekspor Amazon S3
- Memecahkan masalah kesalahan izin PostgreSQL
- Konvensi penamaan file
- Konversi data saat mengekspor ke bucket Amazon S3

Batasan

Batasan berikut berlaku untuk ekspor data snapshot DB ke Amazon S3:

- Anda tidak dapat menjalankan beberapa tugas ekspor untuk snapshot klaster DB yang sama secara bersamaan. Batasan ini berlaku untuk ekspor penuh dan sebagian.
- Anda tidak dapat mengekspor data snapshot dari cluster Aurora Serverless v1 DB ke S3.
- Ekspor ke S3 tidak mendukung awalan S3 yang berisi titik dua (:).
- Karakter berikut di jalur file S3 akan diubah menjadi garis bawah (_) selama ekspor berlangsung:

  ```
  \ ` " (space)
  ```

- Jika basis data, skema, atau tabel memiliki karakter dalam namanya selain yang berikut ini, maka ekspor parsial tidak didukung. Namun, Anda dapat mengekspor seluruh snapshot DB.
  - Huruf latin (A-Z)
  - Digit (0–9)
  - Simbol dolar ($)
  - Garis bawah (_)
- Spasi ( ) dan karakter-karakter tertentu tidak didukung dalam nama kolom tabel basis data. Tabel yang nama kolomnya berisi karakter berikut akan dilewati selama ekspor berlangsung:

  ```
  , ; { } ( ) \n \t = (space)
  ```

- Tabel yang namanya berisi garis miring (/) akan dilewati selama ekspor berlangsung.
- Tabel Aurora PostgreSQL yang bersifat sementara dan tidak tercatat akan dilewati selama ekspor berlangsung.
- Jika data berisi objek besar, seperti BLOB atau CLOB, yang berukuran mendekati atau lebih dari 500 MB, maka ekspornya akan gagal.
- Jika suatu tabel berisi baris besar yang berukuran mendekati atau lebih dari 2 GB, maka tabel tersebut akan dilewati selama ekspor berlangsung.
- Sebaiknya Anda menggunakan nama unik untuk setiap tugas ekspor. Jika tidak menggunakan nama tugas yang unik, Anda mungkin menerima pesan kesalahan berikut:

  ExportTaskAlreadyExistsFault: Terjadi kesalahan (ExportTaskAlreadyExists) saat memanggil StartExportTask operasi: Tugas ekspor dengan ID *xxxxx* sudah ada.

- Anda dapat menghapus snapshot saat sedang mengekspor datanya ke S3, tetapi Anda masih dikenai biaya penyimpanan untuk snapshot tersebut hingga tugas ekspor selesai.
- Anda tidak dapat memulihkan data snapshot yang diekspor dari S3 ke klaster DB baru.

Ringkasan pengeksporan data snapshot

Anda menggunakan proses berikut untuk mengekspor data snapshot DB ke bucket Amazon S3. Untuk detail selengkapnya, lihat bagian berikut.

1. Identifikasi snapshot yang akan diekspor.

   Gunakan snapshot otomatis atau manual yang ada, atau buat snapshot manual instans DB.
2. Siapkan akses ke bucket Amazon S3.

   Bucket adalah kontainer untuk objek atau file Amazon S3. Untuk memberikan informasi agar dapat mengakses bucket, lakukan langkah-langkah berikut:

   a. Identifikasi bucket S3 tempat snapshot akan diekspor. Bucket S3 harus berada di AWS Wilayah yang sama dengan snapshot. Untuk informasi selengkapnya, lihat Mengidentifikasi bucket Amazon S3 untuk ekspor.

   b. Buat peran AWS Identity and Access Management (IAM) yang memberikan akses tugas ekspor snapshot ke bucket S3. Untuk informasi selengkapnya, lihat Memberikan akses ke bucket Amazon S3 menggunakan peran IAM.
3. Buat enkripsi simetris AWS KMS key untuk enkripsi sisi server. Kunci KMS digunakan oleh tugas ekspor snapshot untuk mengatur enkripsi AWS KMS sisi server saat menulis data ekspor ke S3.

   Kebijakan kunci KMS harus menyertakan izin `kms:DescribeKey` dan `kms:CreateGrant`. Untuk informasi selengkapnya tentang cara menggunakan kunci KMS di Amazon Aurora, lihat Manajemen AWS KMS key.

   Jika Anda memiliki pernyataan penolakan dalam kebijakan kunci KMS Anda, pastikan untuk secara eksplisit mengecualikan prinsip layanan. AWS `export.rds.amazonaws.com`

   Anda dapat menggunakan kunci KMS dalam AWS akun Anda, atau Anda dapat menggunakan kunci KMS lintas akun. Untuk informasi selengkapnya, lihat Menggunakan cross-account AWS KMS key.
4. Ekspor snapshot ke Amazon S3 menggunakan konsol atau perintah CLI `start-export-task`. Untuk informasi selengkapnya, lihat Mengekspor snapshot ke bucket Amazon S3.

5. Untuk mengakses data Anda yang diekspor di bucket Amazon S3 lihat Mengunggah, mengunduh, dan mengelola objek dalam Panduan Pengguna Amazon Simple Storage Service.

Menyiapkan akses ke bucket Amazon S3

Identifikasi bucket Amazon S3, lalu berikan izin snapshot untuk mengaksesnya.

Topik

- Mengidentifikasi bucket Amazon S3 untuk ekspor
- Memberikan akses ke bucket Amazon S3 menggunakan peran IAM
- Menggunakan bucket Amazon S3 lintas akun
- Menggunakan cross-account AWS KMS key

Mengidentifikasi bucket Amazon S3 untuk ekspor

Identifikasi bucket Amazon S3 untuk mengekspor snapshot DB. Gunakan bucket S3 yang ada atau buat bucket S3 baru.

> **Note**
>
> Bucket S3 yang akan diekspor harus berada di AWS Wilayah yang sama dengan snapshot.

Untuk informasi selengkapnya tentang cara bekerja dengan bucket Amazon S3, lihat informasi berikut dalam Panduan Pengguna Amazon Simple Storage Service:

- Bagaimana cara melihat properti untuk bucket S3?
- Bagaimana cara mengaktifkan enkripsi default untuk bucket Amazon S3?
- Bagaimana cara membuat bucket S3?

Memberikan akses ke bucket Amazon S3 menggunakan peran IAM

Sebelum Anda mengekspor data snapshot DB ke Amazon S3, beri tugas ekspor snapshot izin akses tulis ke bucket Amazon S3.

Untuk memberikan izin ini, buat kebijakan IAM yang memberikan akses ke bucket, lalu buat peran IAM dan lampirkan kebijakan pada peran tersebut. Setelah itu, Anda dapat menetapkan peran IAM ke tugas ekspor snapshot Anda.

> **Important**
>
> Jika Anda berencana untuk menggunakan AWS Management Console untuk mengekspor snapshot Anda, Anda dapat memilih untuk membuat kebijakan IAM dan peran secara otomatis ketika Anda mengekspor snapshot. Untuk mendapatkan petunjuk, lihat Mengekspor snapshot ke bucket Amazon S3.

Untuk memberi tugas snapshot DB akses ke Amazon S3

1. Buat kebijakan IAM. Kebijakan ini memberi bucket dan objek izin yang memungkinkan tugas ekspor snapshot Anda mengakses Amazon S3.

   Dalam kebijakan tersebut, sertakan tindakan yang diperlukan berikut untuk memungkinkan transfer file dari Amazon Aurora ke bucket S3:

   - `s3:PutObject*`
   - `s3:GetObject*`
   - `s3:ListBucket`
   - `s3:DeleteObject*`
   - `s3:GetBucketLocation`

   Dalam kebijakan tersebut, sertakan sumber daya berikut untuk mengidentifikasi bucket S3 dan objek dalam bucket. Daftar sumber daya berikut menunjukkan format Amazon Resource Name (ARN) untuk mengakses Amazon S3.

   - `arn:aws:s3:::`*`your-s3-bucket`*
   - `arn:aws:s3:::`*`your-s3-bucket`*`/*`

   Untuk informasi selengkapnya tentang cara membuat kebijakan IAM untuk Amazon Aurora, lihat Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM. Lihat juga Tutorial: Membuat dan melampirkan kebijakan yang dikelola pelanggan pertama Anda dalam Panduan Pengguna IAM.

AWS CLI Perintah berikut membuat kebijakan IAM bernama `ExportPolicy` dengan opsi ini. Perintah ini akan memberikan akses ke bucket bernama `your-s3-bucket`.

> **Note**
>
> Setelah Anda membuat kebijakan, catat ARN kebijakan tersebut. Anda memerlukan ARN ini untuk langkah berikutnya saat Anda melampirkan kebijakan tersebut pada peran IAM.

```
aws iam create-policy  --policy-name ExportPolicy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ExportPolicy",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject*",
                "s3:ListBucket",
                "s3:GetObject*",
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::your-s3-bucket",
                "arn:aws:s3:::your-s3-bucket/*"
            ]
        }
    ]
}'
```

2. Buat peran IAM agar Aurora dapat mengambil peran IAM ini atas nama Anda untuk mengakses bucket Amazon S3 Anda. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin kepada pengguna IAM dalam Panduan Pengguna IAM.

   Contoh berikut menunjukkan menggunakan AWS CLI perintah untuk membuat peran bernama`rds-s3-export-role`.

```
aws iam create-role  --role-name rds-s3-export-role  --assume-role-policy-document
 '{
     "Version": "2012-10-17",
```

```
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
           "Service": "export.rds.amazonaws.com"
         },
        "Action": "sts:AssumeRole"
      }
    ]
  }'
```

3. Lampirkan kebijakan IAM yang Anda buat pada peran IAM yang Anda buat.

   AWS CLI Perintah berikut melampirkan kebijakan yang dibuat sebelumnya ke peran bernama`rds-s3-export-role`. Ganti *`your-policy-arn`* dengan ARN kebijakan yang Anda catat di langkah sebelumnya.

   ```
   aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
   export-role
   ```

Menggunakan bucket Amazon S3 lintas akun

Anda dapat menggunakan bucket Amazon S3 di seluruh akun. AWS Untuk menggunakan bucket lintas akun, tambahkan kebijakan bucket untuk mengizinkan akses ke peran IAM yang Anda gunakan untuk ekspor S3. Untuk informasi selengkapnya, lihat Contoh 2: Pemilik bucket yang memberikan izin bucket lintas akun.

- Lampirkan kebijakan bucket pada bucket Anda, seperti yang ditunjukkan dalam contoh berikut.

  ```
  {
      "Version": "2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Principal": {
                  "AWS": "arn:aws:iam::123456789012:role/Admin"
              },
              "Action": [
                  "s3:PutObject*",
                  "s3:ListBucket",
                  "s3:GetObject*",
  ```

```
                "s3:DeleteObject*",
                "s3:GetBucketLocation"
            ],
            "Resource": [
                "arn:aws:s3:::mycrossaccountbucket",
                "arn:aws:s3:::mycrossaccountbucket/*"
            ]
        }
    ]
}
```

Menggunakan cross-account AWS KMS key

Anda dapat menggunakan akun silang AWS KMS key untuk mengenkripsi ekspor Amazon S3. Pertama-tama, tambahkan kebijakan kunci ke akun lokal, lalu tambahkan kebijakan IAM di akun eksternal. Untuk informasi selengkapnya, lihat Mengizinkan pengguna di akun lain untuk menggunakan kunci KMS.

Untuk menggunakan kunci KMS lintas akun

1. Tambahkan kebijakan kunci ke akun lokal.

   Contoh berikut memberi `ExampleRole` dan `ExampleUser` di akun eksternal 444455556666 izin di akun lokal 123456789012.

```
{
    "Sid": "Allow an external account to use this KMS key",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::444455556666:role/ExampleRole",
            "arn:aws:iam::444455556666:user/ExampleUser"
        ]
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
```

```
            "kms:RetireGrant"
        ],
        "Resource": "*"
}
```

2. Tambahkan kebijakan IAM ke akun eksternal tersebut.

   Contoh kebijakan IAM berikut memungkinkan pengguna utama menggunakan kunci KMS di akun 123456789012 untuk operasi kriptografi. Untuk memberikan izin ini kepada `ExampleRole` dan `ExampleUser` di akun 444455556666, lampirkan kebijakan pada keduanya di akun tersebut.

```
{
    "Sid": "Allow use of KMS key in account 123456789012",
    "Effect": "Allow",
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey",
        "kms:RetireGrant"
    ],
    "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}
```

Mengekspor snapshot ke bucket Amazon S3

Anda dapat memiliki hingga lima tugas ekspor snapshot DB bersamaan yang sedang berlangsung per. Akun AWS

> **Note**
>
> Mengekspor snapshot RDS dapat memerlukan banyak waktu, tergantung pada jenis dan ukuran basis data Anda. Tugas ekspor akan terlebih dahulu memulihkan dan menskalakan seluruh basis data sebelum mengekstrak data ke Amazon S3. Dalam fase ini, progres tugas tersebut akan ditampilkan sebagai Memulai. Saat tugas beralih menjadi mengekspor data ke S3, progres akan ditampilkan sebagai Sedang berlangsung.

Waktu yang diperlukan untuk menyelesaikan ekspor tergantung pada data yang disimpan di basis data. Misalnya, tabel yang berisi kolom indeks atau kunci primer numerik yang terdistribusi dengan baik akan diekspor paling cepat. Tabel yang tidak berisi kolom yang sesuai untuk partisi dan tabel yang hanya berisi satu indeks pada kolom berbasis string memerlukan waktu lebih lama. Waktu ekspor yang lebih lama ini terjadi karena ekspor menggunakan proses alur tunggal yang lebih lambat.

Anda dapat mengekspor snapshot DB ke Amazon S3 menggunakan, AWS Management Console API, AWS CLI atau RDS.

Jika Anda menggunakan fungsi Lambda untuk mengekspor snapshot, tambahkan tindakan `kms:DescribeKey` ke kebijakan fungsi Lambda. Untuk informasi selengkapnya, lihat izin AWS Lambda.

Konsol

Opsi konsol Ekspor ke Amazon S3 hanya muncul untuk snapshot yang dapat diekspor ke Amazon S3. Snapshot mungkin tidak tersedia untuk diekspor karena alasan berikut:

- Mesin DB tidak didukung untuk ekspor S3.
- Versi instans DB tidak didukung untuk ekspor S3.
- Ekspor S3 tidak didukung di AWS Wilayah tempat snapshot dibuat.

Untuk mengekspor snapshot DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Dari tabnya, pilih jenis snapshot yang ingin Anda ekspor.
4. Dalam daftar snapshot, pilih snapshot yang ingin Anda ekspor.
5. Untuk Tindakan, pilih Ekspor ke Amazon S3.

   Jendela Ekspor ke Amazon S3 akan muncul.
6. Untuk Pengidentifikasi ekspor, masukkan nama untuk mengidentifikasi tugas ekspor. Nilai ini juga akan digunakan untuk nama file yang dibuat di bucket S3.
7. Pilih data yang akan diekspor:

- Pilih Semua untuk mengekspor semua data dalam snapshot.
- Pilih Sebagian untuk mengekspor bagian tertentu dari snapshot. Untuk mengidentifikasi bagian snapshot yang akan diekspor, masukkan satu atau beberapa basis data, skema, atau tabel untuk Pengidentifikasi, dipisahkan dengan spasi.

  Gunakan format berikut:

  ```
  database[.schema][.table] database2[.schema2][.table2] ... databasen[.schemaN]
  [.tablen]
  ```

  Contohnya:

  ```
  mydatabase mydatabase2.myschema1 mydatabase2.myschema2.mytable1
   mydatabase2.myschema2.mytable2
  ```

8. Untuk Bucket S3, pilih bucket yang akan dijadikan tujuan ekspor.

   Untuk menetapkan data yang diekspor ke jalur folder dalam bucket S3, masukkan jalur opsional untuk Prefiks S3.

9. Untuk Peran IAM, pilih peran yang memberi Anda akses tulis ke bucket S3 yang Anda pilih, atau buat peran baru.

   - Jika Anda membuat peran dengan mengikuti langkah-langkah di Memberikan akses ke bucket Amazon S3 menggunakan peran IAM, pilih peran tersebut.
   - Jika Anda tidak membuat peran yang memberi Anda akses tulis ke bucket S3 yang Anda pilih, pilih Buat peran baru untuk membuat peran secara otomatis. Berikutnya, masukkan nama untuk peran tersebut di Nama peran IAM.

10. Untuk AWS KMS key, masukkan ARN untuk kunci yang akan digunakan untuk mengenkripsi data yang diekspor.
11. Pilih Ekspor ke Amazon S3.

AWS CLI

Untuk mengekspor snapshot DB ke Amazon S3 menggunakan AWS CLI, gunakan perintah dengan start-export-taskopsi yang diperlukan berikut:

- `--export-task-identifier`

- `--source-arn`
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

Dalam contoh berikut, tugas ekspor snapshot diberi nama *my-snapshot-export*, yang mengekspor snapshot ke bucket S3 bernama. *my-export-bucket*

Example

Untuk Linux, macOS, atau Unix:

```
aws rds start-export-task \
    --export-task-identifier my-snapshot-export \
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name \
    --s3-bucket-name my-export-bucket \
    --iam-role-arn iam-role \
    --kms-key-id my-key
```

Untuk Windows:

```
aws rds start-export-task ^
    --export-task-identifier my-snapshot-export ^
    --source-arn arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name ^
    --s3-bucket-name my-export-bucket ^
    --iam-role-arn iam-role ^
    --kms-key-id my-key
```

Berikut adalah contoh output.

```
{
    "Status": "STARTING",
    "IamRoleArn": "iam-role",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "my-export-bucket",
    "PercentProgress": 0,
    "KmsKeyId": "my-key",
    "ExportTaskIdentifier": "my-snapshot-export",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
```

```
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:snapshot-name"
}
```

Untuk menyediakan jalur folder di bucket S3 untuk ekspor snapshot, sertakan `--s3-prefix` opsi dalam perintah. start-export-task

API RDS

Untuk mengekspor snapshot DB ke Amazon S3 menggunakan Amazon RDS API, gunakan operasi dengan parameter StartExportTaskyang diperlukan berikut:

- `ExportTaskIdentifier`
- `SourceArn`
- `S3BucketName`
- `IamRoleArn`
- `KmsKeyId`

Performa ekspor di Aurora MySQL

Snapshot klaster DB Aurora MySQL versi 2 dan versi 3 menggunakan mekanisme ekspor canggih untuk meningkatkan performa dan mengurangi waktu ekspor. Mekanisme ini mencakup pengoptimalan seperti beberapa alur ekspor dan kueri paralel Aurora MySQL untuk memanfaatkan arsitektur penyimpanan bersama Aurora. Pengoptimalan diterapkan secara adaptif, tergantung pada ukuran dan struktur set data.

Anda tidak perlu mengaktifkan kueri paralel untuk menggunakan proses ekspor yang lebih cepat, tetapi prosesnya memiliki batasan yang sama dengan kueri paralel. Selain itu, nilai data tertentu tidak didukung, seperti tanggal yang hari dalam sebulannya adalah `0` atau tahunnya adalah `0000`. Untuk informasi selengkapnya, lihat Bekerja dengan kueri paralel untuk Amazon Aurora MySQL.

Saat pengoptimalan performa diterapkan, Anda mungkin juga melihat file Parket yang berukuran jauh lebih besar (~200 GB) untuk ekspor Aurora MySQL versi 2 dan 3.

Jika proses ekspor yang lebih cepat tidak dapat digunakan, misalnya karena jenis atau nilai data yang tidak kompatibel, Aurora akan otomatis beralih ke mode ekspor alur tunggal tanpa kueri paralel. Bergantung pada proses yang digunakan dan jumlah data yang akan diekspor, performa ekspor dapat bervariasi.

Memantau ekspor snapshot

Anda dapat memantau ekspor snapshot DB menggunakan AWS Management Console, API AWS CLI, atau RDS.

Konsol

Untuk memantau ekspor snapshot DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Ekspor di Amazon S3.

   Ekspor snapshot DB ditunjukkan di kolom Jenis sumber. Status ekspor ditampilkan di kolom Status.
3. Untuk melihat informasi mendetail tentang ekspor snapshot tertentu, pilih tugas ekspornya.

AWS CLI

Untuk memantau ekspor snapshot DB menggunakan AWS CLI, gunakan perintah. describe-export-tasks

Contoh berikut menunjukkan cara menampilkan informasi saat ini tentang semua ekspor snapshot Anda.

Example

```
aws rds describe-export-tasks

{
    "ExportTasks": [
        {
            "Status": "CANCELED",
            "TaskEndTime": "2019-11-01T17:36:46.961Z",
            "S3Prefix": "something",
            "ExportTime": "2019-10-24T20:23:48.364Z",
            "S3Bucket": "examplebucket",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/
bPxRfiCYEXAMPLEKEY",
            "ExportTaskIdentifier": "anewtest",
```

```
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-25T19:10:58.885Z",
            "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:parameter-
groups-test"
        },
{
            "Status": "COMPLETE",
            "TaskEndTime": "2019-10-31T21:37:28.312Z",
            "WarningMessage": "{\"skippedTables\":[],\"skippedObjectives\":[],\"general
\":[{\"reason\":\"FAILED_TO_EXTRACT_TABLES_LIST_FOR_DATABASE\"}]}",
            "S3Prefix": "",
            "ExportTime": "2019-10-31T06:44:53.452Z",
            "S3Bucket": "examplebucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "thursday-events-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 263,
            "TaskStartTime": "2019-10-31T20:58:06.998Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-31-06-44"
        },
        {
            "Status": "FAILED",
            "TaskEndTime": "2019-10-31T02:12:36.409Z",
            "FailureCause": "The S3 bucket my-exports isn't located in the current AWS
 Region. Please, review your S3 bucket name and retry the export.",
            "S3Prefix": "",
            "ExportTime": "2019-10-30T06:45:04.526Z",
            "S3Bucket": "examplebucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/
h3yCo8nvbEXAMPLEKEY",
            "ExportTaskIdentifier": "wednesday-afternoon-test",
            "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
            "TotalExtractedDataInGB": 0,
            "TaskStartTime": "2019-10-30T22:43:40.034Z",
            "SourceArn":
 "arn:aws:rds:AWS_Region:123456789012:snapshot:rds:example-1-2019-10-30-06-45"
        }
    ]
```

```
}
```

Untuk menampilkan informasi tentang ekspor snapshot tertentu, sertakan opsi `--export-task-identifier` dengan perintah `describe-export-tasks`. Sertakan opsi `--Filters` untuk memfilter output. Untuk opsi lainnya, lihat describe-export-tasksperintah.

API RDS

Untuk menampilkan informasi tentang ekspor snapshot DB menggunakan Amazon RDS API, gunakan operasi. DescribeExportTasks

Untuk melacak penyelesaian alur kerja ekspor atau memulai alur kerja lainnya, Anda dapat berlangganan topik Amazon Simple Notification Service. Untuk informasi selengkapnya tentang Amazon SNS, lihat Bekerja dengan pemberitahuan peristiwa Amazon RDS.

Membatalkan tugas ekspor snapshot

Anda dapat membatalkan tugas ekspor snapshot DB menggunakan AWS Management Console, AWS CLI, atau RDS API.

> **Note**
>
> Membatalkan tugas ekspor snapshot tidak akan menghapus data apa pun yang telah diekspor ke Amazon S3. Untuk informasi tentang cara menghapus data menggunakan konsol, lihat Bagaimana cara menghapus objek dari bucket S3? Untuk menghapus data menggunakan CLI, gunakan perintah delete-object.

Konsol

Untuk membatalkan tugas ekspor snapshot

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Ekspor di Amazon S3.

   Ekspor snapshot DB ditunjukkan di kolom Jenis sumber. Status ekspor ditampilkan di kolom Status.
3. Pilih tugas ekspor snapshot yang ingin Anda batalkan.

4. Pilih Batalkan.
5. Pilih Batalkan tugas ekspor di halaman konfirmasi.

AWS CLI

Untuk membatalkan tugas ekspor snapshot menggunakan AWS CLI, gunakan cancel-export-taskperintah. Perintah tersebut memerlukan opsi `--export-task-identifier`.

Example

```
aws rds cancel-export-task --export-task-identifier my_export
{
    "Status": "CANCELING",
    "S3Prefix": "",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "examplebucket",
    "PercentProgress": 0,
    "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "ExportTaskIdentifier": "my_export",
    "IamRoleArn": "arn:aws:iam::123456789012:role/export-to-s3",
    "TotalExtractedDataInGB": 0,
    "TaskStartTime": "2019-11-13T19:46:00.173Z",
    "SourceArn": "arn:aws:rds:AWS_Region:123456789012:snapshot:export-example-1"
}
```

API RDS

Untuk membatalkan tugas ekspor snapshot menggunakan Amazon RDS API, gunakan CancelExportTaskoperasi dengan parameter. `ExportTaskIdentifier`

Pesan kegagalan untuk tugas ekspor Amazon S3

Tabel berikut menjelaskan pesan yang akan ditampilkan jika tugas ekspor Amazon S3 gagal.

Pesan kegagalan	Deskripsi
Terjadi kesalahan internal yang tidak diketahui.	Tugas telah gagal karena kesalahan yang tidak diketahui, pengecualian, atau kegagalan.

Pesan kegagalan	Deskripsi
Terjadi kesalahan internal yang tidak diketahui saat menulis metadata tugas ekspor ke bucket S3 [nama bucket].	Tugas telah gagal karena kesalahan yang tidak diketahui, pengecualian, atau kegagalan.
Ekspor RDS gagal menulis metadata tugas ekspor karena tidak dapat mengambil peran IAM [peran ARN].	Tugas ekspor mengambil peran IAM Anda untuk memvalidasi apakah tugas tersebut diperbolehkan menulis metadata ke bucket S3 Anda. Jika tidak dapat mengambil peran IAM Anda, berarti tugas tersebut gagal.
Ekspor RDS gagal menulis metadata tugas ekspor ke bucket S3 [nama bucket] menggunakan peran IAM [ARN peran] dengan kunci KMS [ID kunci]. Kode kesalahan: [kode kesalahan]	Satu atau beberapa izin tidak ada, sehingga tugas ekspor tidak dapat mengakses bucket S3. Pesan kegagalan ini muncul saat menerima salah satu kode kesalahan berikut: • `AWSSecurityTokenServiceException` dengan kode kesalahan `AccessDenied` • `AmazonS3Exception` dengan kode kesalahan `NoSuchBucket`, `AccessDenied`, `KMS.KMSInvalidStateException`, `403 Forbidden`, atau `KMS.DisabledException` Kode kesalahan ini menunjukkan bahwa pengaturan salah dikonfigurasi untuk peran IAM, bucket S3, atau kunci KMS.
Peran IAM [ARN peran] tidak diizinkan untuk memanggil [tindakan S3] pada bucket S3 [nama bucket]. Tinjau izin Anda dan coba lagi ekspor.	Kebijakan IAM salah dikonfigurasi. Izin untuk tindakan S3 tertentu pada bucket S3 tidak ada, sehingga menyebabkan tugas ekspor gagal.
Pemeriksaan kunci KMS gagal. Periksa kredensial pada kunci KMS Anda, lalu coba lagi.	Pemeriksaan kredensial kunci KMS gagal.

Pesan kegagalan	Deskripsi
Pemeriksaan kredensial S3 gagal. Periksa izin pada bucket S3 dan kebijakan IAM Anda.	Pemeriksaan kredensial S3 gagal.
Bucket S3 [nama bucket] tidak valid. Bucket tersebut tidak berada di Wilayah AWS saat ini atau tidak ada. Tinjau nama bucket S3 Anda, lalu coba lagi ekspor.	Bucket S3 tidak valid.
Bucket S3 [nama bucket] tidak berada di Wilayah AWS saat ini. Tinjau nama bucket S3 Anda, lalu coba lagi ekspor.	Ember S3 salah Wilayah AWS.

Memecahkan masalah kesalahan izin PostgreSQL

Saat mengekspor basis data PostgreSQL ke Amazon S3, Anda mungkin melihat kesalahan `PERMISSIONS_DO_NOT_EXIST` yang menyatakan bahwa tabel tertentu dilewati. Kesalahan ini biasanya terjadi saat pengguna super, yang Anda tetapkan saat membuat instans DB, tidak memiliki izin untuk mengakses tabel tersebut.

Untuk memperbaiki kesalahan ini, jalankan perintah berikut:

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO superuser_name
```

Untuk informasi selengkapnya tentang hak pengguna super, lihat Hak akses akun pengguna master.

Konvensi penamaan file

Data yang diekspor untuk tabel tertentu disimpan dalam format *base_prefix*/*files*, dengan prefiks dasar sebagai berikut:

```
export_identifier/database_name/schema_name.table_name/
```

Contohnya:

```
export-1234567890123-459/rdststdb/rdststdb.DataInsert_7ADB5D19965123A2/
```

Ada dua konvensi cara penamaan file.

- Konvensi saat ini:

  ```
  batch_index/part-partition_index-random_uuid.format-based_extension
  ```

  Indeks batch adalah nomor urut yang mewakili batch data yang dibaca dari tabel. Jika kami tidak dapat mempartisi tabel Anda menjadi bagian-bagian kecil untuk diekspor secara paralel, akan ada beberapa indeks batch. Hal yang sama akan terjadi jika tabel Anda dipartisi menjadi beberapa tabel. Beberapa indeks batch akan tersedia, dengan satu untuk setiap partisi tabel dari tabel utama Anda.

  Jika kami dapat mempartisi tabel Anda menjadi bagian-bagian kecil yang akan dibaca secara paralel, hanya akan ada folder 1 indeks batch.

  Di dalam folder indeks batch, akan ada satu atau beberapa file Parquet yang berisi data tabel Anda. Prefiks file Parket adalah `part-`*`partition_index`*. Jika tabel Anda dipartisi, akan ada beberapa file yang diawali dengan indeks partisi `00000`.

  Mungkin ada kesenjangan dalam urutan indeks partisi. Hal ini terjadi karena setiap partisi diperoleh dari kueri dengan rentang di tabel Anda. Jika tidak ada data dalam rentang partisi tersebut, maka nomor urut itu akan dilewati.

  Misalnya, anggap kolom `id` adalah kunci primer tabel, dan nilai minimum dan maksimumnya adalah `100` dan `1000`. Saat kami mencoba mengekspor tabel ini dengan sembilan partisi, kami membacanya dengan kueri paralel seperti berikut:

  ```
  SELECT * FROM table WHERE id <= 100 AND id < 200
  SELECT * FROM table WHERE id <= 200 AND id < 300
  ```

  Partisi ini akan menghasilkan sembilan file, dari `part-00000-`*`random_uuid`*`.gz.parquet` hingga `part-00008-`*`random_uuid`*`.gz.parquet`. Namun, jika tidak ada baris dengan ID antara `200` dan `350`, maka salah satu partisi yang telah selesai akan kosong, dan tidak ada file yang dibuat untuk partisi itu. Dalam contoh sebelumnya, `part-00001-`*`random_uuid`*`.gz.parquet` tidak dibuat.

- Konvensi yang lebih lama:

```
part-partition_index-random_uuid.format-based_extension
```

Konvensi ini sama seperti konvensi saat ini, tetapi tanpa prefiks *batch_index*, contohnya:

```
part-00000-c5a881bb-58ff-4ee6-1111-b41ecff340a3-c000.gz.parquet
part-00001-d7a881cc-88cc-5ab7-2222-c41ecab340a4-c000.gz.parquet
part-00002-f5a991ab-59aa-7fa6-3333-d41eccd340a7-c000.gz.parquet
```

Konvensi penamaan file dapat berubah sewaktu-waktu. Oleh karena itu, saat membaca tabel target, sebaiknya baca segala sesuatu di dalam prefiks dasar untuk tabel tersebut.

Konversi data saat mengekspor ke bucket Amazon S3

Saat Anda mengekspor snapshot DB ke bucket Amazon S3 Amazon Aurora akan mengonversi data ke, mengekspor data dalam, dan menyimpan data dalam format Parquet. Untuk informasi selengkapnya tentang Parquet, lihat situs web Apache Parquet.

Parquet menyimpan semua data sebagai salah satu jenis primitif berikut:

- BOOLEAN
- INT32
- INT64
- INT96
- FLOAT
- DOUBLE
- BYTE_ARRAY – Array byte dengan panjang variabel, juga dikenal sebagai biner
- FIXED_LEN_BYTE_ARRAY – Array byte dengan panjang tetap yang digunakan saat nilai memiliki ukuran konstan

Jenis data Parquet berjumlah sedikit untuk mengurangi kerumitan membaca dan menulis format. Parquet menyediakan jenis logis untuk memperluas jenis primitif. Jenis logis diimplementasikan sebagai anotasi dengan data di kolom metadata `LogicalType`. Anotasi jenis logis menjelaskan cara menginterpretasikan jenis primitif.

Saat jenis logis `STRING` menganotasi jenis `BYTE_ARRAY`, hal ini menunjukkan bahwa array byte harus diinterpretasikan sebagai string karakter yang dienkode UTF-8. Setelah tugas ekspor selesai, Amazon Aurora akan memberi tahu Anda jika terjadi konversi string. Data dasar yang diekspor selalu sama seperti data dari sumbernya. Namun, karena perbedaan encoding dalam UTF-8, beberapa karakter mungkin terlihat berbeda dari sumber saat dibaca di alat seperti Athena.

Untuk informasi selengkapnya, lihat Parquet logical type definitions dalam dokumentasi Parquet.

Topik

- Pemetaan jenis data MySQL ke Parquet
- Pemetaan jenis data PostgreSQL ke Parquet

Pemetaan jenis data MySQL ke Parquet

Tabel berikut menunjukkan pemetaan dari jenis data MySQL ke jenis data Parquet saat data dikonversi dan diekspor ke Amazon S3.

Jenis data sumber	Jenis primitif Parquet	Anotasi jenis logis	Catatan konversi
Jenis data numerik			
BIGINT	INT64		
BIGINT UNSIGNED	FIXED_LEN _BYTE_ARRAY(9)	DECIMAL(20,0)	Parquet hanya mendukung jenis yang ditandatangani, sehingga pemetaann ya memerlukan tambahan byte (8 plus 1) untuk menyimpan jenis BIGINT_UN SIGNED.
BIT	BYTE_ARRAY		
DECIMAL	INT32	DECIMAL(p,s)	Jika nilai sumber kurang dari 2^{31}, maka nilai tersebut akan

Jenis data sumber	Jenis primitif Parquet	Anotasi jenis logis	Catatan konversi
			disimpan sebagai INT32.
	INT64	DECIMAL(p,s)	Jika nilai sumber adalah 2^{31} atau lebih besar, tetapi kurang dari 2^{63}, maka nilai tersebut akan disimpan sebagai INT64.
	FIXED_LEN _BYTE_ARRAY(N)	DECIMAL(p,s)	Jika nilai sumber adalah 2^{63} atau lebih besar, maka nilai tersebut akan disimpan sebagai FIXED_LEN _BYTE_ARRAY(N).
	BYTE_ARRAY	STRING	Parquet tidak mendukung presisi Desimal yang lebih besar dari 38. Nilai Desimal akan dikonversi menjadi string dalam jenis BYTE_ARRAY dan dienkode sebagai UTF8.
DOUBLE	DOUBLE		
FLOAT	DOUBLE		
INT	INT32		
INT UNSIGNED	INT64		

Jenis data sumber	Jenis primitif Parquet	Anotasi jenis logis	Catatan konversi
MEDIUMINT	INT32		
MEDIUMINT UNSIGNED	INT64		
NUMERIC	INT32	DECIMAL(p,s)	Jika nilai sumber kurang dari 2^{31}, maka nilai tersebut akan disimpan sebagai INT32.
	INT64	DECIMAL(p,s)	Jika nilai sumber adalah 2^{31} atau lebih besar, tetapi kurang dari 2^{63}, maka nilai tersebut akan disimpan sebagai INT64.
	FIXED_LEN _ARRAY(N)	DECIMAL(p,s)	Jika nilai sumber adalah 2^{63} atau lebih besar, maka nilai tersebut akan disimpan sebagai FIXED_LEN _BYTE_ARRAY(N).

Jenis data sumber	Jenis primitif Parquet	Anotasi jenis logis	Catatan konversi
	BYTE_ARRAY	STRING	Parquet tidak mendukung presisi Numerik yang lebih besar dari 38. Nilai Numerik ini akan dikonversi menjadi string dalam jenis BYTE_ARRAY dan dienkode sebagai UTF8.
SMALLINT	INT32		
SMALLINT UNSIGNED	INT32		
TINYINT	INT32		
TINYINT UNSIGNED	INT32		
Jenis data string			
BINARY	BYTE_ARRAY		
BLOB	BYTE_ARRAY		
CHAR	BYTE_ARRAY		
ENUM	BYTE_ARRAY	STRING	
LINESTRING	BYTE_ARRAY		
LONGBLOB	BYTE_ARRAY		
LONGTEXT	BYTE_ARRAY	STRING	
MEDIUMBLOB	BYTE_ARRAY		

Jenis data sumber	Jenis primitif Parquet	Anotasi jenis logis	Catatan konversi
MEDIUMTEXT	BYTE_ARRAY	STRING	
MULTILINESTRING	BYTE_ARRAY		
SET	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TINYBLOB	BYTE_ARRAY		
TINYTEXT	BYTE_ARRAY	STRING	
VARBINARY	BYTE_ARRAY		
VARCHAR	BYTE_ARRAY	STRING	
Jenis data tanggal dan waktu			
DATE	BYTE_ARRAY	STRING	Tanggal akan dikonversi menjadi string dalam jenis BYTE_ARRAY dan dienkode sebagai UTF8.
DATETIME	INT64	TIMESTAMP_MICROS	
TIME	BYTE_ARRAY	STRING	Jenis TIME akan dikonversi menjadi string dalam jenis BYTE_ARRAY dan dienkode sebagai UTF8.
TIMESTAMP	INT64	TIMESTAMP_MICROS	

Jenis data sumber	Jenis primitif Parquet	Anotasi jenis logis	Catatan konversi
YEAR	INT32		
Jenis data geometris			
GEOMETRY	BYTE_ARRAY		
GEOMETRYCOLLECTION	BYTE_ARRAY		
MULTIPOINT	BYTE_ARRAY		
MULTIPOLYGON	BYTE_ARRAY		
POINT	BYTE_ARRAY		
POLYGON	BYTE_ARRAY		
Jenis data JSON			
JSON	BYTE_ARRAY	STRING	

Pemetaan jenis data PostgreSQL ke Parquet

Tabel berikut menunjukkan pemetaan dari dan jenis data PostgreSQL ke jenis data Parquet saat data dikonversi dan diekspor ke Amazon S3.

Jenis data PostgreSQL	Jenis primitif Parquet	Anotasi jenis logis	Catatan pemetaan
Jenis data numerik			
BIGINT	INT64		
BIGSERIAL	INT64		
DECIMAL	BYTE_ARRAY	STRING	Jenis DECIMAL akan dikonversi ke string dalam jenis

Jenis data PostgreSQL	Jenis primitif Parquet	Anotasi jenis logis	Catatan pemetaan
			BYTE_ARRAY dan dienkode sebagai UTF8. Konversi ini dimaksudkan untuk menghindari kerumitan akibat presisi data dan nilai data yang bukan berupa angka (NaN).
DOUBLE PRECISION	DOUBLE		
INTEGER	INT32		
MONEY	BYTE_ARRAY	STRING	
REAL	FLOAT		
SERIAL	INT32		
SMALLINT	INT32	INT_16	
SMALLSERIAL	INT32	INT_16	
Jenis data string dan terkait			

Jenis data PostgreSQL	Jenis primitif Parquet	Anotasi jenis logis	Catatan pemetaan
ARRAY	BYTE_ARRAY	STRING	Array akan dikonversi menjadi string dan dienkode sebagai BINARY (UTF8). Konversi ini dimaksudkan untuk menghindari kerumitan akibat presisi data, nilai data yang bukan berupa angka (NaN), dan nilai data waktu.
BIT	BYTE_ARRAY	STRING	
BIT VARYING	BYTE_ARRAY	STRING	
BYTEA	BINARY		
CHAR	BYTE_ARRAY	STRING	
CHAR(N)	BYTE_ARRAY	STRING	
ENUM	BYTE_ARRAY	STRING	
NAME	BYTE_ARRAY	STRING	
TEXT	BYTE_ARRAY	STRING	
TEXT SEARCH	BYTE_ARRAY	STRING	
VARCHAR(N)	BYTE_ARRAY	STRING	
XML	BYTE_ARRAY	STRING	

Jenis data PostgreSQL	Jenis primitif Parquet	Anotasi jenis logis	Catatan pemetaan
Jenis data tanggal dan waktu			
DATE	BYTE_ARRAY	STRING	
INTERVAL	BYTE_ARRAY	STRING	
TIME	BYTE_ARRAY	STRING	
TIME WITH TIME ZONE	BYTE_ARRAY	STRING	
TIMESTAMP	BYTE_ARRAY	STRING	
TIMESTAMP WITH TIME ZONE	BYTE_ARRAY	STRING	
Jenis data geometris			
BOX	BYTE_ARRAY	STRING	
CIRCLE	BYTE_ARRAY	STRING	
LINE	BYTE_ARRAY	STRING	
LINESEGMENT	BYTE_ARRAY	STRING	
PATH	BYTE_ARRAY	STRING	
POINT	BYTE_ARRAY	STRING	
POLYGON	BYTE_ARRAY	STRING	
Jenis data JSON			
JSON	BYTE_ARRAY	STRING	
JSONB	BYTE_ARRAY	STRING	
Jenis data lainnya			

Jenis data PostgreSQL	Jenis primitif Parquet	Anotasi jenis logis	Catatan pemetaan
BOOLEAN	BOOLEAN		
CIDR	BYTE_ARRAY	STRING	Jenis data jaringan
COMPOSITE	BYTE_ARRAY	STRING	
DOMAIN	BYTE_ARRAY	STRING	
INET	BYTE_ARRAY	STRING	Jenis data jaringan
MACADDR	BYTE_ARRAY	STRING	
OBJECT IDENTIFIER	N/A		
PG_LSN	BYTE_ARRAY	STRING	
RANGE	BYTE_ARRAY	STRING	
UUID	BYTE_ARRAY	STRING	

Memulihkan klaster DB ke waktu tertentu

Anda dapat memulihkan klaster DB ke titik waktu tertentu, sehingga membuat klaster DB baru.

Saat Anda memulihkan klaster DB ke titik waktu tertentu, Anda dapat memilih grup keamanan cloud privat virtual (VPC) default. Atau Anda dapat menerapkan grup keamanan VPC kustom ke klaster DB Anda.

Klaster DB yang dipulihkan secara otomatis dikaitkan dengan klaster DB dan grup parameter DB default. Namun, Anda dapat menerapkan grup parameter kustom dengan menentukannya selama pemulihan.

Amazon Aurora mengunggah catatan log untuk klaster DB ke Amazon S3 terus menerus. Untuk melihat waktu restorable terbaru untuk cluster DB, gunakan AWS CLI describe-db-clustersperintah dan lihat nilai yang dikembalikan di `LatestRestorableTime` bidang untuk cluster DB.

Anda dapat memulihkan ke titik waktu mana pun dalam periode retensi cadangan Anda.
Untuk melihat waktu restorable paling awal untuk cluster DB, gunakan AWS CLI describe-db-clustersperintah dan lihat nilai yang dikembalikan di `EarliestRestorableTime` lapangan untuk cluster DB.

Periode retensi cadangan dari klaster DB yang dipulihkan sama dengan klaster DB sumber.

> **Note**
>
> Informasi dalam topik ini berlaku untuk Amazon Aurora. Untuk informasi tentang cara memulihkan instans DB Amazon RDS, lihat Memulihkan instans DB ke waktu yang ditentukan.
> Untuk informasi tentang pencadangan dan pemulihan klaster DB Aurora, lihat Gambaran umum pencadangan dan pemulihan klaster DB Aurora.
> Untuk Aurora MySQL, Anda dapat memulihkan klaster DB yang disediakan ke klaster DB Aurora Serverless. Untuk informasi selengkapnya, lihat Memulihkan klaster DB Aurora Serverless v1.
> Anda juga dapat menggunakan AWS Backup untuk mengelola cadangan cluster Amazon Aurora DB. Jika cluster DB Anda dikaitkan dengan rencana cadangan di AWS Backup, paket cadangan itu digunakan untuk point-in-time pemulihan. Untuk informasi, lihat Memulihkan cluster DB ke waktu tertentu menggunakan AWS Backup.

Untuk informasi tentang memulihkan klaster Aurora DB atau klaster global dengan versi RDS Extended Support, lihat. Aurora DB atau cluster global dengan Amazon RDS Extended Support

Anda dapat memulihkan cluster DB ke titik waktu menggunakan AWS Management Console, AWS CLI, atau RDS API.

Konsol

Memulihkan klaster DB ke waktu tertentu

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Pencadangan otomatis.

   Pencadangan otomatis ditampilkan di tab Wilayah Saat Ini.

3. Pilih klaster DB yang ingin Anda pulihkan.
4. Untuk Tindakan, pilih Pulihkan ke titik waktu.

   Jendela Pulihkan ke titik waktu akan muncul.
5. Pilih Waktu pemulihan terbaru untuk memulihkan ke waktu terbaru yang dimungkinkan atau pilih Kustom untuk memilih waktu.

   Jika Anda memilih Kustom, masukkan tanggal dan waktu untuk mengembalikan klaster.

   > **Note**
   >
   > Waktu ditampilkan dalam zona waktu lokal Anda, yang ditunjukkan dengan offset dari Coordinated Universal Time (UTC). Misalnya, UTC-5 adalah Waktu Standar Timur/Waktu Musim Panas Tengah.

6. Untuk Pengidentifikasi klaster DB, masukkan nama klaster DB target yang dipulihkan. Nama harus unik.

7. Pilih opsi lain sesuai kebutuhan, seperti kelas instans DB dan konfigurasi penyimpanan klaster DB.

   Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

8. Pilih Pulihkan ke titik waktu.

AWS CLI

Untuk mengembalikan cluster DB ke waktu tertentu, gunakan AWS CLI perintah restore-db-cluster-to-point-in-time untuk membuat cluster DB baru.

Anda dapat menentukan pengaturan lain. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

Pemberian tag sumber daya didukung untuk operasi ini. Saat Anda menggunakan opsi `--tags`, tag klaster DB sumber diabaikan dan tag yang disediakan digunakan. Jika tidak, tag terbaru dari klaster sumber digunakan.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-identifier mysourcedbcluster \
    --db-cluster-identifier mytargetdbcluster \
    --restore-to-time 2017-10-14T23:45:00.000Z
```

Untuk Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-identifier mysourcedbcluster ^
    --db-cluster-identifier mytargetdbcluster ^
    --restore-to-time 2017-10-14T23:45:00.000Z
```

> **Important**
>
> Jika Anda menggunakan konsol untuk memulihkan klaster DB ke waktu tertentu, maka Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan AWS CLI untuk mengembalikan cluster DB ke waktu tertentu, Anda

harus secara eksplisit membuat instance utama untuk cluster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB.
Untuk membuat instance utama untuk cluster DB Anda, panggil create-db-instance AWS CLI perintah. Sertakan nama klaster DB sebagai `--db-cluster-identifier` nilai pilihan.

API RDS

Untuk memulihkan klaster DB ke waktu yang ditentukan, panggil operasi API Amazon RDS `RestoreDBClusterToPointInTime` dengan parameter berikut ini:

- `SourceDBClusterIdentifier`
- `DBClusterIdentifier`
- `RestoreToTime`

Important

Jika Anda menggunakan konsol untuk memulihkan klaster DB ke waktu tertentu, maka Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan API RDS untuk memulihkan klaster DB ke waktu tertentu, pastikan Anda secara eksplisit membuat instans primer untuk klaster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB.
Untuk membuat instans primer untuk klaster DB Anda, panggil operasi API RDS CreateDBInstance. Sertakan nama klaster DB sebagai nilai parameter `DBClusterIdentifier`.

Memulihkan klaster DB ke waktu tertentu dari cadangan otomatis yang dipertahankan

Anda dapat memulihkan klaster DB dari cadangan otomatis yang dipertahankan setelah Anda menghapus klaster DB sumber, jika cadangan tersebut berada dalam periode retensi klaster sumber. Prosesnya mirip dengan pemulihan klaster DB dari cadangan otomatis.

> **Note**
>
> Anda tidak dapat memulihkan klaster DB Aurora Serverless v1 menggunakan prosedur ini, karena pencadangan otomatis untuk klaster Aurora Serverless v1 tidak dipertahankan.

Konsol

Memulihkan klaster DB ke waktu tertentu

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Pencadangan otomatis.
3. Pilih tab Dipertahankan.

4. Pilih klaster DB yang ingin Anda pulihkan.
5. Untuk Tindakan, pilih Pulihkan ke titik waktu.

   Jendela Pulihkan ke titik waktu akan muncul.
6. Pilih Waktu pemulihan terbaru untuk memulihkan ke waktu terbaru yang dimungkinkan atau pilih Kustom untuk memilih waktu.

   Jika Anda memilih Kustom, masukkan tanggal dan waktu untuk mengembalikan klaster.

   > **Note**
   >
   > Waktu ditampilkan dalam zona waktu lokal Anda, yang ditunjukkan dengan offset dari Coordinated Universal Time (UTC). Misalnya, UTC-5 adalah Waktu Standar Timur/Waktu Musim Panas Tengah.
7. Untuk Pengidentifikasi klaster DB, masukkan nama klaster DB target yang dipulihkan. Nama harus unik.

8. Pilih opsi lain sesuai kebutuhan, seperti kelas instans DB.

   Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

9. Pilih Pulihkan ke titik waktu.

AWS CLI

Untuk mengembalikan cluster DB ke waktu tertentu, gunakan AWS CLI perintah restore-db-cluster-to-point-in-time untuk membuat cluster DB baru.

Anda dapat menentukan pengaturan lain. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

Pemberian tag sumber daya didukung untuk operasi ini. Saat Anda menggunakan opsi `--tags`, tag klaster DB sumber diabaikan dan tag yang disediakan digunakan. Jika tidak, tag terbaru dari klaster sumber digunakan.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-to-point-in-time \
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE \
    --db-cluster-identifier mytargetdbcluster \
    --restore-to-time 2017-10-14T23:45:00.000Z
```

Untuk Windows:

```
aws rds restore-db-cluster-to-point-in-time ^
    --source-db-cluster-resource-id cluster-123ABCEXAMPLE ^
    --db-cluster-identifier mytargetdbcluster ^
    --restore-to-time 2017-10-14T23:45:00.000Z
```

Important

Jika Anda menggunakan konsol untuk memulihkan klaster DB ke waktu tertentu, maka Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan AWS CLI untuk mengembalikan cluster DB ke waktu tertentu, Anda harus secara eksplisit membuat instance utama untuk cluster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB.

Untuk membuat instance utama untuk cluster DB Anda, panggil create-db-instance AWS CLI perintah. Sertakan nama klaster DB sebagai `--db-cluster-identifier` nilai pilihan.

API RDS

Untuk memulihkan klaster DB ke waktu yang ditentukan, panggil operasi API Amazon RDS `RestoreDBClusterToPointInTime` dengan parameter berikut ini:

- `SourceDbClusterResourceId`
- `DBClusterIdentifier`
- `RestoreToTime`

Important

Jika Anda menggunakan konsol untuk memulihkan klaster DB ke waktu tertentu, maka Amazon RDS akan secara otomatis membuat instans primer (penulis) untuk klaster DB Anda. Jika Anda menggunakan API RDS untuk memulihkan klaster DB ke waktu tertentu, pastikan Anda secara eksplisit membuat instans primer untuk klaster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB.
Untuk membuat instans primer untuk klaster DB Anda, panggil operasi API RDS CreateDBInstance. Sertakan nama klaster DB sebagai nilai parameter `DBClusterIdentifier`.

Memulihkan cluster DB ke waktu tertentu menggunakan AWS Backup

Anda dapat menggunakan AWS Backup untuk mengelola cadangan otomatis Anda, dan kemudian mengembalikannya ke waktu yang ditentukan. Untuk melakukan ini, Anda membuat rencana cadangan AWS Backup dan menetapkan cluster DB Anda sebagai sumber daya. Kemudian Anda mengaktifkan pencadangan berkelanjutan untuk PITR dalam aturan pencadangan. Untuk informasi selengkapnya tentang rencana pencadangan dan aturan pencadangan, lihat Panduan Developer tentang Pencadangan AWS .

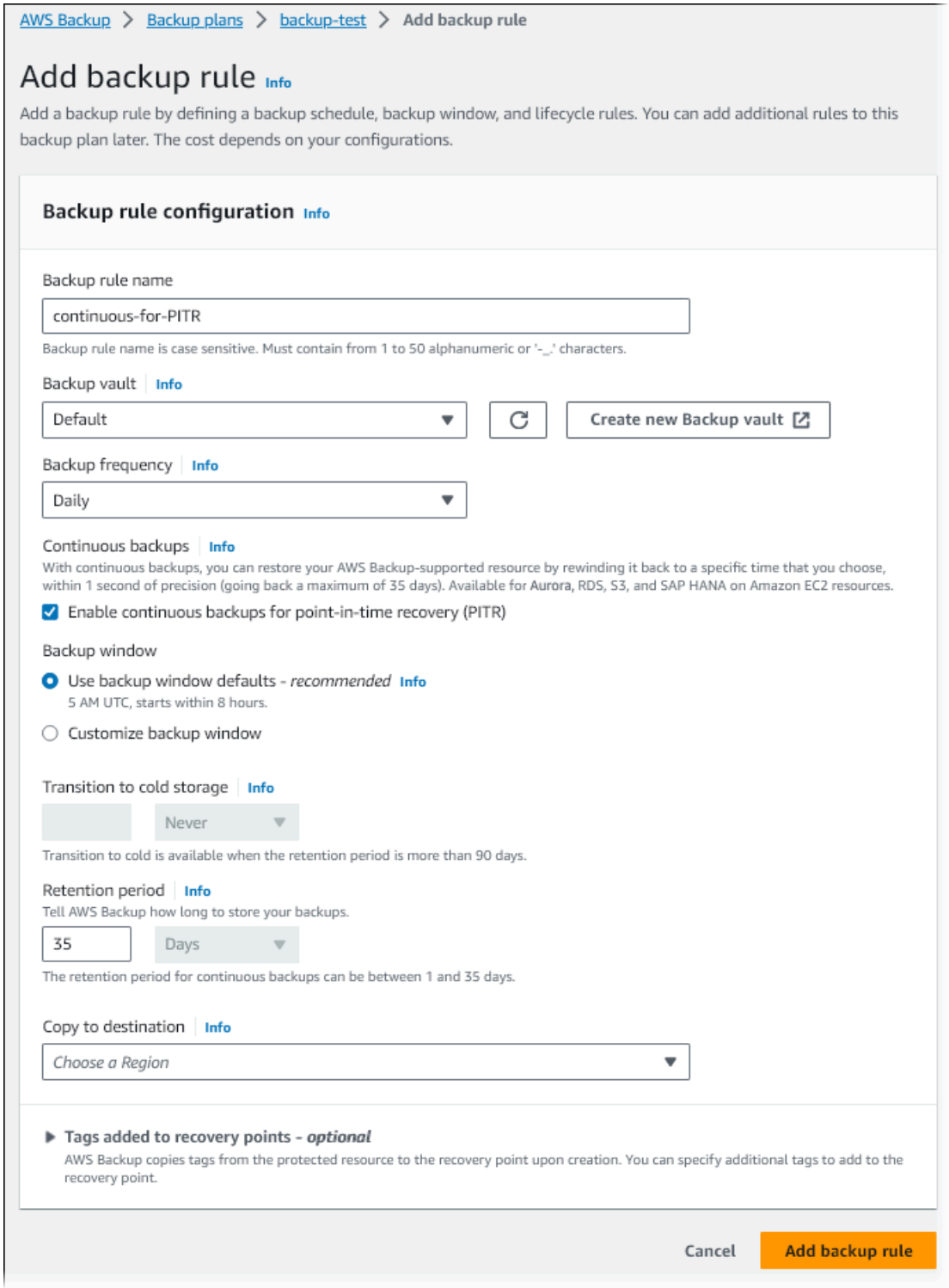
Mengaktifkan pencadangan berkelanjutan di AWS Backup

Anda mengaktifkan pencadangan berkelanjutan dalam aturan pencadangan.

Mengaktifkan pencadangan berkelanjutan untuk PITR

1. Masuk ke AWS Management Console, dan buka AWS Backup konsol di https://console.aws.amazon.com/backup.
2. Di panel navigasi, pilih Rencana cadangan.
3. Di bawah Nama rencana pencadangan, pilih rencana pencadangan yang Anda gunakan untuk membuat cadangan klaster DB Anda.
4. Di bawah bagian Aturan pencadangan, pilih Tambahkan aturan pencadangan.

   Halaman Tambahkan aturan pencadangan ditampilkan.
5. Pilih kotak centang Aktifkan cadangan berkelanjutan untuk point-in-time pemulihan (PITR).



6. Pilih pengaturan lain sesuai kebutuhan, lalu pilih Tambahkan aturan pencadangan.

Memulihkan dari cadangan berkelanjutan di AWS Backup

Anda memulihkan ke waktu tertentu dari brankas cadangan.

Konsol

Anda dapat menggunakan AWS Management Console untuk mengembalikan cluster DB ke waktu yang ditentukan.

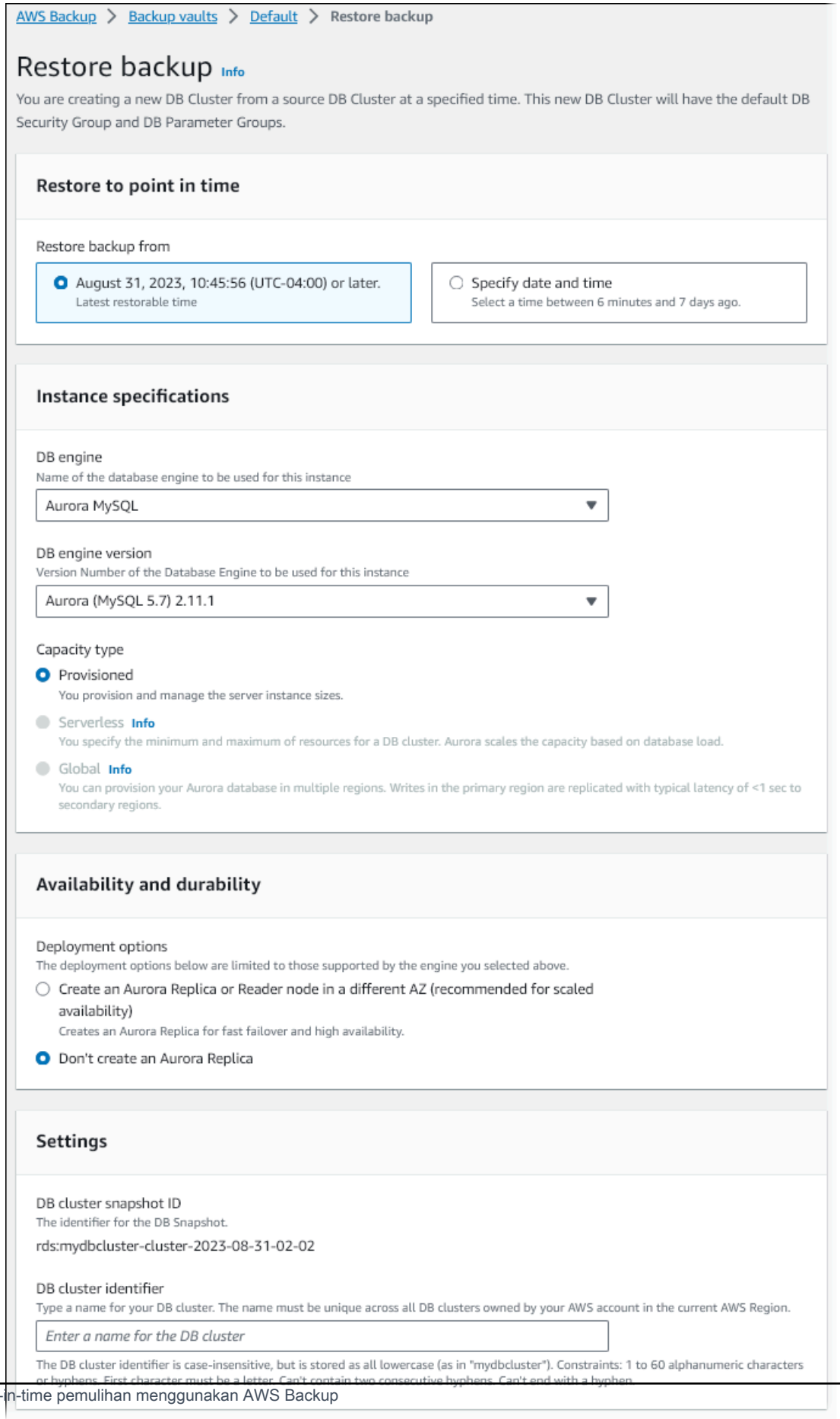
Untuk memulihkan dari cadangan berkelanjutan di AWS Backup

1. Masuk ke AWS Management Console, dan buka AWS Backup konsol di https://console.aws.amazon.com/backup.
2. Di panel navigasi, pilih Brankas cadangan.
3. Pilih brankas cadangan yang berisi cadangan berkelanjutan Anda, misalnya Default.

   Halaman detail brankas cadangan ditampilkan.
4. Di bagian Titik pemulihan, pilih titik pemulihan untuk cadangan otomatis.

   Terdapat jenis cadangan Berkelanjutan dan sebuah nama dengan `continuous:cluster-`*`AWS-Backup-job-number`*.
5. Untuk Tindakan, pilih Pulihkan.

   Halaman Pulihkan cadangan ditampilkan.


AWS Backup > Backup vaults > Default > Restore backup
Restore backup Info
You are creating a new DB Cluster from a source DB Cluster at a specified time. This new DB Cluster will have the default DB Security Group and DB Parameter Groups.
Restore to point in time
Restore backup from
August 31, 2023, 10:45:56 (UTC-04:00) or later.
Latest restorable time
Specify date and time
Select a time between 6 minutes and 7 days ago.
Instance specifications
DB engine
Name of the database engine to be used for this instance
Aurora MySQL
DB engine version
Version Number of the Database Engine to be used for this instance
Aurora (MySQL 5.7) 2.11.1
Capacity type
Provisioned
You provision and manage the server instance sizes.
Serverless Info
You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load.
Global Info
You can provision your Aurora database in multiple regions. Writes in the primary region are replicated with typical latency of <1 sec to secondary regions.
Availability and durability
Deployment options
The deployment options below are limited to those supported by the engine you selected above.
Create an Aurora Replica or Reader node in a different AZ (recommended for scaled availability)
Creates an Aurora Replica for fast failover and high availability.
Don't create an Aurora Replica
Settings
DB cluster snapshot ID
The identifier for the DB Snapshot.
rds:mydbcluster-cluster-2023-08-31-02-02
DB cluster identifier
Type a name for your DB cluster. The name must be unique across all DB clusters owned by your AWS account in the current AWS Region.
Enter a name for the DB cluster
The DB cluster identifier is case-insensitive, but is stored as all lowercase (as in "mydbcluster"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

6. Untuk Pulihkan ke titik waktu, pilih Tentukan tanggal dan waktu untuk memulihkan ke titik waktu tertentu.
7. Pilih pengaturan lain sesuai kebutuhan untuk memulihkan klaster DB, lalu pilih Pulihkan cadangan.

   Halaman Pekerjaan ditampilkan, menampilkan panel Pekerjaan pemulihan. Pesan di bagian atas halaman memberikan informasi tentang pekerjaan pemulihan.

Setelah klaster DB dipulihkan, Anda harus menambahkan instans DB primer (penulis) ke dalamnya. Untuk membuat instance utama untuk cluster DB Anda, panggil create-db-instance AWS CLI perintah. Sertakan nama klaster DB sebagai nilai parameter `--db-cluster-identifier`.

CLI

Anda menggunakan start-restore-job AWS CLI perintah untuk mengembalikan cluster DB ke waktu yang ditentukan. Parameter-parameter berikut diperlukan:

- `--recovery-point-arn` – Amazon Resource Name (ARN) untuk titik pemulihan yang menjadi asal pemulihan.
- `--resource-type` – Gunakan `Aurora`.
- `--iam-role-arn`— ARN untuk peran IAM yang Anda gunakan untuk operasi. AWS Backup
- `--metadata` – Metadata yang Anda gunakan untuk memulihkan klaster DB. Parameter berikut diperlukan:
  - `DBClusterIdentifier`
  - `Engine`
  - `RestoreToTime` atau `UseLatestRestorableTime`

Contoh berikut ini menunjukkan cara memulihkan klaster DB ke waktu tertentu.

```
aws backup start-restore-job \
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
--metadata '{"DBClusterIdentifier":"backup-pitr-test","Engine":"aurora-
mysql","RestoreToTime":"2023-09-01T17:00:00.000Z"}'
```

Contoh berikut ini menunjukkan cara memulihkan klaster DB ke waktu pemulihan terbaru.

```
aws backup start-restore-job \
--recovery-point-arn arn:aws:backup:eu-central-1:123456789012:recovery-
point:continuous:cluster-itsreallyjustanexample1234567890-487278c2 \
--resource-type Aurora \
--iam-role-arn arn:aws:iam::123456789012:role/service-role/AWSBackupDefaultServiceRole
 \
--metadata '{"DBClusterIdentifier":"backup-pitr-latest","Engine":"aurora-
mysql","UseLatestRestorableTime":"true"}'
```

Setelah klaster DB dipulihkan, Anda harus menambahkan instans DB primer (penulis) ke dalamnya. Untuk membuat instance utama untuk cluster DB Anda, panggil create-db-instance AWS CLI perintah. Sertakan nama klaster DB sebagai nilai parameter `--db-cluster-identifier`.

Menghapus snapshot klaster DB

Anda dapat menghapus snapshot klaster DB yang dikelola oleh Amazon RDS jika Anda tidak lagi membutuhkannya.

> **Note**
>
> Untuk menghapus cadangan yang dikelola oleh AWS Backup, gunakan konsol AWS Backup. Untuk informasi selengkapnya tentang AWS Backup, lihat Panduan Developer AWS Backup.

Menghapus snapshot klaster DB

Anda dapat menghapus snapshot klaster DB menggunakan konsol, AWS CLI, atau API RDS.

Untuk menghapus snapshot bersama atau publik, Anda harus masuk ke akun AWS yang memiliki snapshot tersebut.

Konsol

Untuk menghapus snapshot klaster DB

1. Masuk ke AWS Management Console, lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih snapshot klaster DB yang ingin Anda hapus.
4. Untuk Tindakan, pilih Hapus snapshot.
5. Pilih Hapus di halaman konfirmasi.

AWS CLI

Anda dapat menghapus snapshot cluster DB dengan menggunakan AWS CLI perintah delete-db-cluster-snapshot.

Opsi berikut digunakan untuk menghapus snapshot klaster DB.

- `--db-cluster-snapshot-identifier` – Pengidentifikasi untuk snapshot klaster DB.

Example

Kode berikut menghapus snapshot klaster DB `mydbclustersnapshot`.

Untuk Linux, macOS, atau Unix:

```
aws rds delete-db-cluster-snapshot \
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

Untuk Windows:

```
aws rds delete-db-cluster-snapshot ^
    --db-cluster-snapshot-identifier mydbclustersnapshot
```

API RDS

Anda dapat menghapus snapshot cluster DB dengan menggunakan operasi Amazon RDS API DeleteDB. ClusterSnapshot

Parameter berikut digunakan untuk menghapus snapshot klaster DB.

- `DBClusterSnapshotIdentifier` – Pengidentifikasi untuk snapshot klaster DB.

Tutorial: Memulihkan klaster DB Amazon Aurora dari snapshot klaster DB

Skenario umum saat bekerja dengan Amazon Aurora adalah memiliki instans DB yang jarang Anda gunakan, tetapi tidak selalu Anda perlukan. Misalnya, Anda mungkin menggunakan klaster DB untuk menyimpan data untuk laporan yang hanya dijalankan setiap tiga bulan. Salah satu cara untuk menghemat biaya pada skenario tersebut adalah dengan mengambil snapshot klaster DB setelah laporan selesai. Kemudian, Anda menghapus klaster DB, dan memulihkannya saat Anda perlu mengunggah data baru dan menjalankan laporan selama kuartal berikutnya.

Saat memulihkan klaster DB, masukkan nama snapshot klaster DB yang akan digunakan untuk memulihkan. Kemudian, masukkan nama untuk klaster DB baru yang dibuat dari operasi pemulihan. Untuk informasi lebih mendetail tentang cara memulihkan klaster DB dari snapshot, lihat Memulihkan dari snapshot klaster DB.

Dalam tutorial ini, kami juga mengupgrade klaster DB yang dipulihkan dari Aurora MySQL versi 2 (kompatibel dengan MySQL 5.7) ke Aurora MySQL versi 3 (kompatibel dengan MySQL 8.0).

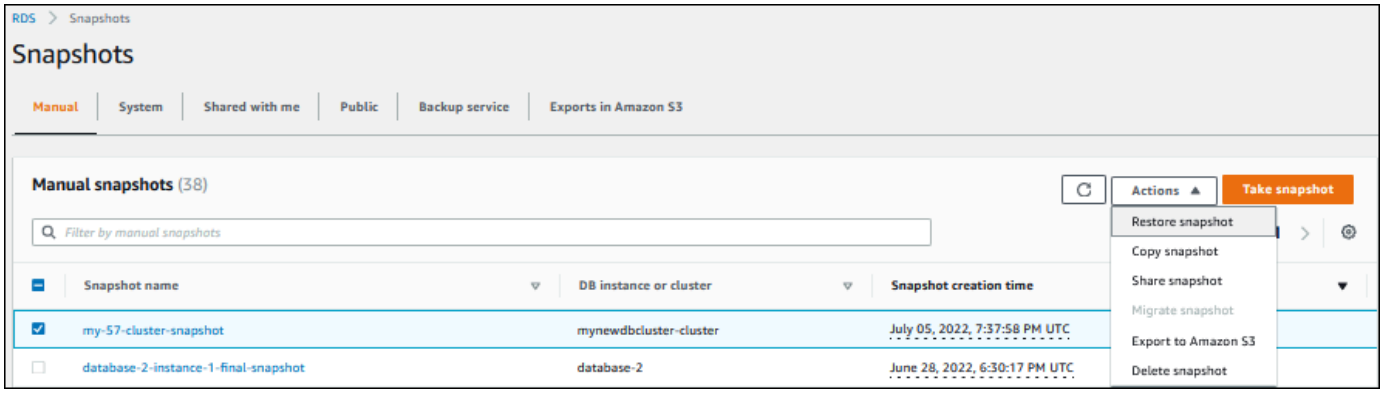
Memulihkan klaster DB dari snapshot klaster DB menggunakan konsol Amazon RDS

Saat Anda memulihkan klaster DB dari snapshot menggunakan AWS Management Console, instans DB primer (penulis) juga dibuat.

> **Note**
>
> Saat instans DB primer sedang dibuat, instans ini muncul sebagai intans pembaca, tetapi setelah pembuatan, intans ini menjadi instans penulis.

Untuk memulihkan klaster DB dari snapshot klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Snapshot.
3. Pilih snapshot klaster DB tempat asal pemulihan.
4. Untuk Tindakan, pilih Pulihkan snapshot.

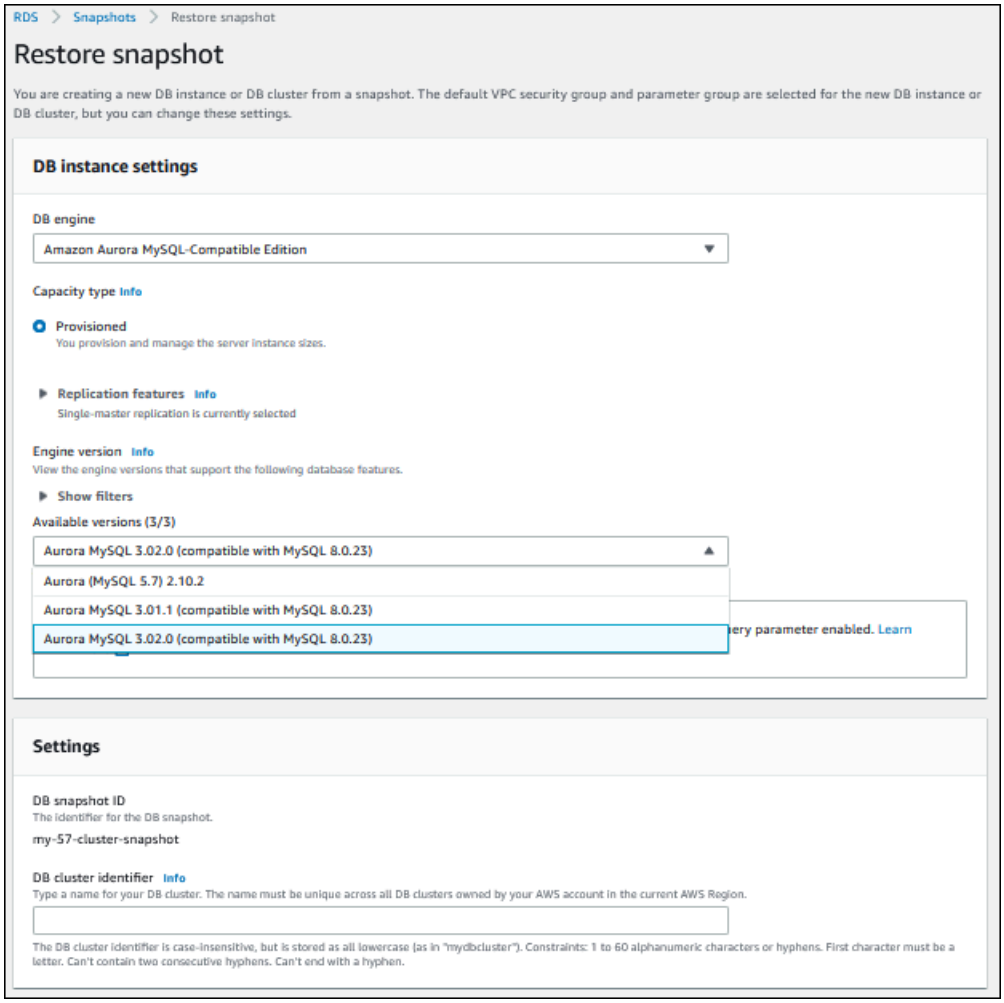


Halaman Pulihkan snapshot muncul.

5. Pada Pengaturan instans DB, lakukan tindakan berikut:

   a. Gunakan pengaturan default untuk mesin DB.

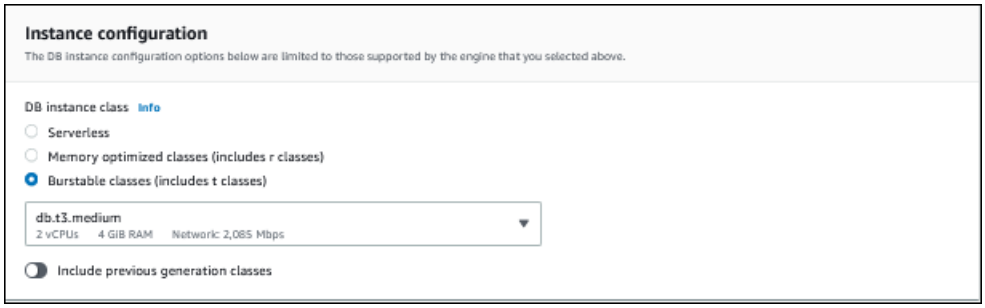
   b. Untuk Versi yang tersedia, pilih versi MySQL-8.0 yang kompatibel, seperti Aurora MySQL 3.02.0 (kompatibel dengan MySQL 8.0.23).



6. Di bagian Pengaturan, untuk ID klaster DB masukkan nama unik yang ingin digunakan untuk klaster DB yang dipulihkan, misalnya **`my-80-cluster`**.
7. Di bagian Konektivitas, gunakan pengaturan default untuk berikut ini:
   - Cloud privat virtual (VPC)
   - Grup subnet DB
   - Akses publik
   - Grup keamanan VPC (firewall)
8. Pilih Kelas instans DB.

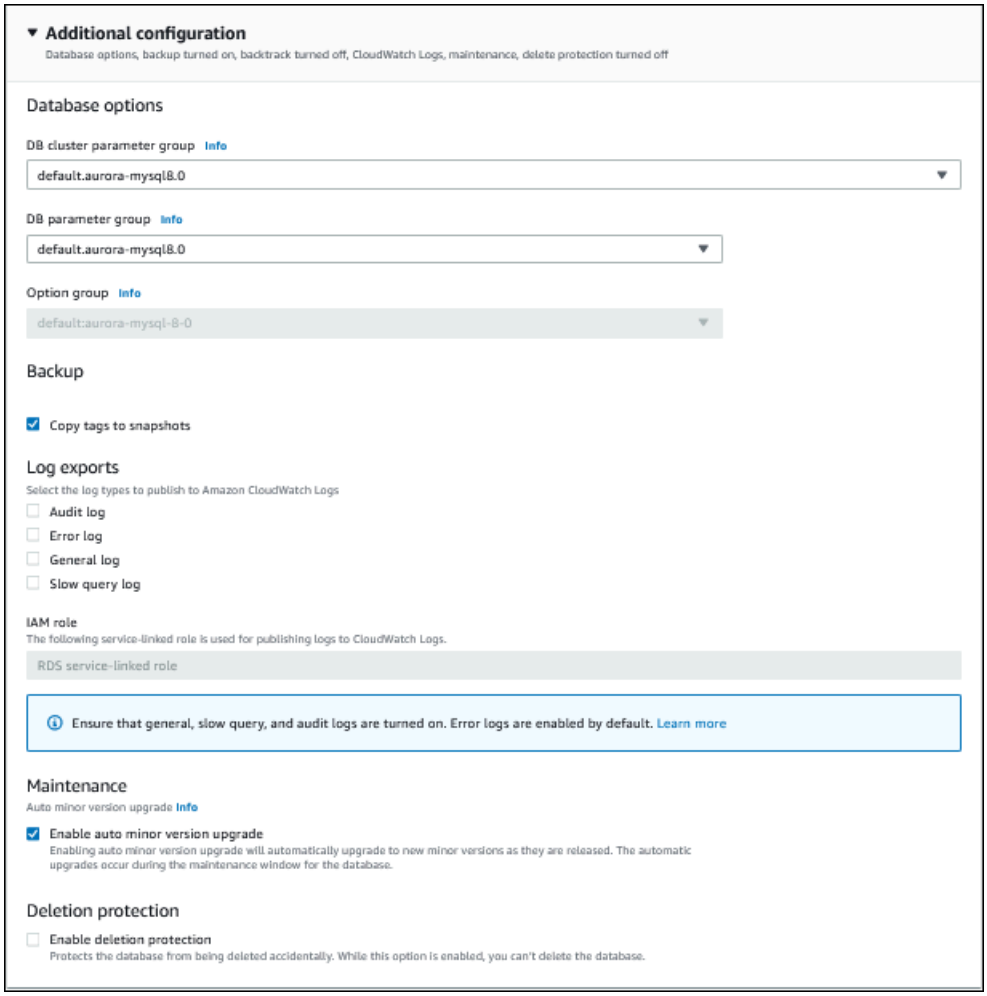
   Untuk tutorial ini, pilih Kelas runtutan (termasuk kelas t), lalu pilih db.t3.medium.

> **Note**
>
> Sebaiknya Anda menggunakan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail lebih lanjut tentang kelas instans T, lihat Jenis kelas instans DB.



9. Untuk Autentikasi basis data, gunakan pengaturan default.
10. Untuk Enkripsi, gunakan pengaturan default.

    Jika klaster DB sumber untuk snapshot dienkripsi, instans DB yang dipulihkan juga dienkripsi. Anda tidak dapat membuatnya tidak terenkripsi.
11. Luaskan Konfigurasi tambahan di bagian bawah halaman.



12. Buat pilihan berikut:

    a. Untuk tutorial ini, gunakan nilai default untuk grup parameter klaster DB.

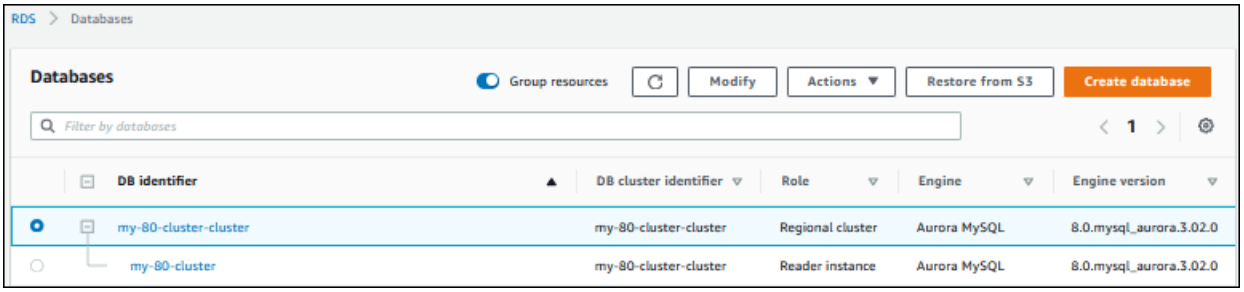
    b. Untuk tutorial ini, gunakan nilai default untuk grup parameter DB.

    c. Untuk Ekspor log, pilih semua kotak centang.

    d. Untuk Perlindungan penghapusan, pilih kotak centang Aktifkan perlindungan penghapusan.

13. Pilih Pulihkan instans DB.

Halaman Basis Data menampilkan klaster DB yang dipulihkan, dengan status `Creating`.



Saat instans DB primer sedang dibuat, instans ini muncul sebagai intans pembaca, tetapi setelah pembuatan, intans ini menjadi instans penulis.

Memulihkan klaster DB dari snapshot klaster DB menggunakan AWS CLI

Memulihkan klaster DB dari snapshot menggunakan AWS CLI terdiri dari dua langkah:

1. Memulihkan klaster DBmenggunakan perintah restore-db-cluster-from-snapshot
2. Membuat instans DB primer (penulis)menggunakan create-db-instanceperintah

Memulihkan klaster DB

Gunakan perintah `restore-db-cluster-from-snapshot`. Opsi berikut diperlukan:

- `--db-cluster-identifier` – Nama klaster DB yang dipulihkan.
- `--snapshot-identifier` – Nama snapshot DB tempat asal pemulihan.
- `--engine` – Mesin basis data klaster DB yang dipulihkan. Ini harus kompatibel dengan mesin basis data klaster DB sumber.

  Pilihannya adalah sebagai berikut:

  - `aurora-mysql` – Aurora MySQL 5.7 dan 8.0 yang kompatibel.
  - `aurora-postgresql` – Aurora PostgreSQL yang kompatibel.

  Dalam contoh ini, kami menggunakan `aurora-mysql`.
- `--engine-version` – Versi klaster DB yang dipulihkan. Dalam contoh ini, kami menggunakan versi MySQL-8.0 yang kompatibel.

Contoh berikut memulihkan klaster DB yang kompatibel dengan Aurora MySQL 8.0 yang bernama `my-new-80-cluster` dari snapshot klaster DB bernama `my-57-cluster-snapshot`.

Untuk memulihkan klaster DB

- Gunakan salah satu perintah berikut ini.

  Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier my-new-80-cluster \
    --snapshot-identifier my-57-cluster-snapshot \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.02.0
```

  Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier my-new-80-cluster ^
    --snapshot-identifier my-57-cluster-snapshot ^
    --engine aurora-mysql ^
    --engine-version 8.0.mysql_aurora.3.02.0
```

Output-nya menyerupai berikut.

```
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "eu-central-1b",
            "eu-central-1c",
            "eu-central-1a"
        ],
        "BackupRetentionPeriod": 14,
        "DatabaseName": "",
        "DBClusterIdentifier": "my-new-80-cluster",
        "DBClusterParameterGroup": "default.aurora-mysql8.0",
        "DBSubnetGroup": "default",
        "Status": "creating",
        "Endpoint": "my-new-80-cluster.cluster-############.eu-
central-1.rds.amazonaws.com",
        "ReaderEndpoint": "my-new-80-cluster.cluster-ro-############.eu-
central-1.rds.amazonaws.com",
        "MultiAZ": false,
```

```
        "Engine": "aurora-mysql",
        "EngineVersion": "8.0.mysql_aurora.3.02.0",
        "Port": 3306,
        "MasterUsername": "admin",
        "PreferredBackupWindow": "01:55-02:25",
        "PreferredMaintenanceWindow": "thu:21:14-thu:21:44",
        "ReadReplicaIdentifiers": [],
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "HostedZoneId": "Z1RLNU0EXAMPLE",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:123456789012:key/#######-5ccc-49cc-8aaa-
############",
        "DbClusterResourceId": "cluster-ZZ12345678ITSJUSTANEXAMPLE",
        "DBClusterArn": "arn:aws:rds:eu-central-1:123456789012:cluster:my-new-80-
cluster",
        "AssociatedRoles": [],
        "IAMDatabaseAuthenticationEnabled": false,
        "ClusterCreateTime": "2022-07-05T20:45:42.171000+00:00",
        "EngineMode": "provisioned",
        "DeletionProtection": false,
        "HttpEndpointEnabled": false,
        "CopyTagsToSnapshot": false,
        "CrossAccountClone": false,
        "DomainMemberships": [],
        "TagList": []
    }
}
```

Membuat instans DB primer (penulis)

Untuk membuat instans DB primer (penulis), gunakan perintah `create-db-instance`. Opsi berikut diperlukan:

- `--db-cluster-identifier` – Nama klaster DB yang dipulihkan.
- `--db-instance-identifier` – Nama instans DB primer.

- `--db-instance-class` – Kelas instans dari instans DB primer. Dalam contoh ini, kami menggunakan `db.t3.medium`.

  > **Note**
  >
  > Sebaiknya Anda menggunakan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail lebih lanjut tentang kelas instans T, lihat Jenis kelas instans DB.

- `--engine` – Mesin basis data instans DB primer. Ini harus berupa mesin basis data yang sama dengan yang digunakan klaster DB yang dipulihkan.

  Pilihannya adalah sebagai berikut:

  - `aurora-mysql` – Aurora MySQL 5.7 dan 8.0 yang kompatibel.
  - `aurora-postgresql` – Aurora PostgreSQL yang kompatibel.

  Dalam contoh ini, kami menggunakan `aurora-mysql`.

Contoh berikut membuat instans DB primer (penulis) bernama `my-new-80-cluster-instance` dalam klaster DB yang kompatibel dengan Aurora MySQL 8.0 yang dipulihkan bernama `my-new-80-cluster`.

Untuk membuat instans DB primer

- Gunakan salah satu perintah berikut ini.

  Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-80-cluster \
    --db-instance-identifier my-new-80-cluster-instance \
    --db-instance-class db.t3.medium \
    --engine aurora-mysql
```

  Untuk Windows:

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-80-cluster ^
    --db-instance-identifier my-new-80-cluster-instance ^
```

```
    --db-instance-class db.t3.medium ^
    --engine aurora-mysql
```

Output-nya menyerupai berikut.

```
{
    "DBInstance": {
        "DBInstanceIdentifier": "my-new-80-cluster-instance",
        "DBInstanceClass": "db.t3.medium",
        "Engine": "aurora-mysql",
        "DBInstanceStatus": "creating",
        "MasterUsername": "admin",
        "AllocatedStorage": 1,
        "PreferredBackupWindow": "01:55-02:25",
        "BackupRetentionPeriod": 14,
        "DBSecurityGroups": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-########",
                "Status": "active"
            }
        ],
        "DBParameterGroups": [
            {
                "DBParameterGroupName": "default.aurora-mysql8.0",
                "ParameterApplyStatus": "in-sync"
            }
        ],
        "DBSubnetGroup": {
            "DBSubnetGroupName": "default",
            "DBSubnetGroupDescription": "default",
            "VpcId": "vpc-2305ca49",
            "SubnetGroupStatus": "Complete",
            "Subnets": [
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1a"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                },
```

```
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1b"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                },
                {
                    "SubnetIdentifier": "subnet-########",
                    "SubnetAvailabilityZone": {
                        "Name": "eu-central-1c"
                    },
                    "SubnetOutpost": {},
                    "SubnetStatus": "Active"
                }
            ]
        },
        "PreferredMaintenanceWindow": "sat:02:41-sat:03:11",
        "PendingModifiedValues": {},
        "MultiAZ": false,
        "EngineVersion": "8.0.mysql_aurora.3.02.0",
        "AutoMinorVersionUpgrade": true,
        "ReadReplicaDBInstanceIdentifiers": [],
        "LicenseModel": "general-public-license",
        "OptionGroupMemberships": [
            {
                "OptionGroupName": "default:aurora-mysql-8-0",
                "Status": "in-sync"
            }
        ],
        "PubliclyAccessible": false,
        "StorageType": "aurora",
        "DbInstancePort": 0,
        "DBClusterIdentifier": "my-new-80-cluster",
        "StorageEncrypted": true,
        "KmsKeyId": "arn:aws:kms:eu-central-1:534026745191:key/#######-5ccc-49cc-8aaa-
############",
        "DbiResourceId": "db-5C6UT5PU0YETANOTHEREXAMPLE",
        "CACertificateIdentifier": "rds-ca-2019",
        "DomainMemberships": [],
        "CopyTagsToSnapshot": false,
        "MonitoringInterval": 0,
        "PromotionTier": 1,
```

```
        "DBInstanceArn": "arn:aws:rds:eu-central-1:123456789012:db:my-new-80-cluster-
instance",
        "IAMDatabaseAuthenticationEnabled": false,
        "PerformanceInsightsEnabled": false,
        "DeletionProtection": false,
        "AssociatedRoles": [],
        "TagList": []
    }
}
```

Memantau metrik di klaster Amazon Aurora

Amazon Aurora menggunakan klaster server basis data yang direplikasi. Pemantauan klaster Aurora biasanya memerlukan pemeriksaan kesehatan pada beberapa instans DB. Instans tersebut mungkin memiliki peran khusus, menangani sebagian besar operasi tulis, operasi hanya baca, atau gabungan keduanya. Anda juga memantau kesehatan keseluruhan klaster dengan mengukur jeda replikasi. Ini adalah jumlah waktu untuk perubahan yang diterapkan oleh satu instans DB agar tersedia bagi instans lainnya.

Topik

- Ikhtisar metrik pemantauan di Amazon Aurora
- Melihat status cluster
- Melihat dan menanggapi rekomendasi Amazon RDS
- Melihat metrik di konsol Amazon RDS
- Menampilkan metrik gabungan di konsol Amazon RDS
- Memantau metrik Amazon Aurora dengan Amazon CloudWatch
- Memantau muatan DB dengan Wawasan Performa di Amazon Aurora
- Menganalisis anomali kinerja dengan Amazon DevOps Guru untuk Amazon RDS
- Memantau ancaman dengan Amazon GuardDuty RDS Protection
- Memantau metrik OS dengan Pemantauan yang Disempurnakan
- Referensi metrik untuk Amazon Aurora

Ikhtisar metrik pemantauan di Amazon Aurora

Pemantauan adalah bagian penting dari upaya memelihara keandalan, ketersediaan, dan kinerja Amazon Aurora dan solusi AWS Anda. Agar dapat menelusuri dengan mudah kegagalan multipoin, kami menganjurkan supaya Anda mengumpulkan data pemantauan dari semua bagian solusi AWS Anda.

Topik

- Rencana pemantauan
- Garis dasar kinerja
- Pedoman kinerja
- Alat-alat pemantauan

Rencana pemantauan

Sebelum Anda memulai pemantauan Amazon Aurora, buat rencana pemantauan. Rencana ini sepatutnya menjawab pertanyaan-pertanyaan berikut:

- Apa sajakah sasaran pemantauan Anda?
- Sumber daya manakah yang akan Anda pantau?
- Seberapa seringkah Anda akan memantau sumber daya ini?
- Apa sajakah alat pemantauan yang akan Anda gunakan?
- Siapakah yang akan melakukan tugas pemantauan?
- Siapakah yang harus diberi tahu apabila ada berjalan salah?

Garis dasar kinerja

Untuk mencapai semua sasaran pemantauan, Anda perlu menetapkan garis dasar. Untuk itu, ukur kinerja pada berbagai kondisi beban dan waktu di lingkungan Amazon Aurora Anda. Anda dapat memantau metrik-metrik seperti:

- Throughput jaringan
- Koneksi klien
- I/O untuk operasi baca, tulis, atau metadata
- Keseimbangan kredit lonjakan untuk instans basis data Anda

Kami menyarankan agar Anda menyimpan data riwayat kinerja untuk Amazon Aurora. Dengan menggunakan data yang disimpan, Anda dapat membandingkan kinerja saat ini dengan tren masa lalu. Anda juga dapat membedakan pola kinerja normal dari anomali, dan merancang teknik untuk mengatasi masalah.

Pedoman kinerja

Secara umum, nilai-nilai yang diperkenankan untuk metrik kinerja bergantung pada kinerja aplikasi Anda secara relatif terhadap garis dasar Anda. Selidiki variansi yang konsisten atau sedang tren dari garis dasar Anda. Metrik-metrik berikut sering menjadi sumber masalah kinerja:

- Konsumsi CPU atau RAM tinggi – Nilai-nilai tinggi untuk konsumsi CPU atau RAM mungkin layak, jika keduanya mengikuti sasaran untuk aplikasi Anda (seperti throughput atau konkurensi) dan diharapkan.
- Konsumsi ruang disk – Selidiki konsumsi ruang disk jika ruang yang digunakan selalu berada pada atau di atas 85 persen dari total ruang disk. Lihat apakah menghapus data dari instans atau mengarsipkan data ke sistem yang lain layak dilakukan guna melegakan ruang.
- Lalu lintas jaringan – Untuk lalu lintas jaringan, bicaralah dengan administrator sistem Anda untuk memahami throughput yang diharapkan bagi jaringan domain dan koneksi internet Anda. Selidiki lalu lintas jaringan jika throughput selalu di bawah yang diharapkan.
- Koneksi basis data – Jika Anda melihat jumlah koneksi pengguna yang tinggi dan juga penurunan kinerja dan waktu respons instans, pertimbangkan untuk membatasi koneksi basis data. Jumlah koneksi pengguna terbaik untuk instans basis data bervariasi berdasarkan kelas instans dan kerumitan operasi yang dilakukan. Untuk menentukan jumlah koneksi basis data, kaitkan instans basis data Anda dengan grup parameter dengan parameter `User Connections` diatur ke nilai selain 0 (tidak terbatas). Anda dapat menggunakan grup parameter yang ada atau membuat grup baru. Untuk informasi selengkapnya, lihat Bekerja dengan grup parameter.
- Metrik IOPS – Nilai yang diharapkan untuk metrik IOPS bergantung pada spesifikasi disk dan konfigurasi server; jadi, gunakan garis dasar Anda untuk mengetahui nilai yang lazim. Selidiki apakah nilai-nilai selalu berbeda dengan garis dasar Anda. Untuk kinerja IOPS terbaik, pastikan bahwa set kerja Anda yang biasa sesuai dengan memori untuk meminimalkan operasi baca dan tulis.

Ketika kinerja berada di luar garis dasar yang telah ditetapkan, Anda mungkin perlu membuat perubahan untuk mengoptimalkan ketersediaan basis data bagi beban kerja Anda. Misalnya, Anda

mungkin perlu mengubah kelas instans dari instans basis data Anda. Atau Anda mungkin perlu mengubah jumlah instans basis data dan replika baca yang tersedia untuk klien.

Alat-alat pemantauan

Pemantauan adalah bagian penting dari upaya memelihara keandalan, ketersediaan, dan kinerja Amazon Aurora dan solusi Anda AWS yang lain. AWS menyediakan berbagai alat pemantauan untuk mengawasi Amazon Aurora, melaporkan saat ada yang tidak beres, dan mengambil tindakan otomatis jika layak.

Topik

- Alat-alat pemantauan otomatis
- Alat-alat pemantauan manual

Alat-alat pemantauan otomatis

Kami menyarankan agar Anda mengotomatiskan tugas-tugas pemantauan sebanyak mungkin.

Topik

- Status dan rekomendasi klaster Amazon Aurora
- CloudWatch Metrik Amazon untuk
- Wawasan Performa Amazon RDS dan pemantauan sistem operasi
- Layanan terintegrasi

Status dan rekomendasi klaster Amazon Aurora

Anda dapat menggunakan alat-alat otomatis berikut untuk memantau Amazon Aurora dan melaporkan saat ada yang salah:

- Status klaster Amazon Aurora — Lihat detail status instans Anda saat ini dengan menggunakan konsol Amazon RDS, AWS CLI, atau API RDS.
- Rekomendasi Amazon Aurora — Tanggapi rekomendasi otomatis untuk sumber daya basis data, seperti instans basis data, klaster basis data, , dan grup parameter klaster basis data. Untuk informasi selengkapnya, lihat Melihat dan menanggapi rekomendasi Amazon RDS.

CloudWatch Metrik Amazon untuk

Amazon Aurora terintegrasi dengan CloudWatch Amazon untuk kemampuan pemantauan tambahan.

- Amazon CloudWatch — Layanan ini memantau AWS sumber daya Anda dan aplikasi yang Anda jalankan AWS secara real time. Anda dapat menggunakan CloudWatch fitur Amazon berikut dengan Amazon :
  - CloudWatch Metrik Amazon — Aurora secara otomatis mengirimkan metrik ke setiap menit CloudWatch untuk setiap basis data aktif. Anda tidak mendapatkan biaya tambahan untuk metrik Amazon RDS di. CloudWatch Lihat informasi yang lebih lengkap di CloudWatch Metrik Amazon untuk Amazon Aurora
  - CloudWatch Alarm Amazon - Anda dapat menonton satu metrik Amazon Aurora selama periode waktu tertentu. Anda lalu dapat melakukan satu atau beberapa tindakan berdasarkan nilai metrik itu relatif terhadap ambang batas yang Anda tetapkan.

Wawasan Performa Amazon RDS dan pemantauan sistem operasi

Anda dapat menggunakan alat-alat otomatis berikut untuk memantau kinerja Amazon Aurora:

- Wawasan Performa Amazon RDS – Telaah beban pada basis data Anda, dan tentukan kapan dan di mana harus mengambil tindakan. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.
- Pemantauan Disempurnakan Amazon RDS – Lihat secara waktu nyata metrik-metrik untuk sistem operasi. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan.

Layanan terintegrasi

Layanan AWS berikut terintegrasi dengan Amazon Aurora:

- Amazon EventBridge adalah layanan bus acara tanpa server yang memudahkan untuk menghubungkan aplikasi Anda dengan data dari berbagai sumber. Untuk informasi selengkapnya, lihat Memantau peristiwa Amazon Aurora.
- Amazon CloudWatch Logs memungkinkan Anda memantau, menyimpan, dan mengakses file log Anda dari instans Amazon Aurora CloudTrail, dan sumber lainnya. Untuk informasi selengkapnya, lihat Memantau file log Amazon Aurora.

- AWS CloudTrail menangkap panggilan API dan peristiwa terkait yang dibuat oleh atau atas nama Akun AWS Anda dan mengirimkan berkas log ke bucket Amazon S3 yang Anda tentukan. Untuk informasi selengkapnya, lihat Memantau panggilan API Amazon Aurora di AWS CloudTrail.
- Aliran Aktivitas Database Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.
- DevOpsGuru for RDS adalah kemampuan Amazon DevOps Guru yang menerapkan pembelajaran mesin ke metrik Performance Insights untuk database Amazon Aurora. Untuk informasi selengkapnya, lihat Menganalisis anomali kinerja dengan Amazon DevOps Guru untuk Amazon RDS.

Alat-alat pemantauan manual

Anda perlu memonitor secara manual item yang tidak CloudWatch tercakup oleh alarm. Amazon RDS, CloudWatch, AWS Trusted Advisor dan dasbor AWS konsol lainnya memberikan at-a-glance tampilan keadaan lingkungan AndaAWS. Kami menyarankan agar Anda juga memeriksa file log pada instans basis data Anda.

- Dari konsol Amazon RDS, Anda dapat memantau butir-butir berikut untuk sumber daya Anda:
  - Jumlah koneksi dengan instans basis data
  - Jumlah operasi baca dan tulis ke instans basis data
  - Jumlah penyimpanan yang saat ini digunakan oleh instans basis data
  - Jumlah memori dan CPU yang sedang digunakan untuk instans basis data
  - Jumlah lalu lintas jaringan ke dan dari instans basis data
- Dari dasbor Trusted Advisor, Anda dapat meninjau pemeriksan-pemeriksaan optimasi biaya, keamanan, toleransi kesalahan, dan peningkatan kinerja berikut:
  - Amazon RDS Idle DB Instances
  - Amazon RDS Security Group Access Risk
  - Amazon RDS Backups
  - Amazon RDS Multi-AZ
  - Aurora DB Instance Accessibility

  Lihat informasi yang lebih lengkap tentang pemeriksaan-pemeriksaan ini di Praktik terbaik (pemeriksaan) Trusted Advisor.
- CloudWatch halaman rumah menunjukkan:

- Alarm dan status saat ini
- Grafik alarm dan sumber daya
- Status kesehatan layanan

Selain itu, Anda dapat menggunakan CloudWatch untuk melakukan hal berikut:

- Membuat dasbor yang disesuaikan untuk memantau layanan yang Anda pedulikan.
- Data metrik grafik untuk memecahkan masalah dan mengungkap tren.
- Mencari dan menelusuri semua metrik sumber daya AWS Anda.
- Membuat dan mengedit alarm agar diberi tahu tentang masalah.

Melihat status cluster

Menggunakan konsol Amazon RDS, Anda dapat dengan cepat mengakses status cluster DB Anda.

Topik

- Melihat klaster DB Amazon Aurora
- Melihat status klaster DB
- Melihat status instans DB di klaster Aurora

Melihat klaster DB Amazon Aurora

Anda memiliki beberapa opsi untuk melihat informasi tentang klaster DB Amazon Aurora dan instans DB dalam klaster DB Anda.

- Anda dapat melihat klaster DB dan instans DB di konsol Amazon RDS dengan memilih Basis data dari panel navigasi.
- Anda bisa mendapatkan cluster DB dan informasi instans DB menggunakan AWS Command Line Interface (AWS CLI).
- Anda bisa mendapatkan informasi klaster DB dan instans DB menggunakan API Amazon RDS.

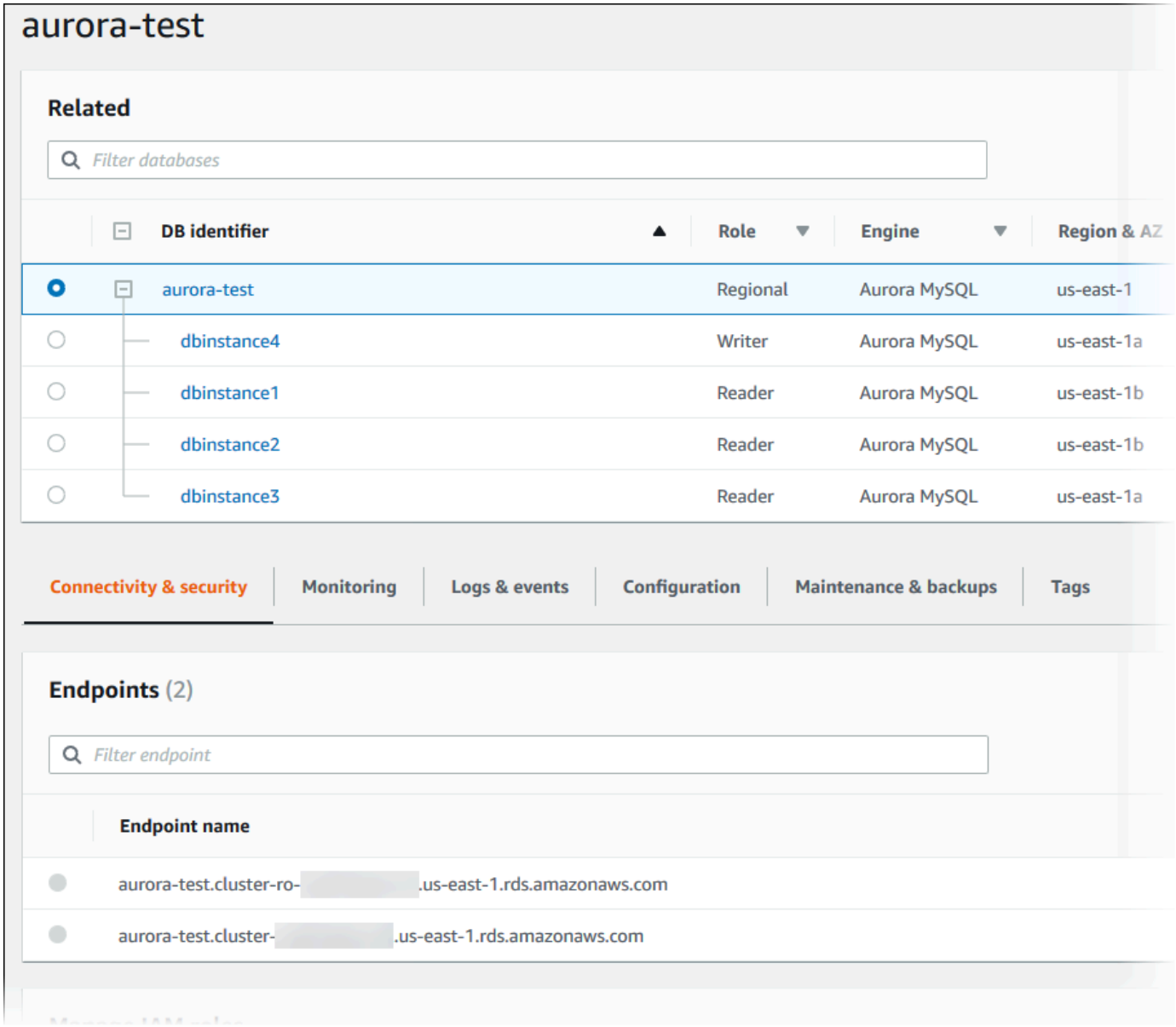
Konsol

Di konsol Amazon RDS, Anda dapat melihat detail tentang klaster DB dengan memilih Basis data dari panel navigasi konsol. Anda juga dapat melihat detail tentang instans DB yang merupakan anggota klaster DB Amazon Aurora DB.

Untuk melihat atau memodifikasi klaster DB di konsol Amazon RDS

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih nama klaster DB Aurora yang ingin Anda lihat dari daftar.

   Misalnya, gambar berikut menunjukkan halaman detail untuk klaster DB bernama `aurora-test`. Klaster DB ini menampilkan empat instans DB dalam daftar pengidentifikasi DB. Instans DB penulis, yakni `dbinstance4`, merupakan instans DB utama untuk klaster DB.



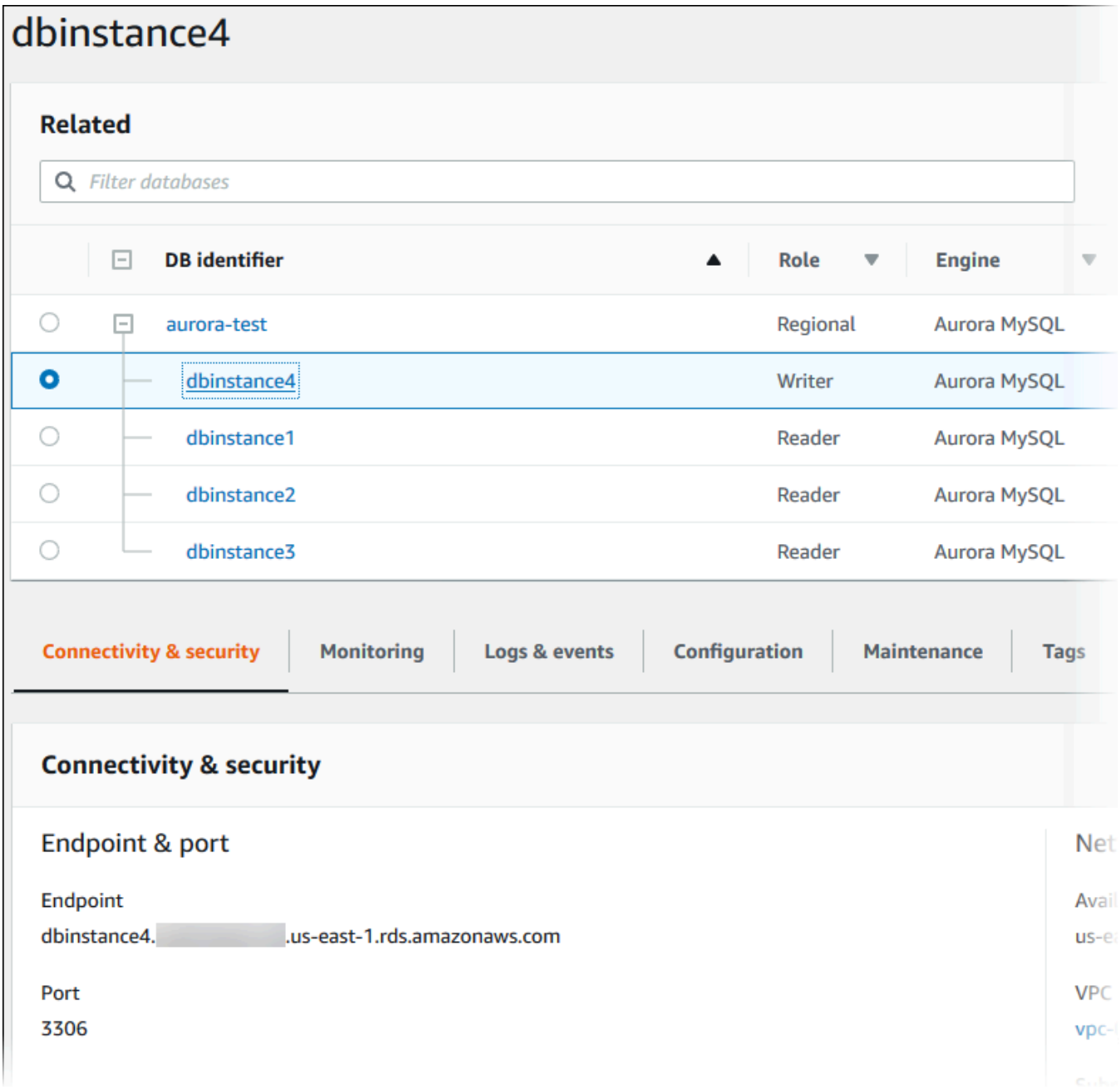
4. Untuk memodifikasi klaster DB, pilih klaster DB dari daftar dan pilih Modifikasi.

Untuk melihat atau memodifikasi instans DB klaster DB di konsol Amazon RDS

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Lakukan salah satu hal berikut:

   - Untuk melihat instans DB, pilih salah satu dari daftar yang merupakan anggota klaster DB Aurora.

Misalnya, jika Anda memilih pengidentifikasi instans DB `dbinstance4`, konsol menampilkan halaman detail untuk instans DB `dbinstance4`, seperti yang ditampilkan di gambar berikut.



- Untuk memodifikasi instans DB, pilih instans DB dari daftar dan pilih Modifikasi. Untuk informasi selengkapnya tentang cara memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

AWS CLI

Untuk melihat informasi cluster DB dengan menggunakan AWS CLI, gunakan describe-db-clustersperintah. Misalnya, AWS CLI perintah berikut mencantumkan informasi cluster DB untuk semua cluster DB di `us-east-1` wilayah modifikasi untuk AWS akun yang dikonfigurasi.

```
aws rds describe-db-clusters --region us-east-1
```

Perintah mengembalikan output berikut jika Anda AWS CLI dikonfigurasi untuk output JSON.

```
{
    "DBClusters": [
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "sample-cluster1.cluster-123456789012.us-
east-1.rds.amazonaws.com"
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "sample-cluster1",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T03:35:42.563Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-replica"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "sample-primary"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "03:34-04:04",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-ddb65fec"
                }
            ],
            "DBSubnetGroup": "default",
            "StorageEncrypted": false,
```

```
            "DatabaseName": "sample",
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:06:08.903Z",
            "PreferredMaintenanceWindow": "wed:08:15-wed:08:45"
        },
        {
            "Status": "available",
            "Engine": "aurora-mysql",
            "Endpoint": "aurora-sample.cluster-123456789012.us-
east-1.rds.amazonaws.com",
            "AllocatedStorage": 1,
            "DBClusterIdentifier": "aurora-sample-cluster",
            "MasterUsername": "mymasteruser",
            "EarliestRestorableTime": "2023-03-30T10:21:34.826Z",
            "DBClusterMembers": [
                {
                    "IsClusterWriter": false,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-replica-sample"
                },
                {
                    "IsClusterWriter": true,
                    "DBClusterParameterGroupStatus": "in-sync",
                    "DBInstanceIdentifier": "aurora-sample"
                }
            ],
            "Port": 3306,
            "PreferredBackupWindow": "10:20-10:50",
            "VpcSecurityGroups": [
                {
                    "Status": "active",
                    "VpcSecurityGroupId": "sg-55da224b"
                }
            ],
            "DBSubnetGroup": "default",
            "StorageEncrypted": false,
            "DatabaseName": "sample",
```

```
            "EngineVersion": "5.7.mysql_aurora.2.11.0",
            "DBClusterParameterGroup": "default.aurora-mysql5.7",
            "BackupRetentionPeriod": 1,
            "AvailabilityZones": [
                "us-east-1b",
                "us-east-1c",
                "us-east-1d"
            ],
            "LatestRestorableTime": "2023-03-31T20:00:11.491Z",
            "PreferredMaintenanceWindow": "sun:03:53-sun:04:23"
        }
    ]
}
```

API RDS

Untuk melihat informasi klaster DB menggunakan API Amazon RDS, gunakan operasi DescribeDBClusters.

Melihat status klaster DB

Status klaster DB menunjukkan kondisinya. Anda dapat melihat status cluster DB dan instance cluster dengan menggunakan konsol Amazon RDS, the AWS CLI, atau API.

> **Note**
>
> Aurora juga menggunakan status lain yang disebut status pemeliharaan, yang ditunjukkan di kolom Pemeliharaan pada konsol Amazon RDS. Nilai ini menunjukkan status patch pemeliharaan yang perlu diterapkan ke klaster DB. Status pemeliharaan bergantung pada status klaster DB. Untuk informasi lebih lanjut tentang status pemeliharaan, lihat Menerapkan pembaruan untuk klaster DB.

Temukan kemungkinan nilai status untuk klaster DB dalam tabel berikut.

Status klaster DB	Ditagih	Deskripsi
Available	Ditagih	Klaster DB berkondisi baik dan tersedia. Ketika klaster Nirserver Aurora tersedia dan dijeda, Anda akan ditagih hanya untuk penyimpanan.
Backing-up	Ditagih	Klaster DB saat ini sedang dicadangkan.
Backtracking	Ditagih	Backtrack sedang dilakukan untuk klaster DB. Status ini hanya berlaku untuk Aurora MySQL.
Cloning-failed	Tidak ditagih	Klaster DB gagal dikloning.
Creating	Tidak ditagih	Klaster DB sedang dibuat. Klaster DB tidak dapat diakses saat sedang dibuat.
Deleting	Tidak ditagih	Klaster DB sedang dihapus.
Failing-over	Ditagih	Failover dari instans utama ke Aurora Replica sedang dilakukan.

Status klaster DB	Ditagih	Deskripsi
Aku naccessible-encryption-credentials	Tidak ditagih	Yang AWS KMS key digunakan untuk mengenkripsi atau mendekripsi cluster DB tidak dapat diakses atau dipulihkan.
Aku naccessible-encryption-cred entials-recoverable	Ditagih untuk penyimpanan	Kunci KMS digunakan untuk mengenkripsi atau mendekripsi klaster DB tidak dapat diakses. Namun, jika kunci KMS aktif, Anda dapat memulai ulang klaster DB untuk memulihka nnya. Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.
Maintenance	Ditagih	Amazon RDS menerapkan pembaruan pemeliharaan pada klaster DB. Status ini digunakan untuk pemeliharaan tingkat klaster DB yang dijadwalkan RDS sejak jauh hari sebelumnya.
Migrating	Ditagih	Snapshot klaster DB sedang dipulihkan ke klaster DB.
Migration-failed	Tidak ditagih	Migrasi gagal.
Modifying	Ditagih	Klaster DB sedang dimodifikasi karena ada permintaan pelanggan untuk memodifikasi klaster DB.
Promoting	Ditagih	Replika baca sedang dipromosikan ke klaster DB mandiri.
P reparing-data-migr ation	Ditagih	Amazon RDS sedang bersiap untuk memigrasi kan data ke Aurora.
Renaming	Ditagih	Nama klaster DB sedang diganti karena ada permintaan pelanggan untuk mengganti namanya.

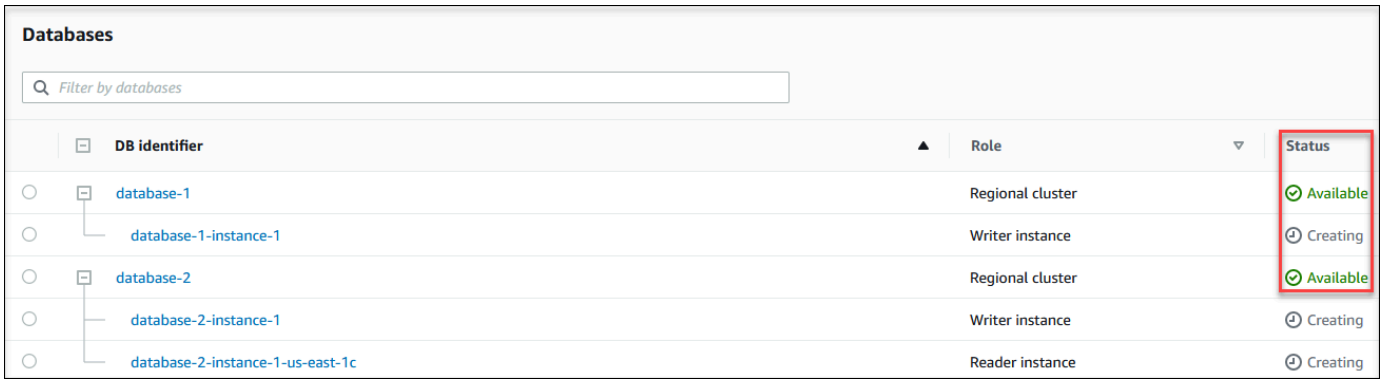
Status klaster DB	Ditagih	Deskripsi
R esetting-master-cr edentials	Ditagih	Kredensial master untuk klaster DB sedang direset karena ada permintaan pelanggan untuk meresetnya.
Starting	Ditagih untuk penyimpanan	Klaster DB dimulai.
Stopped	Ditagih untuk penyimpanan	Klaster DB dihentikan.
Stopping	Ditagih untuk penyimpanan	Klaster DB sedang dihentikan.
Storage-optimization	Ditagih	Instans DB Anda sedang dimodifikasi untuk mengubah ukuran atau jenis penyimpanan. Instans DB berfungsi sepenuhnya. Namun, meskipun status instans DB Anda adalah storage-optimization, Anda tidak dapat meminta perubahan apa pun untuk penyimpanan instans DB Anda. Proses pengoptimalan penyimpan an biasanya singkat, tetapi terkadang dapat memakan waktu hingga dan bahkan lebih dari 24 jam.
U pdate-iam-db-auth	Ditagih	Otorisasi IAM untuk klaster DB sedang diperbarui.
Meningkatkan	Ditagih	Versi mesin klaster DB sedang ditingkatkan.

Konsol

Untuk melihat status klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.

Halaman Basis Data muncul dengan daftar klaster DB. Nilai status untuk setiap klaster DB akan ditampilkan.



CLI

Untuk melihat hanya status cluster DB, gunakan kueri berikut di AWS CLI.

```
aws rds describe-db-clusters --query 'DBClusters[*].[DBClusterIdentifier,Status]' --
output table
```

Melihat status instans DB di klaster Aurora

Status instans DB dalam klaster Aurora menunjukkan kondisi instans DB. Anda dapat menggunakan prosedur berikut untuk melihat status instans DB klaster di konsol Amazon RDS, AWS CLI perintah, atau operasi API.

> **Note**
>
> Amazon RDS juga menggunakan status lain yang disebut status pemeliharaan, yang ditunjukkan di kolom Pemeliharaan pada konsol Amazon RDS. Nilai ini menunjukkan status patch pemeliharaan yang perlu diterapkan ke instans DB. Status pemeliharaan bergantung pada status instans DB. Untuk informasi lebih lanjut tentang status pemeliharaan, lihat Menerapkan pembaruan untuk klaster DB.

Temukan kemungkinan nilai status untuk instans DB dalam tabel berikut. Tabel ini juga menunjukkan apakah Anda akan ditagih untuk instans DB dan penyimpanan, ditagih hanya untuk penyimpanan, atau Anda tidak dikenai tagihan. Untuk semua status instans DB, Anda selalu ditagih untuk penggunaan cadangan.

Status instans DB	Ditagih	Deskripsi
Available	Ditagih	Instans DB berkondisi baik dan tersedia.
Backing-up	Ditagih	Instans DB saat ini sedang dicadangkan.
Backtracking	Ditagih	Backtrack sedang dilakukan untuk instans DB. Status ini hanya berlaku untuk Aurora MySQL.
C onfiguring-enhanced-monitoring	Ditagih	Pemantauan yang ditingkatkan diaktifkan atau dinonaktifkan untuk instans DB ini.
C onfiguring-iam-database-auth	Ditagih	AWS Identity and Access Management (IAM) otentikasi database sedang diaktifkan atau dinonaktifkan untuk instans DB ini.
C onfiguring-log-exports	Ditagih	Menerbitkan file log ke Amazon CloudWatch Logs sedang diaktifkan atau dinonaktifkan untuk instans DB ini.

Status instans DB	Ditagih	Deskripsi
C onverting-to-vpc	Ditagih	Instans DB sedang dikonversi dari instans DB yang tidak ada di Amazon Virtual Private Cloud (Amazon VPC) menjadi instans DB yang ada di Amazon VPC.
Creating	Tidak ditagih	Instans DB sedang dibuat. Instans DB tidak dapat diakses saat sedang dibuat.
Delete-precheck	Tidak ditagih	Amazon RDS memvalidasi bahwa replika baca berkondisi baik dan aman untuk dihapus.
Deleting	Tidak ditagih	Instans DB sedang dihapus.
Failed	Tidak ditagih	Instans DB gagal dan Amazon RDS tidak dapat dipulihkan. Lakukan point-in-time pemulihan ke waktu restorable terbaru dari instans DB untuk memulihkan data.
Aku naccessible-encryption-credentials	Tidak ditagih	Yang AWS KMS key digunakan untuk mengenkripsi atau mendekripsi instans DB tidak dapat diakses atau dipulihkan.
Aku naccessible-encryption-credentials-recoverable	Ditagih untuk penyimpan	Kunci KMS digunakan untuk mengenkripsi atau mendekripsi instans DB tidak dapat diakses. Namun, jika kunci KMS aktif, Anda dapat memulai ulang instans DB untuk memulihkannya. Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.
Incompatible-network	Tidak ditagih	Amazon RDS mencoba melakukan tindakan pemulihan di instans DB tetapi tidak dapat melakukannya karena VPC berada dalam keadaan yang mencegah penyelesaian tindakan. Status ini dapat terjadi jika, misalnya, semua alamat IP yang tersedia di subnet sedang digunakan dan Amazon RDS tidak bisa mendapatkan alamat IP untuk instans DB.

Status instans DB	Ditagih	Deskripsi
Aku ncompatible-option-group	Ditagih	Amazon RDS mencoba menerapkan perubahan grup opsi tetapi tidak dapat melakukannya, dan Amazon RDS tidak dapat melakukan roll back ke status grup opsi sebelumnya. Untuk informasi selengkapnya, lihat daftar Peristiwa Terbaru untuk instans DB. Status ini dapat terjadi jika, misalnya, grup opsi berisi opsi seperti TDE dan instans DB tidak berisi informasi terenkripsi.
Incompatible-parameters	Ditagih	Amazon RDS tidak dapat memulai instans DB karena parameter yang ditentukan dalam grup parameter DB instans DB tidak kompatibel dengan instans DB. Kembalikan perubahan parameter atau buat parameter kompatibel dengan instans DB untuk mendapatkan akses ke instans DB Anda. Untuk informasi selengkapnya tentang parameter yang tidak kompatibel, periksa daftar Peristiwa Terbaru untuk instans DB.
Incompatible-restore	Tidak ditagih	Amazon RDS tidak dapat melakukan point-in-time pemulihan. Penyebab umum untuk status ini termasuk penggunaan tabel sementara atau penggunaan tabel MyISAM dengan MySQL.
Insufficient-capacity	Tidak ditagih	Amazon RDS tidak dapat membuat instans karena kapasitas yang tersedia saat ini tidak cukup. Untuk membuat instans DB di AZ yang sama dengan jenis instans yang sama, hapus instans DB Anda, tunggu beberapa jam, lalu coba untuk membuat lagi. Atau, buat instans baru menggunakan kelas instans atau AZ yang berbeda.
Maintenance	Ditagih	Amazon RDS menerapkan pembaruan pemeliharaan pada instans DB. Status ini digunakan untuk pemeliharaan tingkat instans yang dijadwalkan RDS sejak jauh hari sebelumnya.
Modifying	Ditagih	Instans DB sedang dimodifikasi karena ada permintaan pelanggan untuk memodifikasi instans DB.
M oving-to-vpc	Ditagih	Instans DB sedang dipindahkan ke Amazon Virtual Private Cloud (Amazon VPC) baru.

Status instans DB	Ditagih	Deskripsi
Rebooting	Ditagih	Instans DB sedang di-boot ulang karena permintaan pelanggan atau proses Amazon RDS yang memerlukan boot ulang instans DB.
R esetting-master-cr edentials	Ditagih	Kredensial master untuk instans DB sedang direset karena ada permintaan pelanggan untuk meresetnya.
Renaming	Ditagih	Nama instans DB sedang diganti karena ada permintaan pelanggan untuk mengganti namanya.
Restore-error	Ditagih	Instans DB mengalami kesalahan saat mencoba mengembal ikan ke point-in-time atau dari snapshot.
Starting	Ditagih untuk penyim an	Instans DB dimulai.
Stopped	Ditagih untuk penyim an	Instans DB dihentikan.
Stopping	Ditagih untuk penyim an	Instans DB sedang dihentikan.
S torage-config-upgr ade	Ditagih	Konfigurasi sistem file penyimpanan instans DB sedang ditingkatkan. Status ini hanya berlaku untuk basis data green dalam deployment blue/green, atau untuk replika baca instans DB.

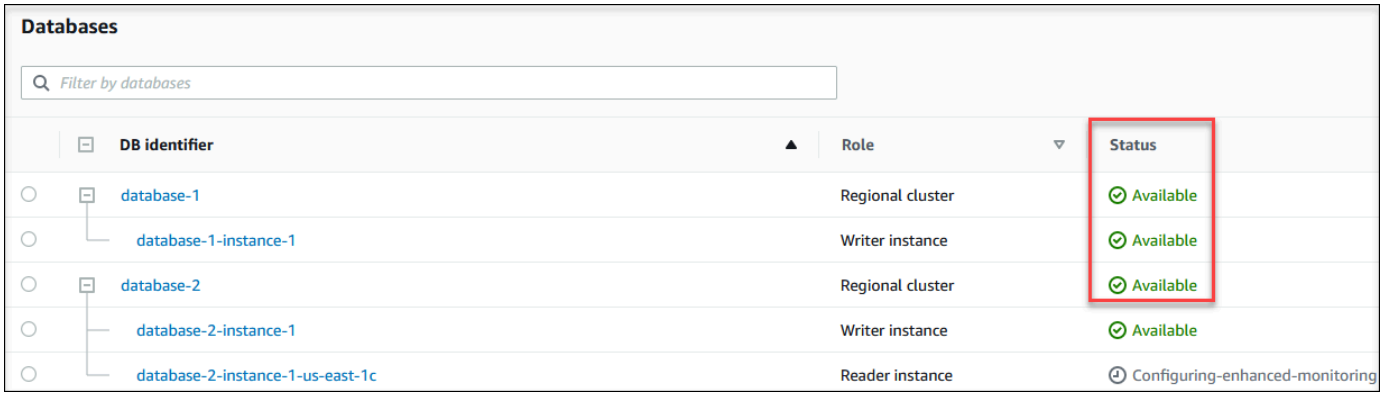
Status instans DB	Ditagih	Deskripsi
Storage-full	Ditagih	Instans DB telah mencapai alokasi kapasitas penyimpanannya. Ini merupakan status kritis. Sebaiknya segera perbaiki masalah ini. Untuk melakukannya, perbesar penyimpanan Anda dengan memodifikasi instans DB. Untuk menghindari situasi ini, setel CloudWatch alarm Amazon untuk memperingatkan Anda saat ruang penyimpanan semakin rendah.
Storage-optimization	Ditagih	Amazon RDS mengoptimalkan penyimpanan instans DB Anda. Instans DB berfungsi sepenuhnya. Proses pengoptim alan penyimpanan biasanya singkat, tetapi terkadang dapat memakan waktu hingga lebih dari 24 jam.
Meningkatkan	Ditagih	Versi mesin basis data sedang ditingkatkan.

Konsol

Untuk melihat status instans DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.

   Halaman Basis Data muncul dengan daftar instans DB. Nilai status setiap instans DB dalam klaster akan ditampilkan.



CLI

Untuk melihat instans DB dan informasi statusnya dengan menggunakan AWS CLI, gunakan describe-db-instancesperintah. Misalnya, AWS CLI perintah berikut mencantumkan semua informasi instance DB.

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

Untuk melihat instans DB tertentu dan statusnya, panggil describe-db-instancesperintah dengan opsi berikut:

- `DBInstanceIdentifier` – Nama instans DB.

```
aws rds describe-db-instances --db-instance-identifier mydbinstance
```

Untuk melihat status semua instans DB, gunakan kueri berikut di AWS CLI.

```
aws rds describe-db-instances --query 'DBInstances[*].
[DBInstanceIdentifier,DBInstanceStatus]' --output table
```

API

Untuk melihat status instans DB menggunakan API Amazon RDS, panggil operasi DescribeDBInstances.

Melihat dan menanggapi rekomendasi Amazon RDS

Amazon Aurora memberikan rekomendasi otomatis untuk sumber daya database, seperti instans DB, cluster DB, dan grup parameter DB. Rekomendasi ini memberikan panduan praktik terbaik dengan menganalisis konfigurasi klaster DB, konfigurasi instans DB, penggunaan, dan data performa.

Amazon RDS Performance Insights memantau metrik tertentu dan secara otomatis membuat ambang batas dengan menganalisis level apa yang dianggap berpotensi bermasalah untuk sumber daya tertentu. Ketika nilai metrik baru melewati ambang batas yang telah ditentukan selama periode waktu tertentu, Performance Insights menghasilkan rekomendasi proaktif. Rekomendasi ini membantu mencegah dampak kinerja database future. Misalnya, rekomendasi “Idle In Transaction” dihasilkan untuk instance PostgreSQL ketika sesi yang terhubung ke database tidak melakukan pekerjaan aktif, tetapi dapat membuat sumber daya database diblokir. Untuk menerima rekomendasi proaktif, Anda harus mengaktifkan Performance Insights dengan periode retensi tingkat berbayar. Untuk informasi tentang mengaktifkan Performance Insights, lihat. Mengaktifkan dan menonaktifkan Wawasan Performa Untuk informasi tentang harga dan retensi data untuk Performance Insights, lihat. Harga dan retensi data untuk Wawasan Performa

DevOpsGuru untuk RDS memantau metrik tertentu untuk mendeteksi kapan perilaku metrik menjadi sangat tidak biasa atau anomali. Anomali ini dilaporkan sebagai wawasan reaktif dengan rekomendasi. Misalnya, DevOps Guru untuk RDS mungkin menyarankan Anda untuk mempertimbangkan peningkatan kapasitas CPU atau menyelidiki peristiwa tunggu yang berkontribusi pada pemuatan DB. DevOpsGuru untuk RDS juga memberikan rekomendasi proaktif berbasis ambang batas. Untuk rekomendasi ini, Anda harus mengaktifkan DevOps Guru untuk RDS. Untuk informasi tentang mengaktifkan DevOps Guru untuk RDS, lihatMengaktifkan DevOps Guru dan menentukan cakupan sumber daya.

Rekomendasi akan berada dalam salah satu status berikut: aktif, diberhentikan, tertunda, atau diselesaikan. Rekomendasi yang diselesaikan tersedia selama 365 hari.

Anda dapat melihat atau mengabaikan rekomendasi. Anda dapat segera menerapkan rekomendasi aktif berbasis konfigurasi, menjadwalkannya di jendela pemeliharaan berikutnya, atau mengabaikannya. Untuk rekomendasi reaktif berbasis proaktif dan pembelajaran mesin berbasis ambang batas, Anda perlu meninjau penyebab masalah yang disarankan dan kemudian melakukan tindakan yang disarankan untuk memperbaiki masalah.

Topik

- Melihat rekomendasi Amazon Aurora

- Menanggapi rekomendasi Amazon Aurora

Melihat rekomendasi Amazon Aurora

Amazon Aurora memberikan rekomendasi untuk sumber daya ketika sumber daya tersebut dibuat atau dimodifikasi.

Rekomendasi berbasis konfigurasi didukung di wilayah berikut:

- AS Timur (Ohio)
- AS Timur (Virginia Utara)
- AS Barat (California Utara)
- AS Barat (Oregon)
- Asia Pasifik (Mumbai)
- Asia Pasifik (Seoul)
- Asia Pasifik (Singapura)
- Asia Pasifik (Sydney)
- Asia Pasifik (Tokyo)
- Kanada (Pusat)
- Eropa (Frankfurt)
- Eropa (Irlandia)
- Eropa (London)
- Eropa (Paris)
- Amerika Selatan (São Paulo)

Anda dapat menemukan contoh rekomendasi berbasis konfigurasi dalam tabel berikut.

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
Sumber Daya Pencadang an otomatis dimatikan	Pencadangan otomatis tidak diaktifkan untuk instans DB Anda. Pencadangan	Aktifkan pencadang an otomatis dengan periode retensi hingga 14 hari.	Ya	Gambaran umum pencadangan dan pemulihan klaster DB Aurora

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
	otomatis direkomen dasikan karena memungkinkan point-in-time pemulihan instans DB Anda.			Mengungkap biaya penyimpan an cadangan Amazon RDS di Blog Database AWS
Diperlukan upgrade versi minor engine	Sumber daya database Anda tidak menjalank an versi mesin DB minor terbaru. Versi minor terbaru berisi perbaikan keamanan terbaru dan peningkat an lainnya.	Tingkatkan ke versi mesin terbaru.	Ya	Memelihara klaster DB Amazon Aurora
Peningkatan Monitoring dimatikan	Sumber daya database Anda tidak mengaktif kan Enhanced Monitoring. Peningkat an Pemantauan menyediakan metrik sistem operasi waktu nyata untuk pemantauan dan pemecahan masalah.	Aktifkan Pemantauan yang Ditingkatkan.	Tidak	Memantau metrik OS dengan Pemantauan yang Disempurnakan

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
Enkripsi penyimpanan dimatikan	Amazon RDS mendukung enkripsi saat istirahat untuk semua mesin database dengan menggunakan kunci yang Anda kelola di AWS Key Management Service (AWSKMS). Pada instans DB aktif dengan enkripsi Amazon RDS, data yang disimpan saat istirahat di penyimpan an dienkripsi, mirip dengan pencadang an otomatis, replika baca, dan snapshot. Jika enkripsi tidak diaktifkan saat membuat cluster Aurora DB, Anda harus mengembal ikan snapshot yang didekripsi ke cluster DB terenkripsi.	Aktifkan enkripsi data saat istirahat untuk cluster DB Anda.	Ya	Keamanan dalam Amazon Aurora

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
Cluster DB dengan semua instance di Availability Zone yang sama	Cluster DB saat ini berada dalam satu Availability Zone. Gunakan beberapa Availability Zone untuk meningkatkan ketersediaan.	Tambahkan instans DB ke beberapa Availability Zone di cluster DB Anda.	Tidak	Ketersediaan yang tinggi untuk Amazon Aurora
Instans DB dalam cluster dengan ukuran instans heterogen	Kami menyarankan Anda menggunak an kelas dan ukuran instans DB yang sama untuk semua instans DB di cluster DB Anda.	Gunakan kelas dan ukuran instance yang sama untuk semua instans DB di cluster DB Anda.	Ya	Replikasi dengan Amazon Aurora
Instans DB dalam cluster dengan kelas instance heterogen	Kami menyarankan Anda menggunak an kelas dan ukuran instans DB yang sama untuk semua instans DB di cluster DB Anda.	Gunakan kelas dan ukuran instance yang sama untuk semua instans DB di cluster DB Anda.	Ya	Replikasi dengan Amazon Aurora
Instans DB dalam cluster dengan kelompok parameter heterogen	Kami merekomen dasikan bahwa semua instance DB di cluster DB menggunakan grup parameter DB yang sama.	Kaitkan instans DB dengan grup parameter DB yang terkait dengan instance penulis di cluster DB Anda.	Tidak	Bekerja dengan grup parameter

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
Cluster Amazon RDS DB memiliki satu instans DB	Tambahkan setidakny a satu instans DB lagi ke cluster DB Anda untuk meningkatkan ketersediaan dan kinerja.	Tambahkan instans DB pembaca ke cluster DB Anda.	Tidak	Ketersediaan yang tinggi untuk Amazon Aurora
Performan ce Insights dimatikan	Performance Insights memantau pemuatan instans DB untuk membantu Anda menganalisis dan menyelesaikan masalah kinerja database. Sebaiknya aktifkan Performance Insights.	Mengaktifkan Wawasan Performa.	Tidak	Memantau muatan DB dengan Wawasan Performa di Amazon Aurora
Sumber daya RDS pembaruan versi utama diperlukan	Database dengan versi utama saat ini untuk mesin DB tidak akan didukung. Kami menyarankan Anda meningkatkan ke versi utama terbaru yang mencakup fungsiona litas dan peningkatan baru.	Tingkatkan ke versi utama terbaru untuk mesin DB.	Ya	Pembaruan Amazon Aurora Membuat deployment blue/green

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
Cluster DB hanya mendukung volume hingga 64 TiB	Cluster DB Anda mendukung volume hingga 64 TiB. Versi mesin terbaru mendukung volume hingga 128 TiB untuk cluster DB Anda. Kami menyarankan Anda meningkatkan versi mesin cluster DB Anda ke versi terbaru untuk mendukung volume hingga 128 TiB.	Tingkatkan versi mesin cluster DB Anda untuk mendukung volume hingga 128 TiB.	Ya	Batas ukuran Amazon Aurora

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
Cluster DB dengan semua instance pembaca di Availability Zone yang sama	Availability Zones (AZ) adalah lokasi yang berbeda satu sama lain untuk memberika n isolasi jika terjadi pemadaman di setiap AWS Wilayah. Sebaiknya Anda mendistribusikan instans utama dan instans pembaca di cluster DB Anda di beberapa AZ untuk meningkatkan ketersediaan cluster DB Anda. Anda dapat membuat klaster Multi-AZ menggunak an AWS Managemen t Console, AWS CLI, atau Amazon RDS API saat membuat cluster. Anda dapat memodifikasi cluster Aurora yang ada ke cluster multi-AZ dengan menambahk an instance pembaca baru dan menentukan AZ yang berbeda.	Cluster DB Anda memiliki semua instance bacaannya di Availability Zone yang sama. Kami menyarankan Anda mendistribusikan instance pembaca di beberapa Availabil ity Zone. Distribus i meningkatkan ketersediaan dan meningkatkan waktu respons dengan mengurangi latensi jaringan antara klien dan database.	Tidak	Ketersediaan yang tinggi untuk Amazon Aurora

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
Parameter memori DB menyimpang dari default	Parameter memori instans DB berbeda secara signifikan dari nilai default. Pengaturan ini dapat memengaruhi kinerja dan menyebabkan kesalahan. Kami menyarankan Anda mengatur ulang parameter memori khusus untuk instans DB ke nilai defaultnya di grup parameter DB.	Setel ulang parameter memori ke nilai defaultnya.	Tidak	Bekerja dengan grup parameter
Parameter cache kueri diaktifkan	Ketika perubahan mengharuskan cache kueri Anda dibersihk an, instans DB Anda akan tampak macet. Cache kueri tidak bermanfaat untuk sebagian besar beban kerja. Cache kueri dihapus dari MySQL versi 8.0. Kami menyarank an Anda mengatur parameter query_cac he_type ke 0.	Tetapkan nilai `query_cac he_type` parameter ke `0` dalam grup parameter DB Anda.	Ya	Bekerja dengan grup parameter

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
`log_outpu t` parameter diatur ke tabel	Ketika `log_output` diatur ke`TABLE`, lebih banyak penyimpanan digunakan daripada ketika `log_outpu t` diatur ke`FILE`. Kami menyarank an Anda mengatur parameter ke`FILE`, untuk menghinda ri mencapai batas ukuran penyimpanan.	Tetapkan nilai `log_output` parameter ke `FILE` dalam grup parameter DB Anda.	Tidak	File log basis data Aurora MySQL
`synchrono us_commit` parameter dimatikan	Ketika `synchrono us_commit` parameter dimatikan , data dapat hilang dalam kerusakan database. Daya tahan database berisiko. Sebaiknya aktifkan parameter `synchrono us_commit` .	Aktifkan `synchrono us_commit` parameter di grup parameter DB Anda.	Ya	Parameter Amazon Aurora PostgreSQL: Replikasi, keamanan, dan logging di Blog Database AWS

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
`track_cou nts` parameter dimatikan	Ketika `track_cou nts` parameter dimatikan, database tidak mengumpul kan statistik aktivitas database. Autovacuu m membutuhkan statistik ini untuk berfungsi dengan benar. Sebaiknya tetapkan parameter `track_counts` ke 1.	Setel `track_cou nts` parameter ke1.	Tidak	Statistik Run-time untuk PostgreSQL
`enable_in dexonlysc an` parameter dimatikan	Perencana kueri atau pengoptimal tidak dapat menggunakan jenis paket pemindaia n khusus indeks saat dimatikan. Kami menyarank an Anda mengatur nilai `enable_in dexonlyscan` parameter ke1.	Tetapkan nilai `enable_in dexonlyscan` parameter ke1.	Tidak	Konfigurasi Metode Perencana untuk PostgreSQL

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
enable_in dexscan para dimatikan	Perencana kueri atau pengoptimal tidak dapat menggunak an jenis rencana pemindaian indeks saat dimatikan. Kami menyarankan Anda menetapka n enable_in dexscan nilainya1.	Tetapkan nilai enable_in dexscan parameter ke1.	Tidak	Konfigurasi Metode Perencana untuk PostgreSQL
innodb_fl ush_log_a t_trx parame dimatikan	Nilai innodb_fl ush_log_at_trx parameter instans DB Anda bukanlah nilai aman. Parameter ini mengontrol persisten si operasi commit ke disk. Sebaiknya tetapkan parameter innodb_fl ush_log_at_trx ke 1.	Tetapkan nilai innodb_fl ush_log_at_trx parameter ke1.	Tidak	Mengonfigurasi seberapa sering buffer log di-flush

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
`innodb_st ats_persi stent` parame dimatikan	Instans DB Anda tidak dikonfigurasi untuk mempertahankan statistik InnoDB ke disk. Ketika statistik tidak disimpan, mereka dihitung ulang setiap kali instance restart dan tabel diakses. Hal ini menyebabkan variasi dalam rencana eksekusi query. Anda dapat memodifikasi nilai parameter global ini di tingkat tabel. Kami menyarank an Anda mengatur nilai `innodb_st ats_persistent` parameter ke`ON`.	Tetapkan nilai `innodb_st ats_persistent` parameter ke`ON`.	Tidak	Bekerja dengan grup parameter

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
`innodb_op en_files` Par rendah	`innodb_op en_files` Parameter mengontrol jumlah file InnoDB dapat membuka pada satu waktu. InnoDB membuka semua log dan file tablespace sistem saat mysqld berjalan. Instans DB Anda memiliki nilai rendah untuk jumlah maksimum file yang dapat dibuka InnoDB pada satu waktu. Sebaiknya tetapkan parameter `innodb_op en_files` ke nilai minimum 65.	Atur `innodb_op en_files` parameter ke nilai minimum65.	Ya	InnoDB membuka file untuk MySQL

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
`max_user_ connectio ns` Parameter rendah	Instans DB Anda memiliki nilai rendah untuk jumlah maksimum koneksi simultan untuk setiap akun basis data. Kami merekomen dasikan pengatura n `max_user_ connections` parameter ke angka yang lebih besar dari5.	Tingkatkan nilai `max_user_ connections` parameter ke angka yang lebih besar dari5.	Ya	Menetapkan Batas Sumber Daya Akun untuk MySQL
Baca Replika terbuka dalam mode yang dapat ditulis	Instans DB Anda memiliki replika baca dalam mode yang dapat ditulis, yang memungkinkan pembaruan dari klien. Kami menyarank an Anda mengatur `read_only` parameter `TrueIfReplica` agar replika baca tidak dalam mode yang dapat ditulis.	Tetapkan nilai `read_only` parameter `keTrueIfReplica .`	Tidak	Bekerja dengan grup parameter

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
`innodb_de fault_row _format` peng n parameter tidak aman	Instans DB Anda mengalami masalah yang diketahui: Tabel yang dibuat dalam versi MySQL yang lebih rendah dari 8.0.26 dengan `row_format` set `COMPACT` ke `REDUNDANT` atau akan tidak dapat diakses dan tidak dapat dipulihkan ketika indeks melebihi 767 byte. Kami menyarank an Anda mengatur nilai `innodb_de fault_row _format` parameter ke`DYNAMIC`.	Tetapkan nilai `innodb_de fault_row _format` parameter ke`DYNAMIC`.	Tidak	Perubahan MySQL 8.0.26

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
`general_l ogging` param dihidupkan	Pencatatan umum diaktifkan untuk instans DB Anda. Pengaturan ini berguna saat memecahkan masalah database. Namun, menyalaka n logging umum meningkatkan jumlah operasi I/O dan ruang penyimpanan yang dialokasikan, yang dapat mengakibatkan pertengkaran dan penurunan kinerja. Periksa persyarat an Anda untuk penggunaan logging umum. Kami menyarankan Anda mengatur nilai `general_logging` parameter ke`0`.	Periksa persyarat an Anda untuk penggunaan logging umum. Jika tidak wajib, kami sarankan Anda untuk mengatur nilai `general_l ogging` parameter ke`0`.	Tidak	Ikhtisar log basis data Aurora MySQL

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
Cluster DB kurang disediakan untuk beban kerja baca	Kami menyarankan Anda menambahkan instans DB pembaca ke cluster DB Anda dengan kelas dan ukuran instans yang sama dengan instance DB penulis di cluster. Konfigurasi saat ini memiliki satu instans DB dengan beban database yang terus menerus tinggi yang sebagian besar disebabka n oleh operasi baca. Distribusikan operasi ini dengan menambahkan instans DB lain ke cluster dan mengarahkan beban kerja baca ke titik akhir read-only cluster DB.	Tambahkan instance DB pembaca ke cluster.	Tidak	Menambahkan Replika Aurora ke klaster DB Mengelola performa dan penskalaan untuk klaster DB Aurora Penetapan Harga

Tipe	Deskripsi	Rekomendasi	Diperlu n downtir	Informasi tambahan
Instans RDS kurang disediaka n untuk kapasitas sistem	Kami menyarank an Anda menyetel kueri Anda untuk menggunakan memori yang lebih rendah atau menggunak an jenis instans DB dengan memori yang dialokasikan lebih tinggi. Ketika instance kehabisan memori, maka kinerja database terpengar uh.	Upsize kelas instance	Ya	Menskalakan Instans Amazon RDS Anda Secara Vertikal dan Horizontal di Blog Database AWS Jenis instans Amazon RDS Penetapan Harga

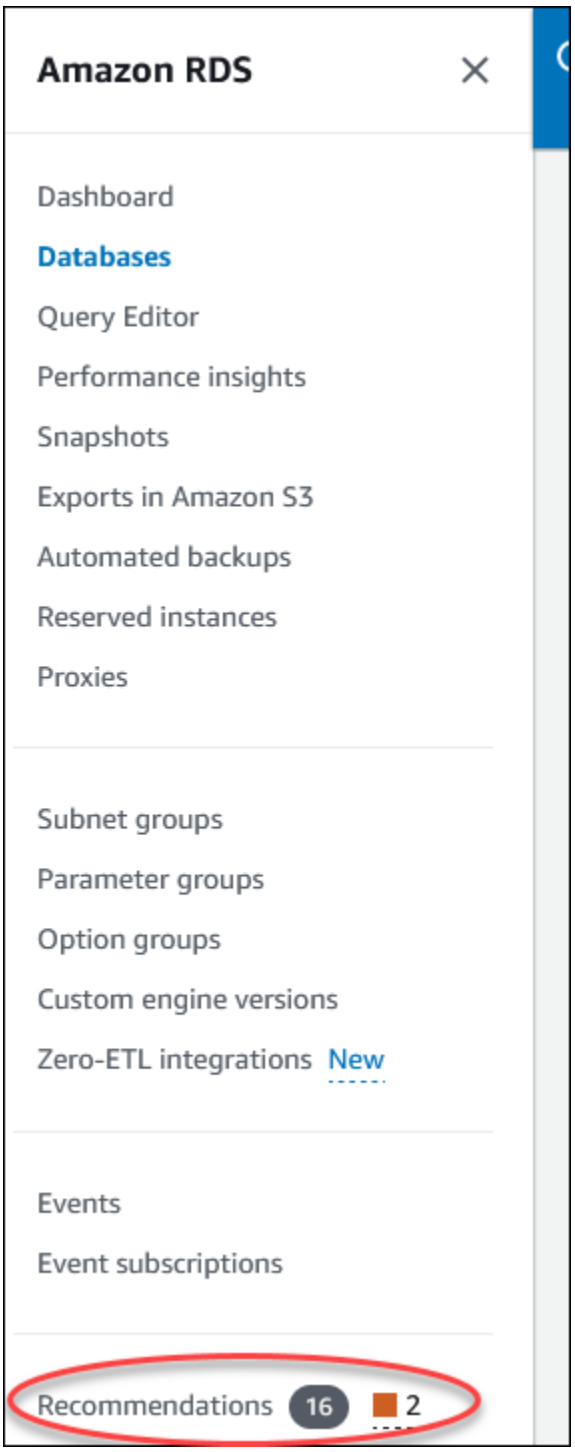
Menggunakan konsol Amazon RDS, Anda dapat melihat rekomendasi Aurora untuk sumber daya database Anda. Untuk cluster DB, rekomendasi muncul untuk cluster DB dan instance-nya.

Konsol

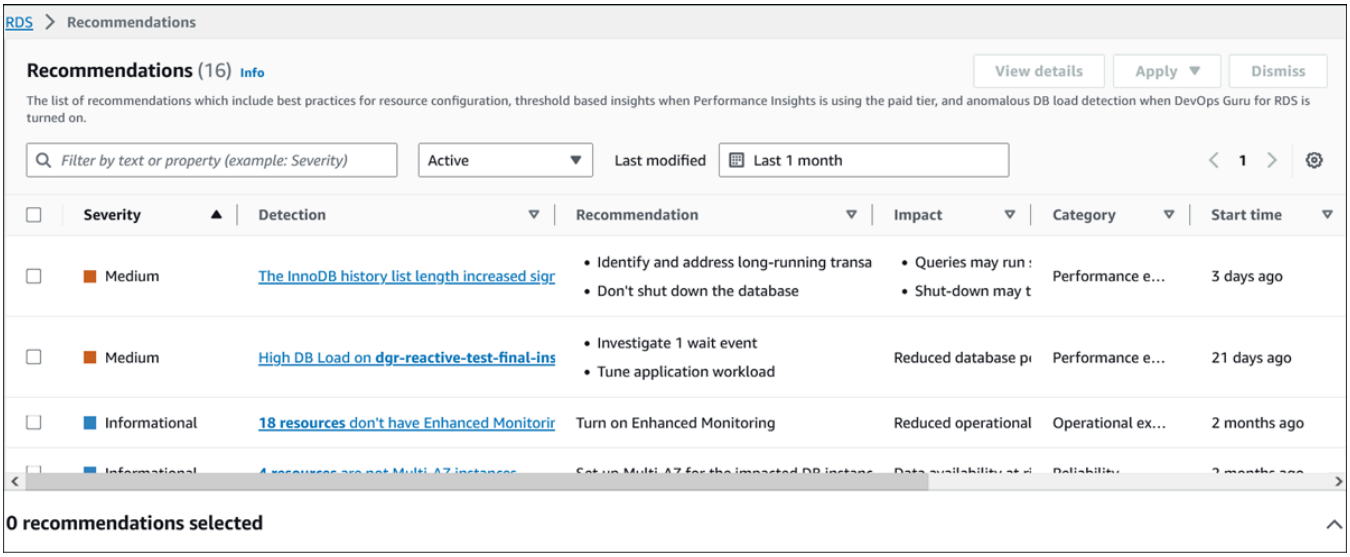
Untuk melihat rekomendasi Amazon Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, lakukan salah satu hal berikut:

   - Pilih Rekomendasi. Jumlah rekomendasi aktif untuk sumber daya Anda dan jumlah rekomendasi dengan tingkat keparahan tertinggi tersedia di sebelah Rekomendasi. Untuk menemukan jumlah rekomendasi aktif untuk setiap tingkat keparahan, pilih angka yang menunjukkan tingkat keparahan paling tinggi.

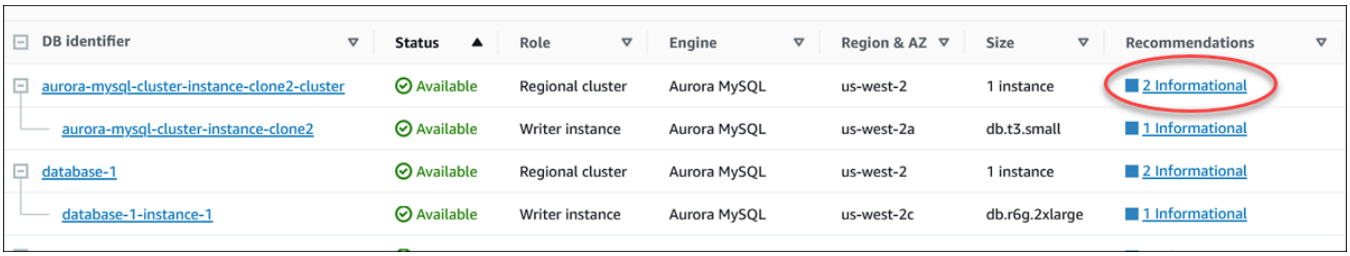


Halaman Rekomendasi menampilkan daftar rekomendasi yang diurutkan berdasarkan tingkat keparahan semua sumber daya di akun Anda.

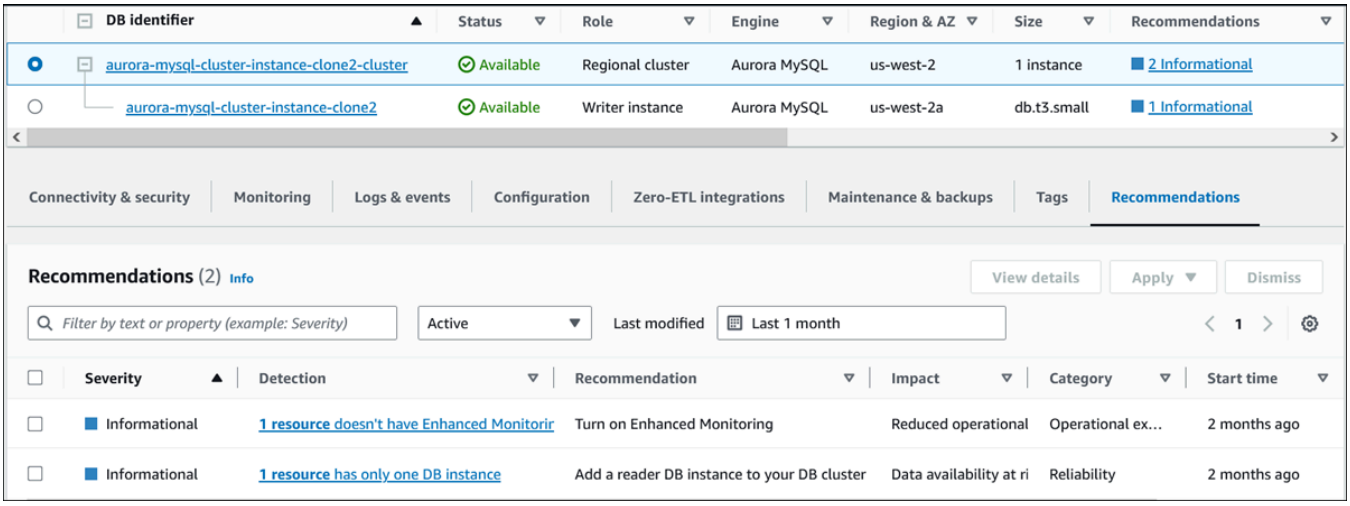


Anda dapat memilih rekomendasi untuk melihat bagian di bagian bawah halaman yang berisi sumber daya yang terpengaruh dan detail tentang bagaimana rekomendasi akan diterapkan.

- Di halaman Database, pilih Rekomendasi untuk sumber daya.



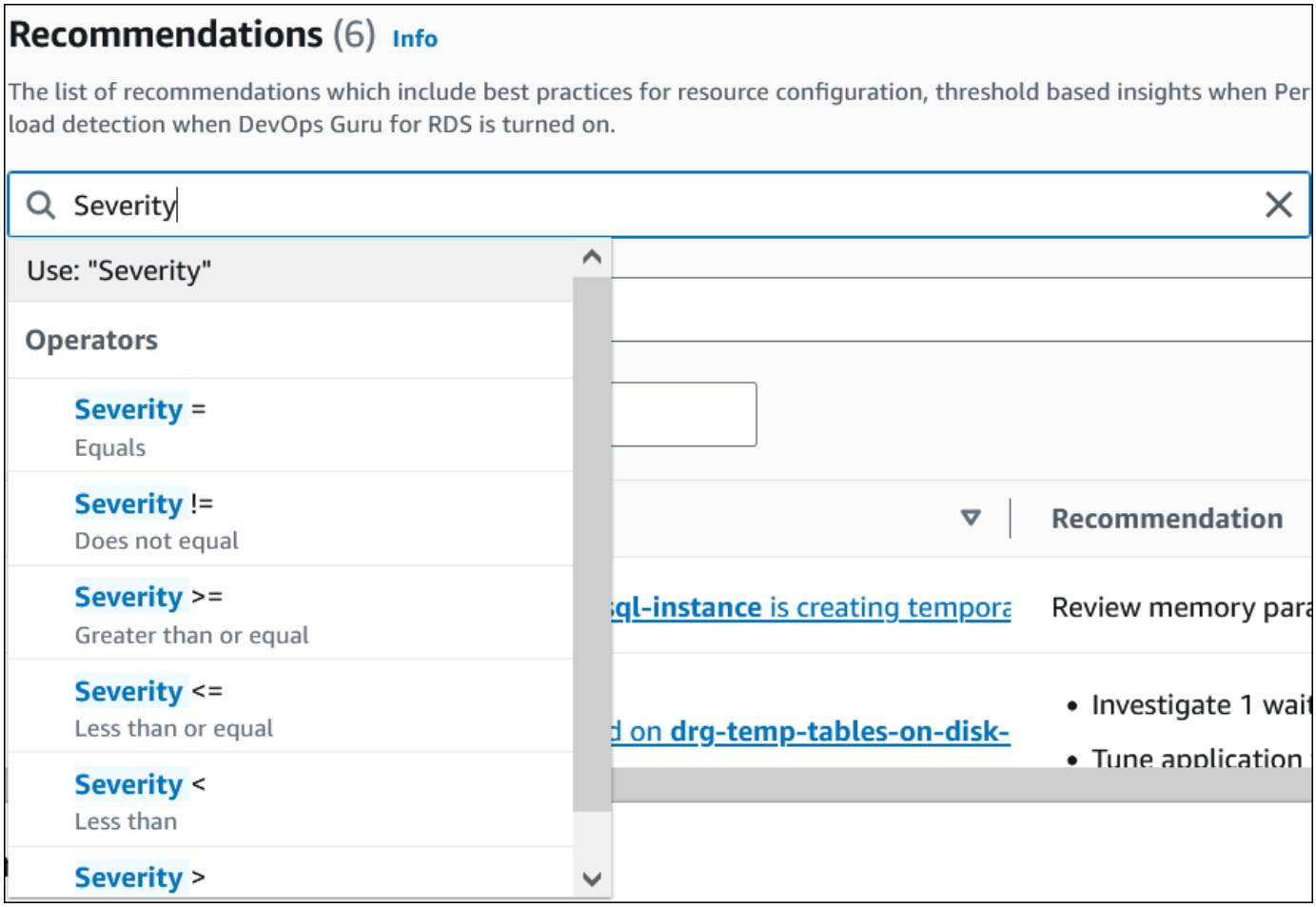
Tab Rekomendasi menampilkan rekomendasi dan detailnya untuk sumber daya yang dipilih.



Rincian berikut tersedia untuk rekomendasi:

- Keparahan — Tingkat implikasi dari masalah ini. Tingkat keparahannya adalah Tinggi, Sedang, Rendah, dan Informasi.
- Deteksi — Jumlah sumber daya yang terpengaruh dan deskripsi singkat tentang masalah ini. Pilih tautan ini untuk melihat rekomendasi dan detail analisis.
- Rekomendasi — Deskripsi singkat tentang tindakan yang disarankan untuk diterapkan.
- Dampak — Deskripsi singkat tentang kemungkinan dampak ketika rekomendasi tidak diterapkan.
- Kategori — Jenis rekomendasi. Kategori tersebut adalah Efisiensi kinerja, Keamanan, Keandalan, Optimalisasi biaya, keunggulan operasional, dan Keberlanjutan.
- Status — Status rekomendasi saat ini. Status yang mungkin adalah Semua, Aktif, Diberhentikan, Diselesaikan, dan Tertunda.
- Waktu mulai — Waktu ketika masalah dimulai. Misalnya, 18 jam yang lalu.
- Terakhir diubah - Waktu ketika rekomendasi terakhir diperbarui oleh sistem karena perubahan Tingkat Keparahan, atau waktu Anda menanggapi rekomendasi. Misalnya, 10 jam yang lalu.
- Waktu akhir - Waktu ketika masalah berakhir. Waktu tidak akan ditampilkan untuk masalah yang berkelanjutan.
- Pengenal sumber daya — Nama satu atau lebih sumber daya.

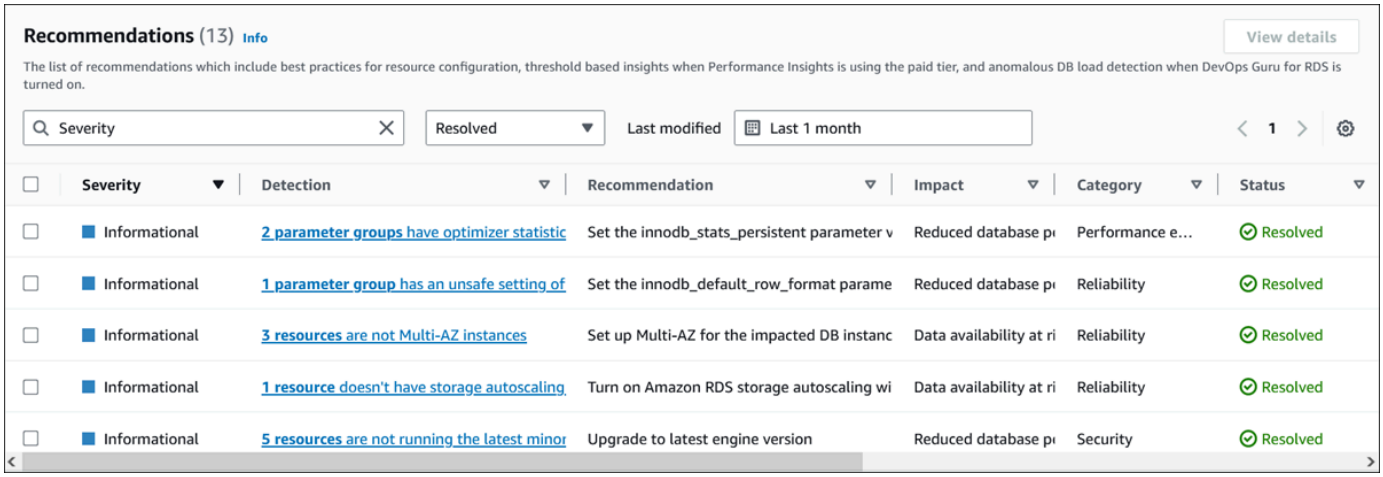
3. (Opsional) Pilih operator Keparahan atau Kategori di bidang untuk memfilter daftar rekomendasi.



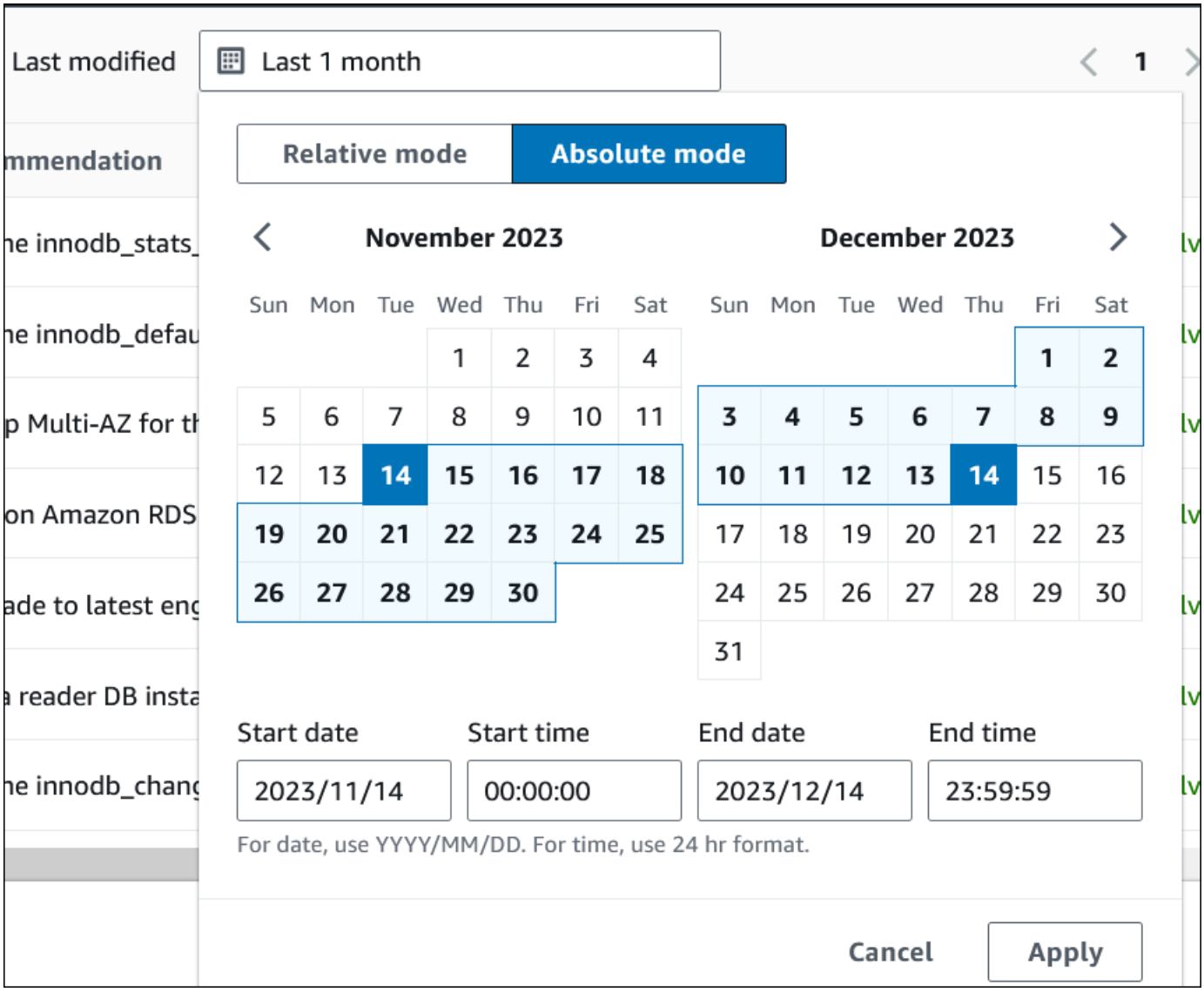
Rekomendasi untuk operasi yang dipilih muncul.

4. (Opsional) Pilih salah satu status rekomendasi berikut:

   - Aktif (default) - Menampilkan rekomendasi saat ini yang dapat Anda terapkan, menjadwalkannya untuk jendela pemeliharaan berikutnya, atau memberhentikan.
   - Semua - Menampilkan semua rekomendasi dengan status saat ini.
   - Diberhentikan — Menunjukkan rekomendasi yang diberhentikan.
   - Terselesaikan - Menunjukkan rekomendasi yang diselesaikan.
   - Tertunda — Menunjukkan rekomendasi yang tindakan rekomendasinya sedang berlangsung atau dijadwalkan untuk jendela pemeliharaan berikutnya.

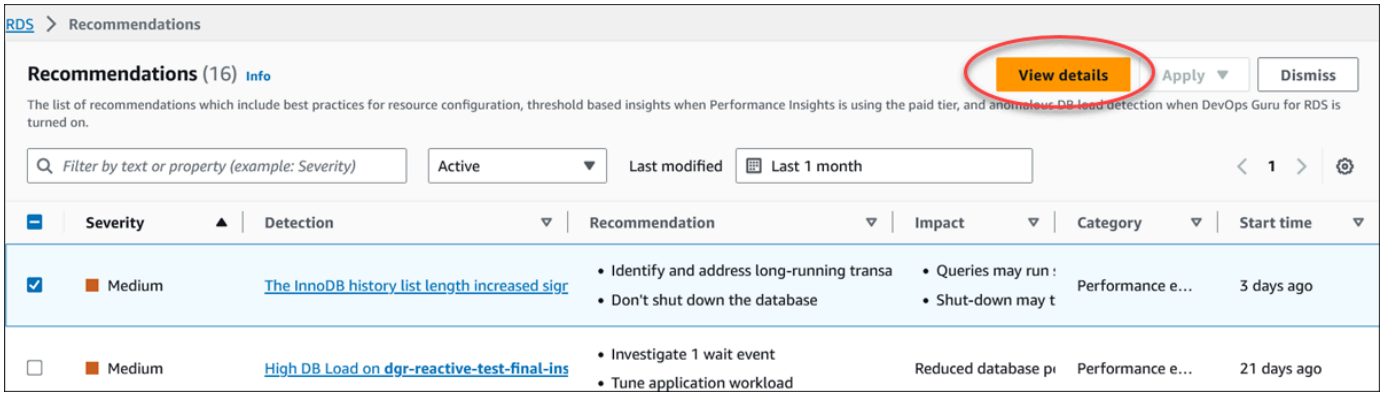


5. (Opsional) Pilih mode Relatif atau Mode absolut di Terakhir diubah untuk mengubah periode waktu untuk menampilkan rekomendasi. Dalam mode Absolute, Anda dapat memilih periode waktu, atau memasukkan waktu di bidang Tanggal mulai dan Tanggal akhir.



Rekomendasi untuk tampilan periode waktu yang ditetapkan.

6. (Opsional) Pilih Preferensi di sebelah kanan untuk menyesuaikan detail yang akan ditampilkan. Anda dapat memilih ukuran halaman, membungkus baris teks, dan mengizinkan atau menyembunyikan kolom.
7. (Opsional) Pilih rekomendasi dan kemudian pilih Lihat detail.



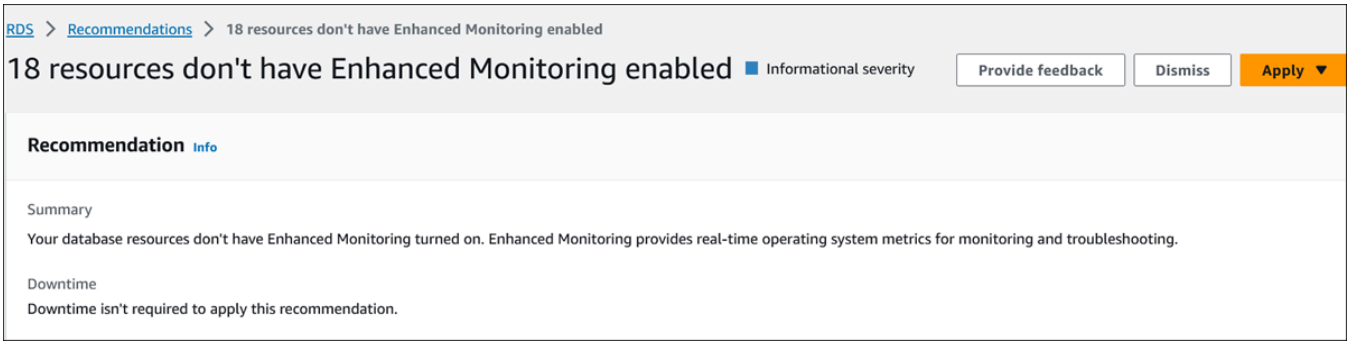
Halaman detail rekomendasi muncul. Judul memberikan jumlah total sumber daya dengan masalah yang terdeteksi dan tingkat keparahannya.

Untuk informasi tentang komponen di halaman detail untuk rekomendasi reaktif berbasis anomali, lihat Melihat anomali reaktif di Panduan Pengguna Amazon Guru. DevOps

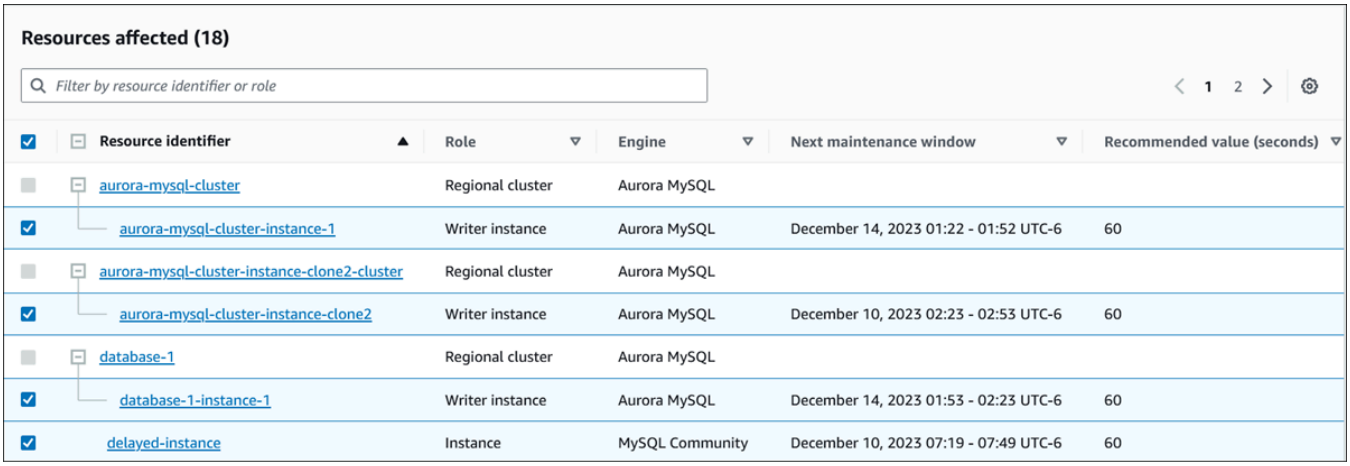
Untuk informasi tentang komponen pada halaman detail untuk rekomendasi proaktif berbasis ambang batas, lihatMelihat rekomendasi proaktif Performance Insights.

Rekomendasi otomatis lainnya menampilkan komponen berikut di halaman detail rekomendasi:

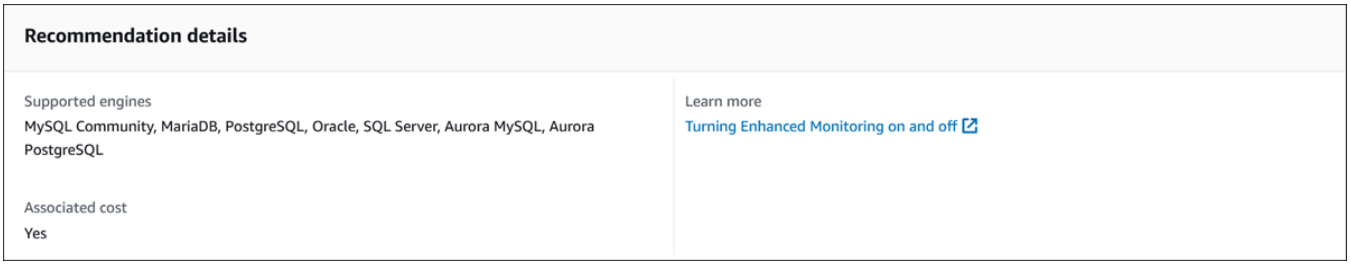
- Rekomendasi — Ringkasan rekomendasi dan apakah downtime diperlukan untuk menerapkan rekomendasi.



- Sumber daya yang terpengaruh — Detail sumber daya yang terpengaruh.



- Detail rekomendasi — Informasi mesin yang didukung, biaya terkait yang diperlukan untuk menerapkan rekomendasi, dan tautan dokumentasi untuk mempelajari lebih lanjut.



CLI

Untuk melihat rekomendasi Amazon RDS dari instans DB atau kluster DB, gunakan perintah berikut di. AWS CLI

```
aws rds describe-db-recommendations
```

API RDS

Untuk melihat rekomendasi Amazon RDS menggunakan Amazon RDS API, gunakan operasi DescribedBrecommendations.

Menanggapi rekomendasi Amazon Aurora

Dari daftar rekomendasi Aurora, Anda dapat:

- Terapkan rekomendasi berbasis konfigurasi segera atau tunda hingga jendela pemeliharaan berikutnya.
- Singkirkan satu atau lebih rekomendasi.

- Pindahkan satu atau lebih rekomendasi yang diberhentikan ke rekomendasi aktif.

Menerapkan Amazon Aurora

Menggunakan konsol Amazon RDS, pilih rekomendasi berbasis konfigurasi atau sumber daya yang terpengaruh di halaman detail, dan segera terapkan rekomendasi atau jadwalkan untuk jendela pemeliharaan berikutnya. Sumber daya mungkin perlu dimulai ulang agar perubahan diterapkan. Untuk beberapa rekomendasi grup parameter DB, Anda mungkin perlu memulai ulang sumber daya.

Rekomendasi reaktif berbasis proaktif atau anomali berbasis ambang batas tidak akan memiliki opsi penerapan dan mungkin memerlukan tinjauan tambahan.

Konsol

Untuk menerapkan rekomendasi berbasis konfigurasi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, lakukan salah satu hal berikut:

   - Pilih Rekomendasi.

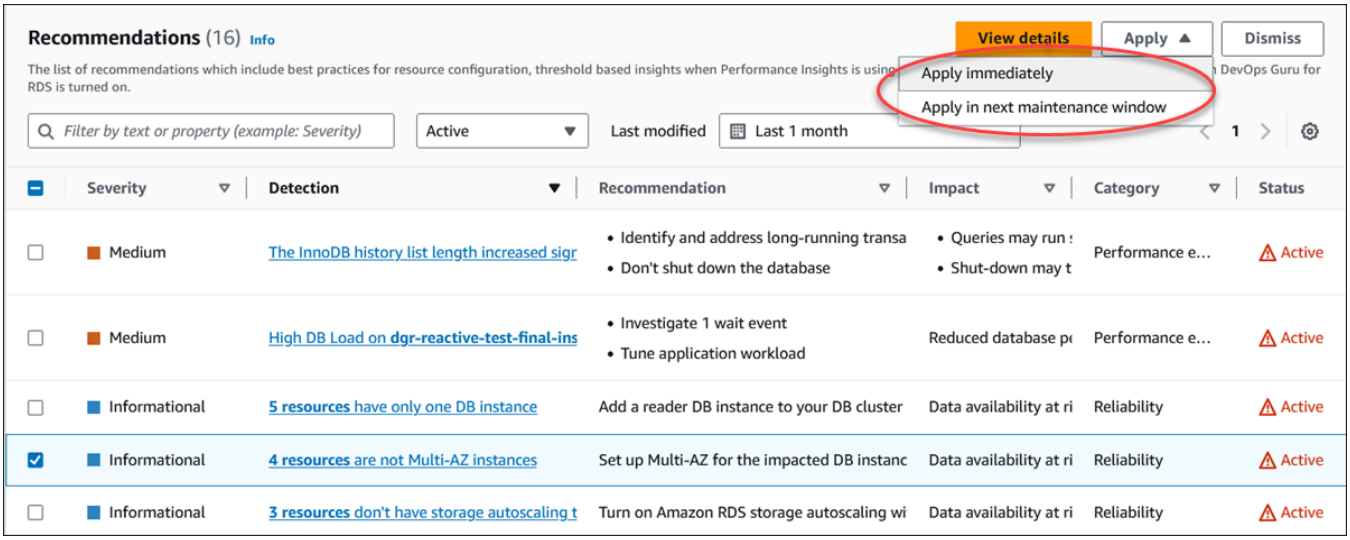
     Halaman Rekomendasi muncul dengan daftar semua rekomendasi.
   - Pilih Database dan kemudian pilih Rekomendasi untuk sumber daya di halaman database.

     Detailnya muncul di tab Rekomendasi untuk rekomendasi yang dipilih.
   - Pilih Deteksi untuk rekomendasi aktif di halaman Rekomendasi atau tab Rekomendasi di halaman Database.

     Halaman detail rekomendasi muncul.
3. Pilih rekomendasi, atau satu atau beberapa sumber daya yang terpengaruh di halaman detail rekomendasi, dan lakukan salah satu hal berikut:

   - Pilih Terapkan dan kemudian pilih Terapkan segera untuk segera menerapkan rekomendasi.
   - Pilih Terapkan dan kemudian pilih Terapkan di jendela pemeliharaan berikutnya untuk menjadwalkan di jendela pemeliharaan berikutnya.

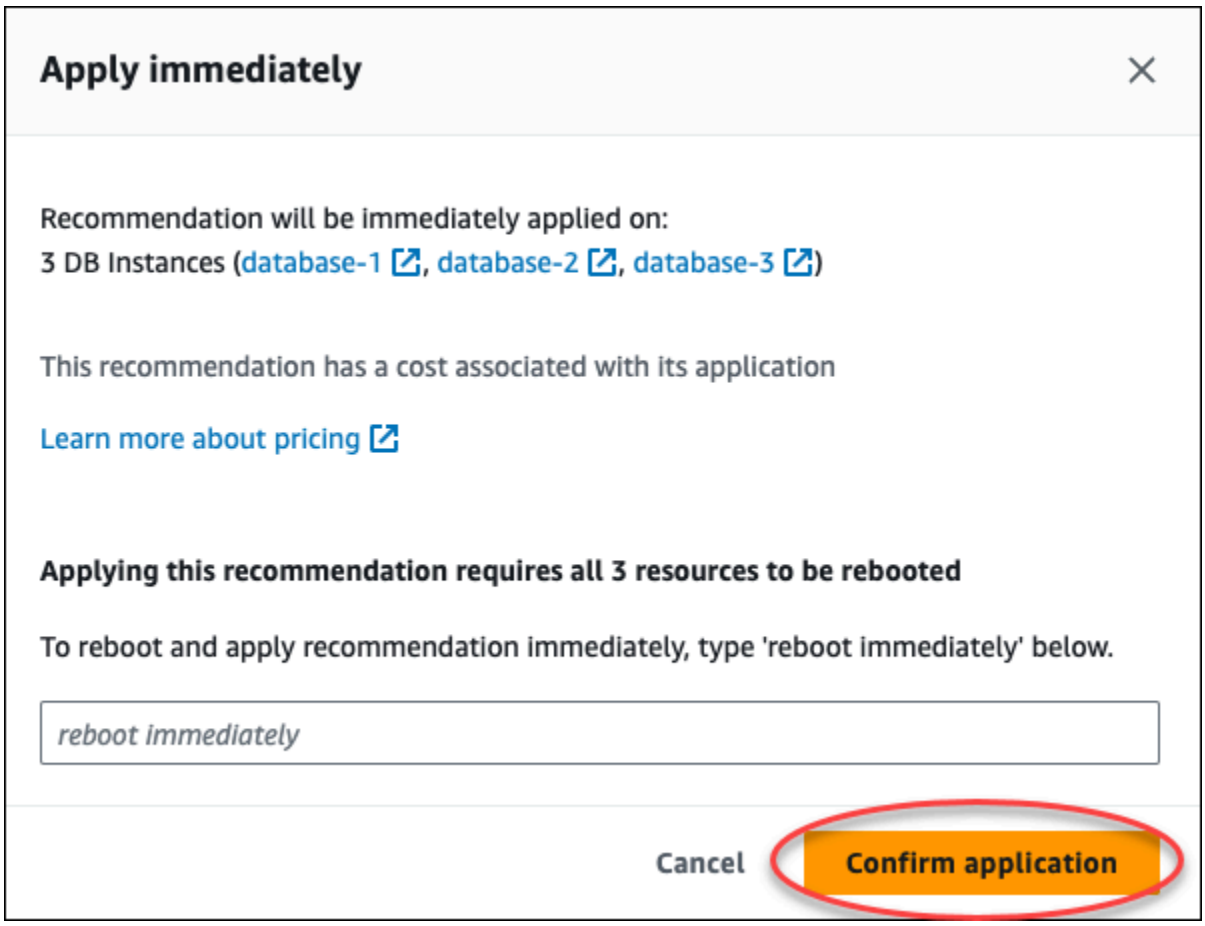
     Status rekomendasi yang dipilih diperbarui ke pending hingga jendela pemeliharaan berikutnya.



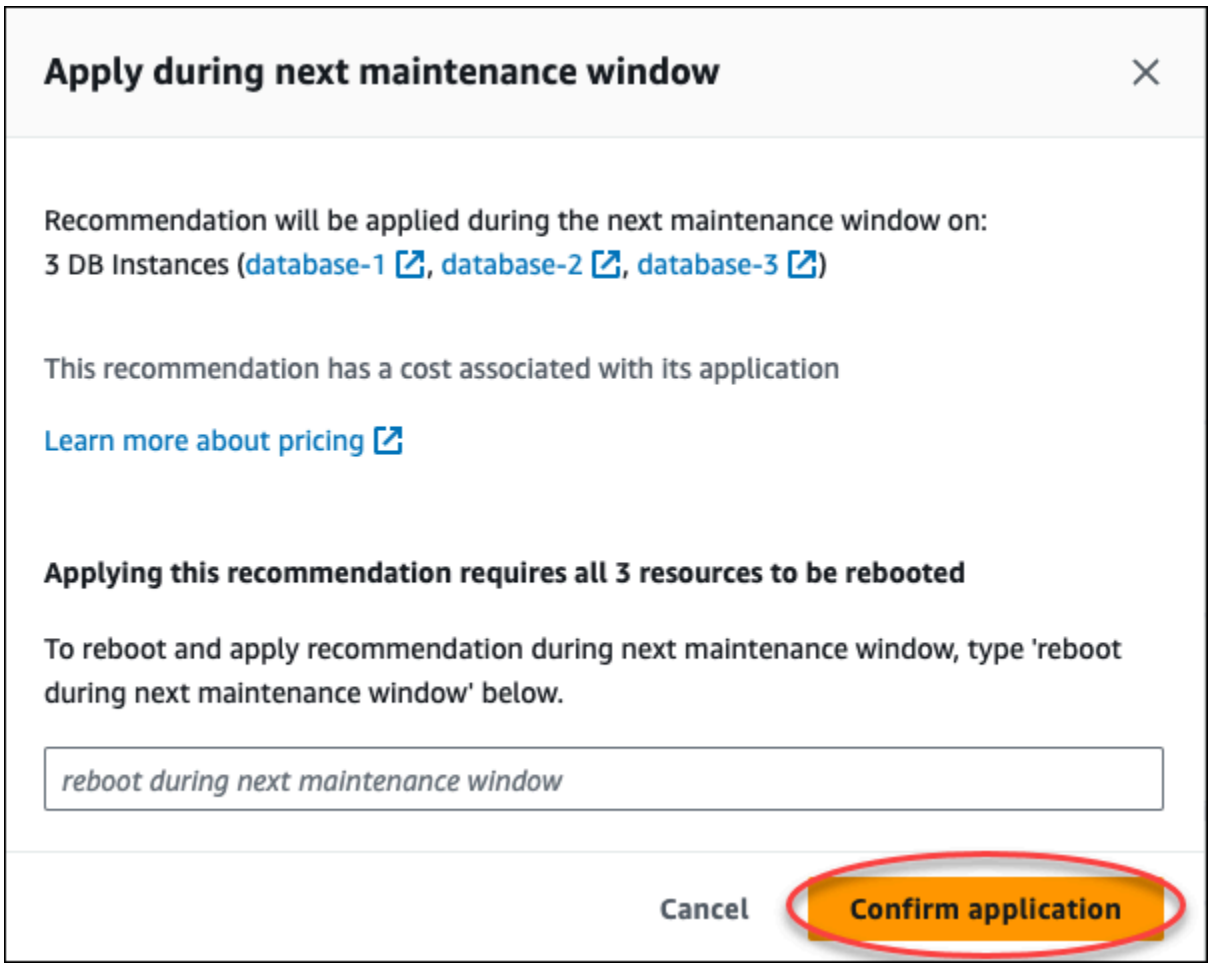
Jendela konfirmasi muncul.

4. Pilih Konfirmasi aplikasi untuk menerapkan rekomendasi. Jendela ini mengonfirmasi apakah sumber daya memerlukan restart otomatis atau manual agar perubahan diterapkan.

   Contoh berikut menunjukkan jendela konfirmasi untuk segera menerapkan rekomendasi.

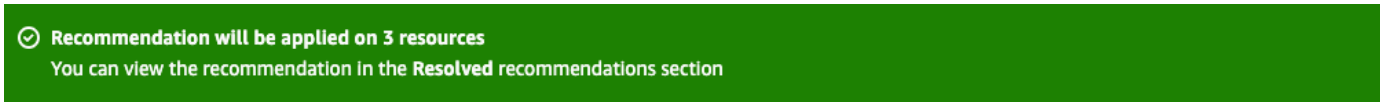


Contoh berikut menunjukkan jendela konfirmasi untuk menjadwalkan penerapan rekomendasi di jendela pemeliharaan berikutnya.



Spanduk menampilkan pesan ketika rekomendasi yang diterapkan berhasil atau gagal.

Contoh berikut menunjukkan spanduk dengan pesan yang berhasil.



Contoh berikut menunjukkan spanduk dengan pesan kegagalan.

API RDS

Untuk menerapkan rekomendasi Aurora berbasis konfigurasi menggunakan Amazon RDS API

1. Gunakan operasi DescribedBrecommendations. `RecommendedActions`Dalam output dapat memiliki satu atau lebih tindakan yang direkomendasikan.
2. Gunakan RecommendedActionobjek untuk setiap tindakan yang direkomendasikan dari langkah 1. Outputnya berisi `Operation` dan`Parameters`.

   Contoh berikut menunjukkan output dengan satu tindakan yang direkomendasikan.

```
   "RecommendedActions": [
       {
           "ActionId": "0b19ed15-840f-463c-a200-b10af1b552e3",
           "Title": "Turn on auto backup", // localized
           "Description": "Turn on auto backup for my-mysql-instance-1", //
localized
           "Operation": "ModifyDbInstance",
           "Parameters": [
               {
                   "Key": "DbInstanceIdentifier",
                   "Value": "my-mysql-instance-1"
               },
               {
                   "Key": "BackupRetentionPeriod",
                   "Value": "7"
               }
           ],
           "ApplyModes": ["immediately", "next-maintenance-window"],
           "Status": "applied"
       },
       ... // several others
   ],
```

3. Gunakan `operation` untuk setiap tindakan yang direkomendasikan dari output pada langkah 2 dan masukkan `Parameters` nilainya.
4. Setelah operasi di langkah 2 berhasil, gunakan operasi ModifydBreCommendation untuk memodifikasi status rekomendasi.

Mengabaikan rekomendasi Aurora

Anda dapat mengabaikan satu atau lebih rekomendasi.

Konsol

Untuk mengabaikan satu atau lebih rekomendasi

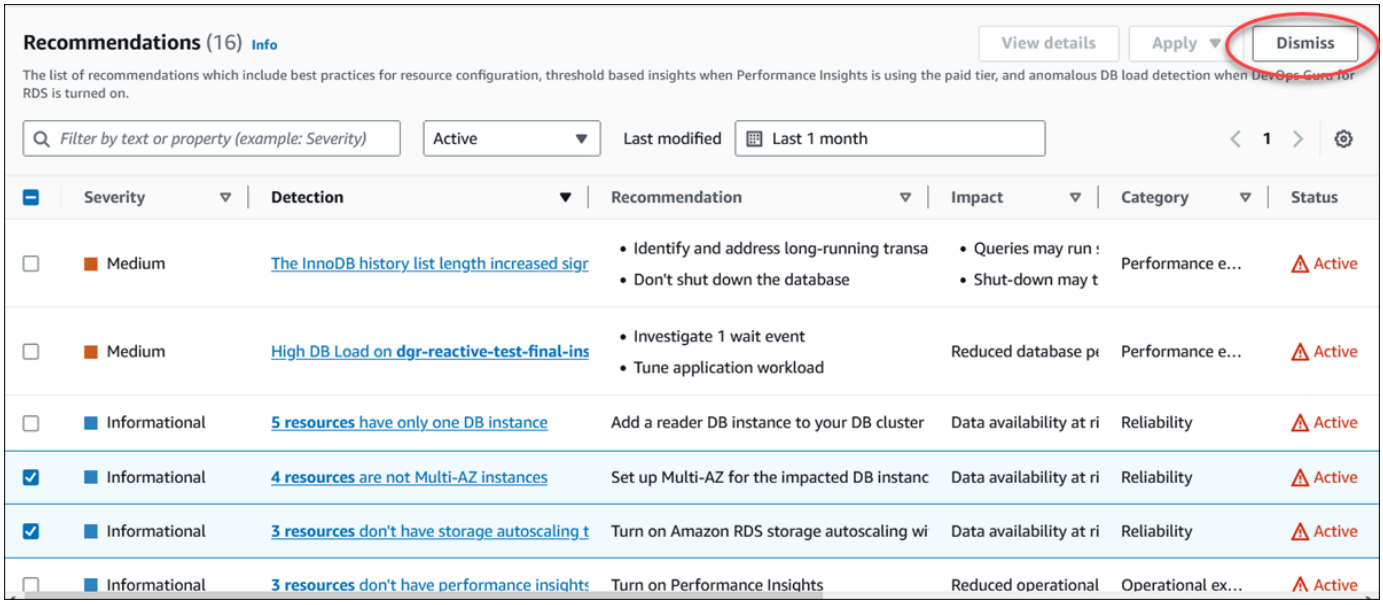
1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, lakukan salah satu hal berikut:
   - Pilih Rekomendasi.

     Halaman Rekomendasi muncul dengan daftar semua rekomendasi.
   - Pilih Database dan kemudian pilih Rekomendasi untuk sumber daya di halaman database.

     Detailnya muncul di tab Rekomendasi untuk rekomendasi yang dipilih.
   - Pilih Deteksi untuk rekomendasi aktif di halaman Rekomendasi atau tab Rekomendasi di halaman Database.

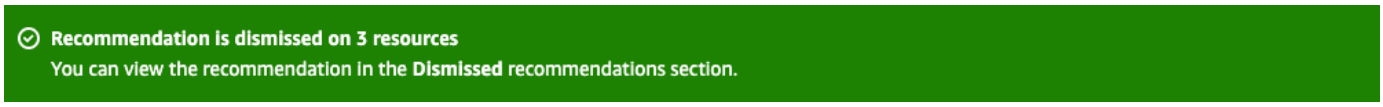
     Halaman detail rekomendasi menampilkan daftar sumber daya yang terpengaruh.
3. Pilih satu atau beberapa rekomendasi, atau satu atau beberapa sumber daya yang terpengaruh di halaman detail rekomendasi, lalu pilih Singkirkan.

   Contoh berikut menunjukkan halaman Rekomendasi dengan beberapa rekomendasi aktif yang dipilih untuk diberhentikan.



Spanduk menampilkan pesan ketika satu atau beberapa rekomendasi yang dipilih diberhentikan.

Contoh berikut menunjukkan spanduk dengan pesan yang berhasil.



Contoh berikut menunjukkan spanduk dengan pesan kegagalan.

CLI

Untuk mengabaikan rekomendasi AWS CLI

1. Jalankan perintah `aws rds describe-db-recommendations --filters "Name=status,Values=active"`.

   Output menyediakan daftar rekomendasi dalam `active` status.
2. Temukan `recommendationId` rekomendasi yang ingin Anda abaikan dari langkah 1.
3. Jalankan perintah `>aws rds modify-db-recommendation --status dismissed --recommendationId <ID>` dengan `recommendationId` dari langkah 2 untuk mengabaikan rekomendasi.

API RDS

Untuk mengabaikan rekomendasi dan Aurora menggunakan Amazon RDS API, gunakan operasi ModifydBrecommendation.

Memodifikasi rekomendasi Amazon yang diberhentikan menjadi rekomendasi aktif

Anda dapat memindahkan satu atau lebih rekomendasi yang diberhentikan ke rekomendasi aktif.

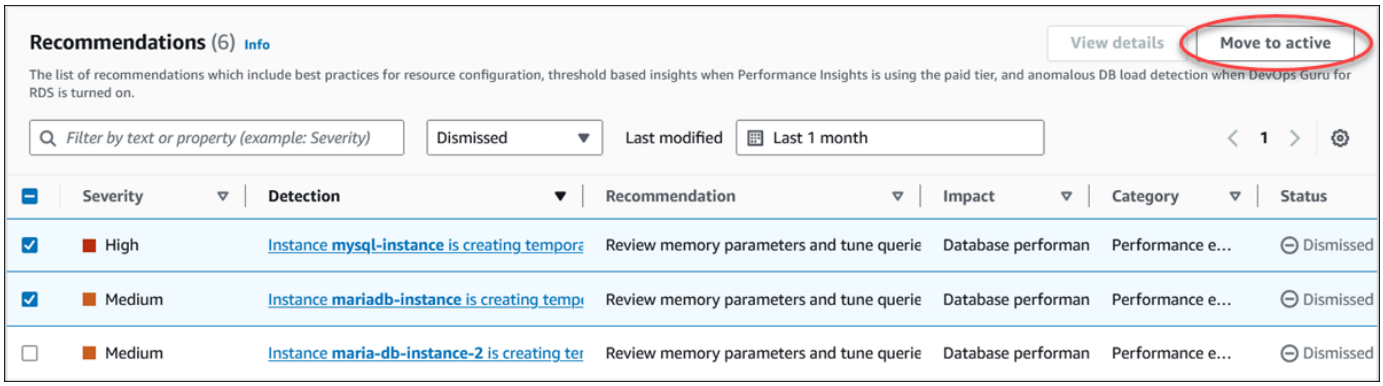
Konsol

Untuk memindahkan satu atau lebih rekomendasi yang diberhentikan ke rekomendasi aktif

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, lakukan salah satu hal berikut:
   - Pilih Rekomendasi.

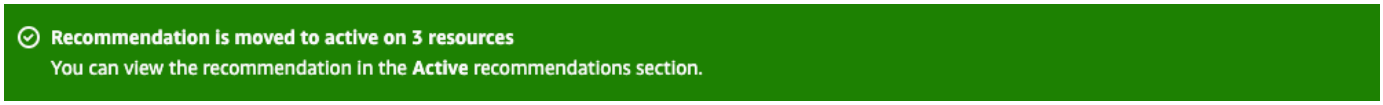
     Halaman Rekomendasi menampilkan daftar rekomendasi yang diurutkan berdasarkan tingkat keparahan semua sumber daya di akun Anda.
   - Pilih Database dan kemudian pilih Rekomendasi untuk sumber daya di halaman database.

     Tab Rekomendasi menampilkan rekomendasi dan detailnya untuk sumber daya yang dipilih.
3. Pilih satu atau beberapa rekomendasi yang diberhentikan dari daftar dan kemudian pilih Pindah ke aktif.



Spanduk menampilkan pesan yang berhasil atau gagal saat memindahkan rekomendasi yang dipilih dari diberhentikan ke status aktif.

Contoh berikut menunjukkan spanduk dengan pesan yang berhasil.



Contoh berikut menunjukkan spanduk dengan pesan kegagalan.

CLI

Untuk mengubah rekomendasi yang diberhentikan menjadi rekomendasi aktif menggunakan AWS CLI

1. Jalankan perintah `aws rds describe-db-recommendations --filters "Name=status,Values=dismissed"`.

   Output menyediakan daftar rekomendasi dalam `dismissed` status.
2. Temukan `recommendationId` rekomendasi yang ingin Anda ubah statusnya dari langkah 1.
3. Jalankan perintah `>aws rds modify-db-recommendation --status active --recommendationId <ID>` dengan `recommendationId` dari langkah 2 untuk mengubah ke rekomendasi aktif.

API RDS

Untuk mengubah rekomendasi yang diberhentikan menjadi rekomendasi aktif menggunakan Amazon RDS API, gunakan operasi ModifydBrecommendation.

Melihat metrik di konsol Amazon RDS

Amazon RDS terintegrasi dengan Amazon CloudWatch untuk menampilkan berbagai metrik klaster DB Aurora di konsol RDS. Beberapa metrik diterapkan di tingkat klaster, sedangkan metrik lainnya diterapkan di tingkat instans. Untuk deskripsi metrik tingkat instans dan tingkat klaster, lihat Referensi metrik untuk Amazon Aurora.

Untuk klater DB Aurora, kategori metrik berikut dipantau:

- CloudWatch – Menampilkan metrik Amazon CloudWatch untuk Aurora yang dapat diakses di konsol RDS. Anda dapat metrik-metrik ini di konsol CloudWatch. Setiap metrik berisi grafik yang menunjukkan metrik yang dipantau selama rentang waktu tertentu. Untuk daftar metrik CloudWatch, lihat CloudWatch Metrik Amazon untuk Amazon Aurora.
- Pemantauan yang disempurnakan – Menampilkan ringkasan metrik sistem operasi saat klaster DB Aurora telah mengaktifkan Pemantauan yang Disempurnakan. RDS mengirimkan metrik dari Pemantauan yang Disempurnakan ke akun Log Amazon CloudWatch Anda. Setiap metrik OS berisi grafik yang menunjukkan metrik yang dipantau selama rentang waktu tertentu. Untuk ikhtisar, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan. Untuk daftar metrik Pemantauan yang Disempurnakan, lihat Metrik OS dalam Pemantauan yang Disempurnakan.
- Daftar Proses OS – Menampilkan detail setiap proses yang berjalan dalam klaster DB Anda.
- Wawasan Performa – Membuka dasbor Wawasan Performa Amazon RDS untuk instans DB di klaster DB Aurora Anda. Wawasan Performa tidak didukung di tingkat klaster. Untuk ikhtisar Wawasan Performa, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora. Untuk daftar metrik Wawasan Performa, lihat CloudWatch Metrik Amazon untuk Performance Insights.

Amazon RDS kini menyediakan tampilan gabungan metrik Wawasan Performa dan CloudWatch di dasbor Wawasan Performa. Untuk menggunakan tampilan ini, Wawasan Performa harus diaktifkan klaster DB Anda. Anda dapat memilih tampilan pemantauan baru di tab Pemantauan atau Wawasan Performa di panel navigasi. Untuk melihat petunjuk cara memilih tampilan ini, lihat Menampilkan metrik gabungan di konsol Amazon RDS.

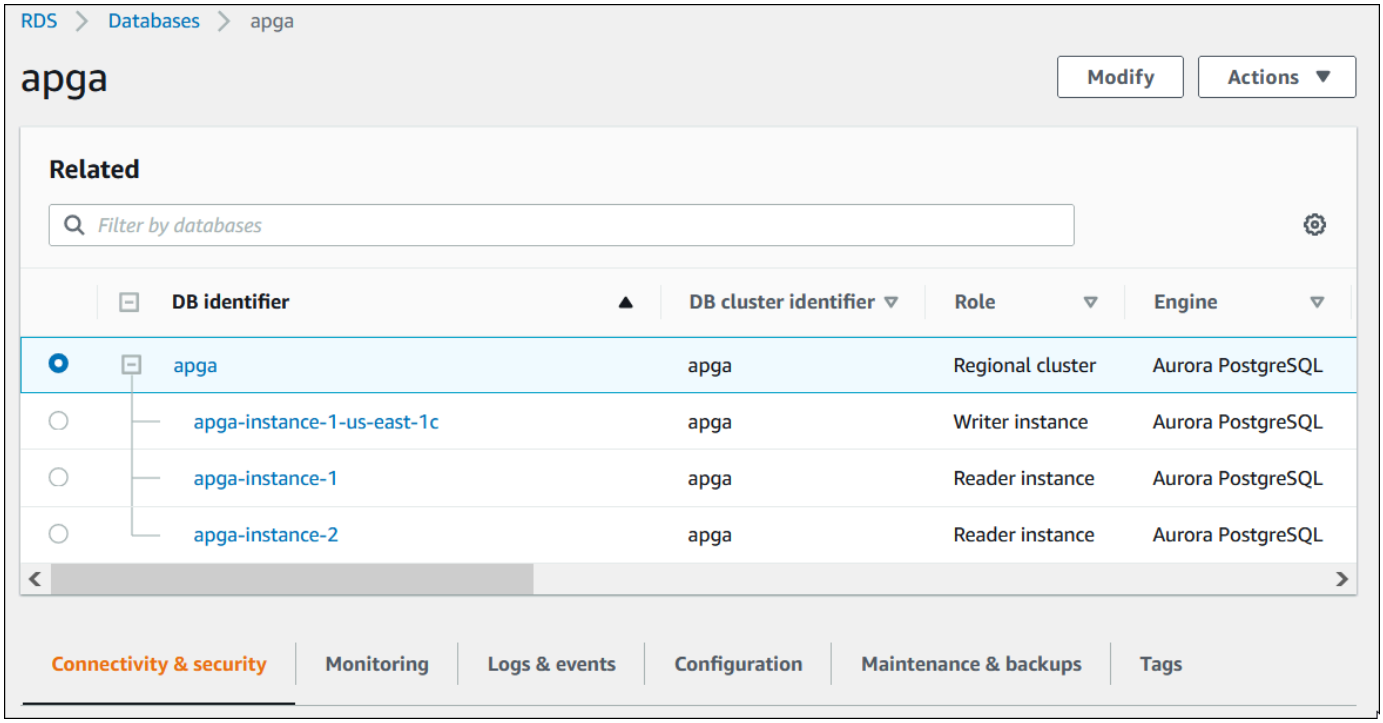
Jika Anda ingin melanjutkan tampilan pemantauan lama, lanjutkan dengan prosedur ini.

> **Note**
>
> Tampilan pemantauan lama akan dihentikan pada tanggal 15 Desember 2023.

Untuk melihat metrik klaster DB Anda di tampilan pemantauan lama:

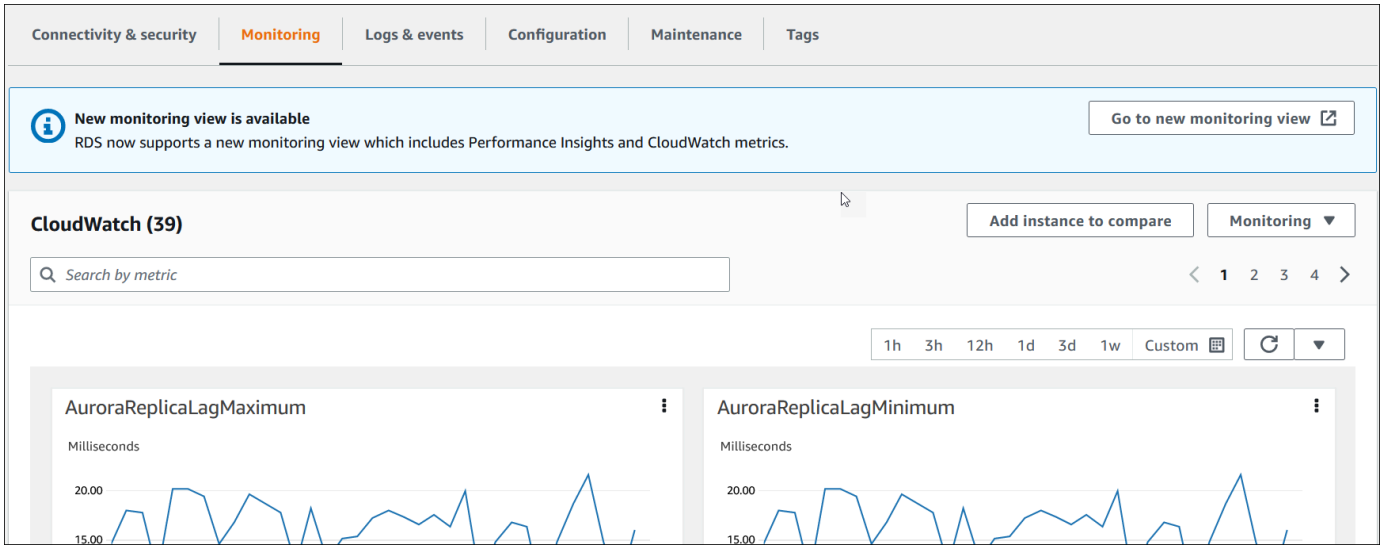
1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih nama klaster DB Aurora yang ingin dipantau.

   Halaman basis data akan muncul. Contoh berikut menunjukkan basis data Amazon Aurora PostgreSQL bernama apga.

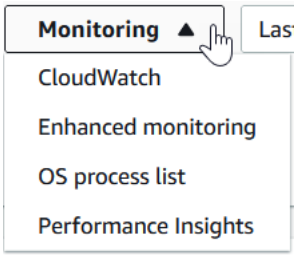


4. Gulir ke bawah dan pilih Pemantauan.

   Bagian pemantauan muncul. Secara default, metrik CloudWatch ditampilkan. Untuk deskripsi metrik ini, lihat CloudWatch Metrik Amazon untuk Amazon Aurora.

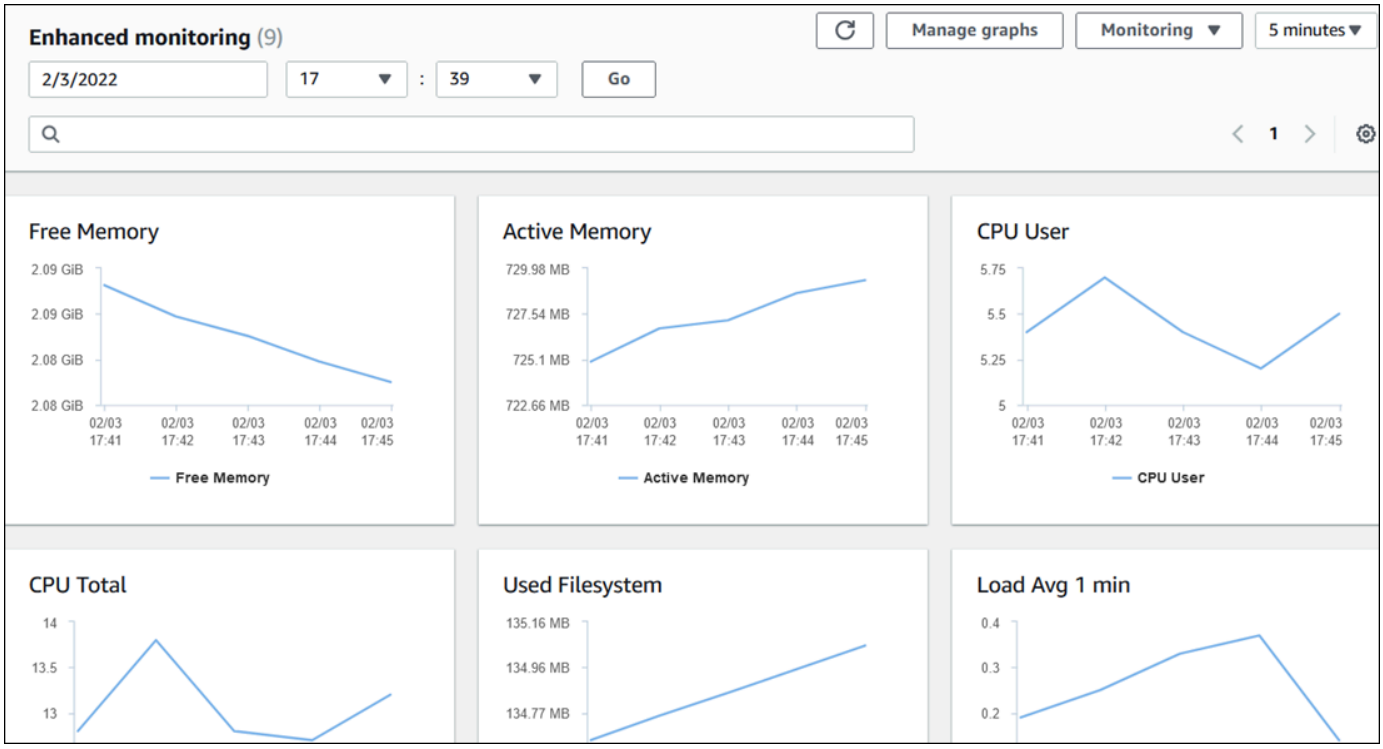


5. Pilih Pemantauan untuk melihat kategori metrik.



6. Pilih kategori metrik yang ingin dilihat.

   Contoh berikut menunjukkan metrik Pemantauan yang Disempurnakan. Untuk deskripsi metrik ini, lihat Metrik OS dalam Pemantauan yang Disempurnakan.



Tip

Untuk memilih rentang waktu metrik yang ditampilkan oleh grafik, Anda dapat menggunakan daftar rentang waktu.

Untuk memunculkan tampilan yang lebih mendetail, Anda dapat memilih grafik mana pun. Anda juga dapat menerapkan filter spesifik metrik pada data.

Menampilkan metrik gabungan di konsol Amazon RDS

Amazon RDS kini menyediakan tampilan metrik Wawasan Performa dan CloudWatch yang terpadu untuk instans DB Anda di dasbor Wawasan Performa. Anda dapat menggunakan dasbor yang telah dikonfigurasi sebelumnya atau membuat dasbor kustom. Dasbor yang telah dikonfigurasi sebelumnya menghadirkan metrik yang paling umum digunakan untuk membantu mendiagnosis masalah kinerja untuk mesin basis data. Atau, Anda dapat membuat dasbor kustom dengan metrik untuk mesin basis data yang memenuhi persyaratan analisis Anda. Kemudian, gunakan dasbor ini untuk semua instans DB dari jenis mesin basis data di akun AWS Anda.

Anda dapat memilih tampilan pemantauan baru di tab Pemantauan atau Wawasan Performa di panel navigasi. Saat membuka halaman Wawasan Performa, Anda akan melihat opsi untuk memilih antara tampilan pemantauan baru dan tampilan lama. Opsi yang Anda pilih disimpan sebagai tampilan default.

Wawasan Performa harus diaktifkan agar klaster DB Anda dapat melihat metrik gabungan di dasbor Wawasan Performa. Untuk informasi selengkapnya tentang cara mengaktifkan Wawasan Performa, lihat Mengaktifkan dan menonaktifkan Wawasan Performa.

> **Note**
>
> Sebaiknya Anda memilih tampilan pemantauan baru. Anda dapat terus menggunakan tampilan pemantauan lama hingga dihentikan pada 15 Desember 2023.

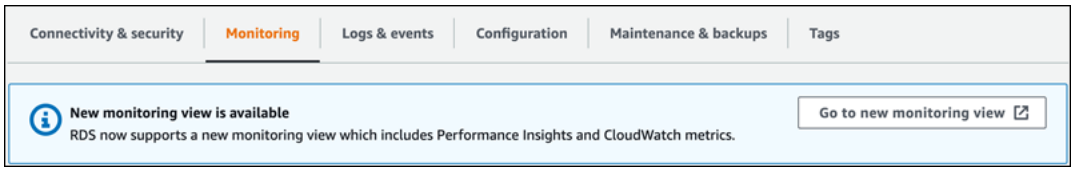
Memilih tampilan pemantauan baru di tab Pemantauan

Untuk memilih tampilan pemantauan baru di tab Pemantauan:

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Basis Data.
3. Pilih klaster DB yang ingin Anda pantau.

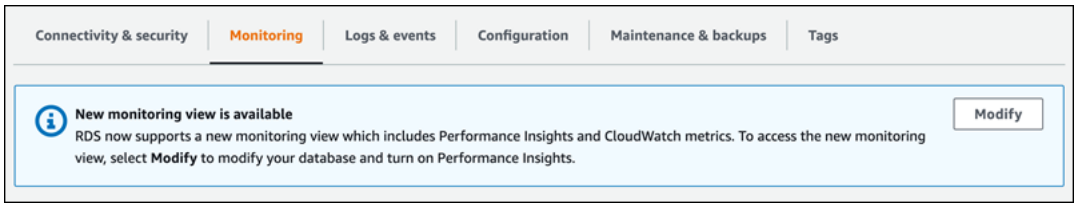
   Halaman basis data akan muncul.
4. Gulir ke bawah dan pilih tab Pemantauan.

Banner akan muncul dengan opsi untuk memilih tampilan pemantauan baru. Contoh berikut menampilkan banner untuk memilih tampilan pemantauan baru.



5. Pilih Masuk ke tampilan pemantauan baru untuk membuka dasbor Wawasan Performa dengan metrik Wawasan Performa dan CloudWatch untuk klaster DB Anda.
6. (Opsional) Jika Wawasan Performa dinonaktifkan untuk instans DB Anda, banner akan muncul dengan opsi untuk memodifikasi instans DB dan mengaktifkan Wawasan Performa.

   Contoh berikut menampilkan banner untuk memodifikasi instans DB di tab Pemantauan.



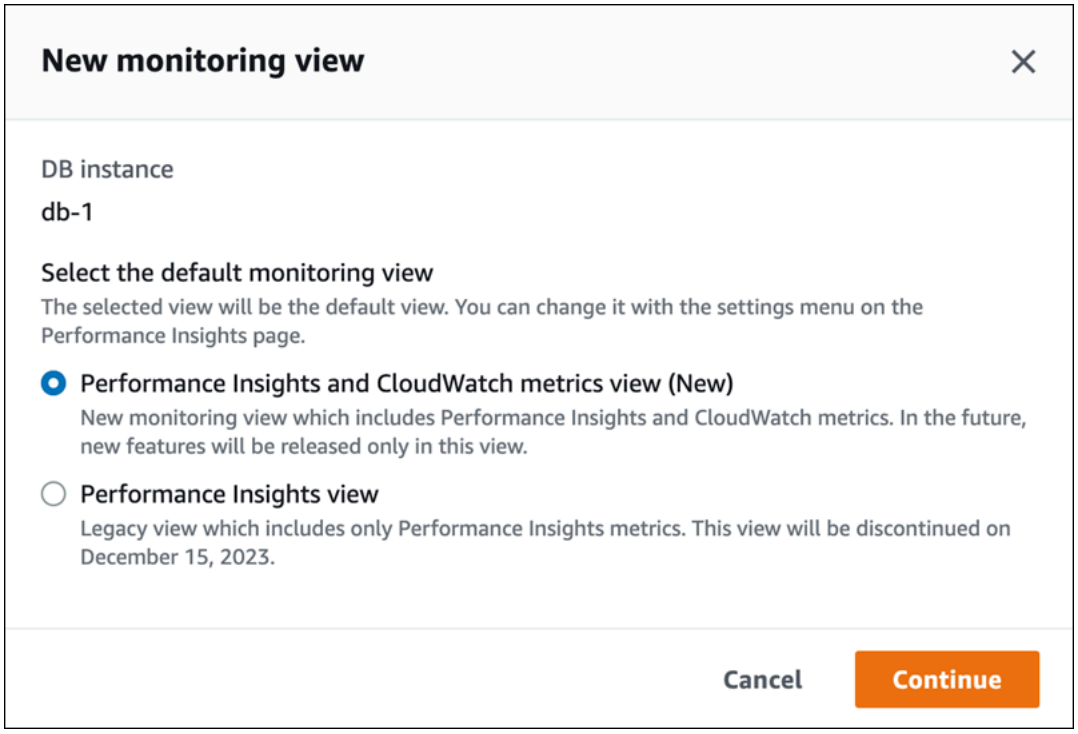
   Pilih Modifikasi untuk memodifikasi instans DB dan aktifkan Wawasan Performa. Untuk informasi selengkapnya tentang cara mengaktifkan Wawasan Performa, lihat Mengaktifkan dan menonaktifkan Wawasan Performa

Memilih tampilan pemantauan baru dengan Wawasan Performa di panel navigasi

Untuk memilih tampilan pemantauan baru dengan Wawasan Performa di panel navigasi:

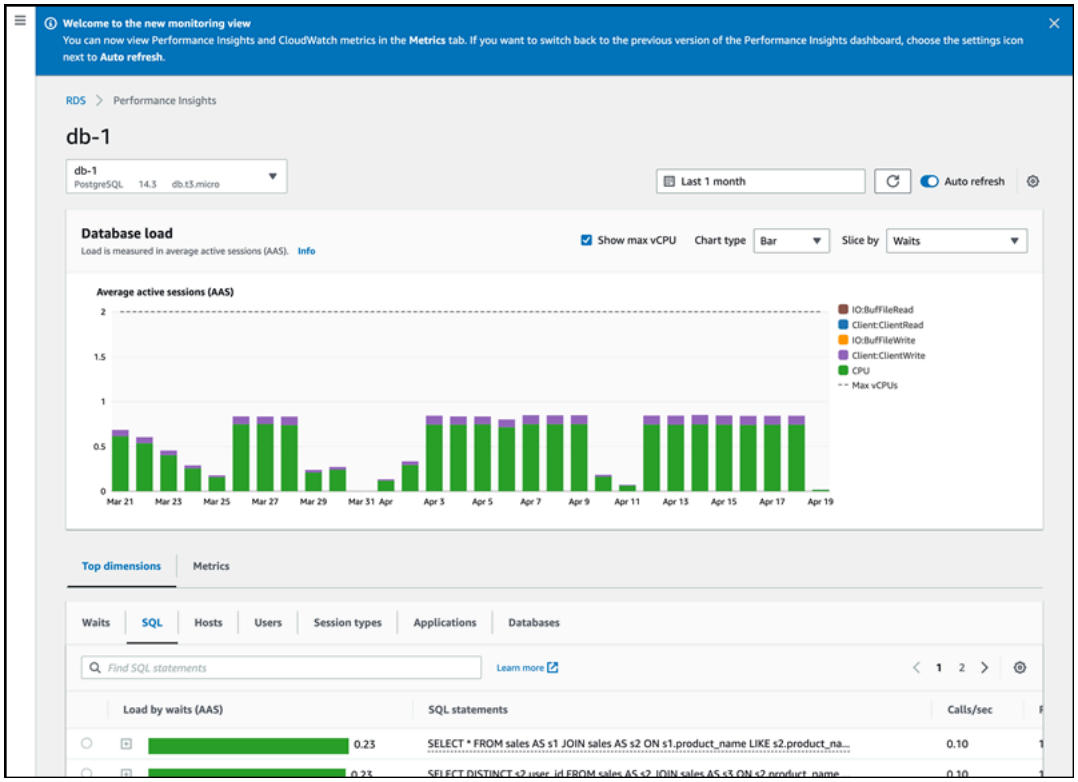
1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB untuk membuka jendela yang memiliki opsi tampilan pemantauan.

   Contoh berikut menampilkan jendela dengan opsi tampilan pemantauan.



4. Pilih opsi Tampilan metrik Wawasan Performa dan CloudWatch (Baru), lalu pilih Lanjutkan.

   Anda dapat melihat dasbor Wawasan Performa yang menampilkan metrik Wawasan Performa dan CloudWatch untuk instans DB Anda. Contoh berikut menampilkan metrik Wawasan Performa dan CloudWatch di dasbor.



Memilih tampilan lama dengan Wawasan Performa di panel navigasi

Anda dapat memilih tampilan pemantauan lama untuk hanya melihat metrik Wawasan Performa untuk instans DB Anda.

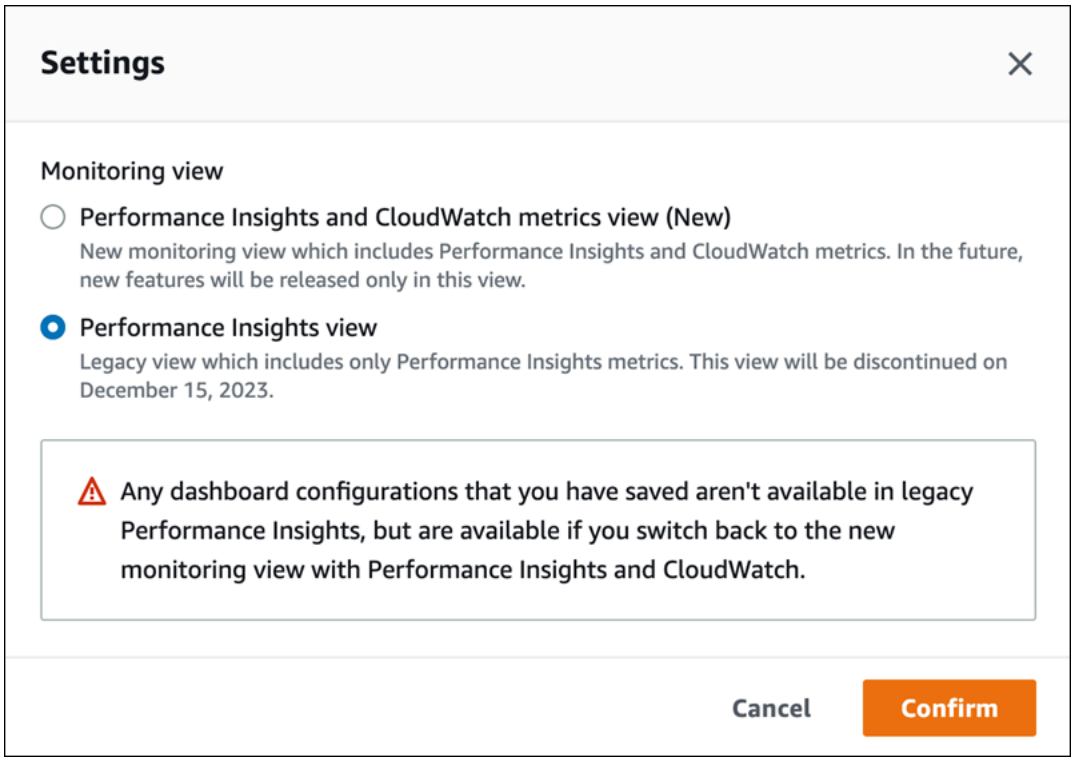
> **Note**
>
> Tampilan ini akan dihentikan pada 15 Desember 2023.

Untuk memilih tampilan pemantauan lama dengan Wawasan Performa di panel navigasi:

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.
4. Pilih ikon pengaturan di dasbor Wawasan Performa.

Sekarang Anda dapat melihat jendela Pengaturan yang menampilkan opsi untuk memilih tampilan Wawasan Performa lama.

Contoh berikut menampilkan jendela dengan opsi tampilan pemantauan lama.



5. Pilih opsi Tampilan Wawasan Performa dan pilih Lanjutkan.

   Pesan peringatan akan muncul. Konfigurasi dasbor apa pun yang Anda simpan tidak akan tersedia dalam tampilan ini.

6. Pilih Konfirmasi untuk melanjutkan ke tampilan Wawasan Performa lama.

   Anda sekarang dapat melihat dasbor Wawasan Performa yang menampilkan metrik Wawasan Performa saja untuk instans DB Anda.

Membuat dasbor kustom dengan Wawasan Performa di panel navigasi

Di tampilan pemantauan baru, Anda dapat membuat dasbor kustom dengan metrik yang Anda butuhkan untuk memenuhi persyaratan analisis Anda.

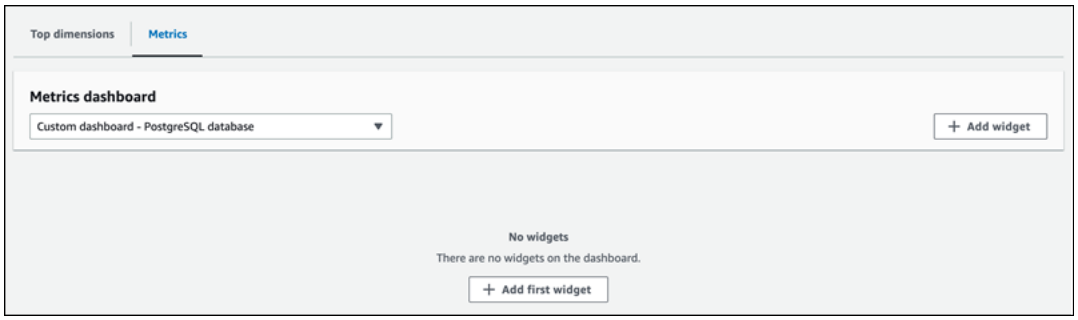
Anda dapat membuat dasbor kustom dengan memilih metrik Wawasan Performa dan CloudWatch untuk instans DB Anda. Anda dapat menggunakan dasbor kustom ini untuk instans DB lainnya dari jenis mesin basis data yang sama di akun AWS Anda.

> **Note**
>
> Dasbor yang disesuaikan mendukung hingga 50 metrik.

Gunakan menu pengaturan widget untuk mengedit atau menghapus dasbor, dan memindahkan atau mengubah ukuran jendela widget.

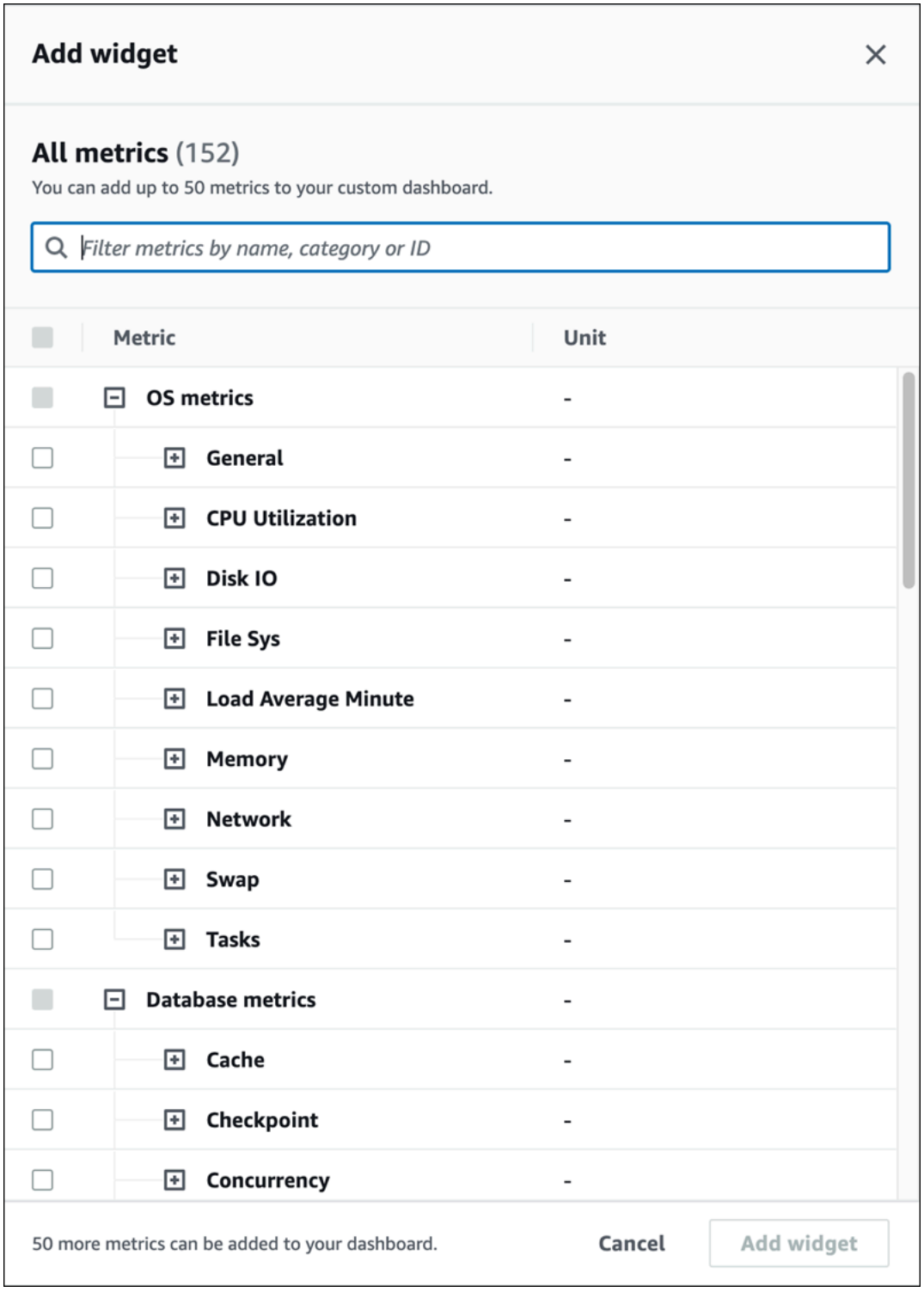
Untuk membuat dasbor kustom dengan Wawasan Performa di panel navigasi:

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.
4. Gulir ke tab Metrik di jendela.
5. Pilih dasbor kustom dari daftar drop-down. Contoh berikut menampilkan pembuatan dasbor kustom.



6. Pilih Tambah widget untuk membuka jendela Tambah widget. Anda dapat membuka dan melihat metrik sistem operasi (OS), metrik basis data, dan metrik CloudWatch yang tersedia di jendela.

   Contoh berikut menampilkan jendela Tambah widget dengan metrik.



7. Pilih metrik yang ingin Anda lihat di dasbor dan pilih Tambah widget. Anda dapat menggunakan bidang pencarian untuk menemukan metrik tertentu.

   Metrik yang dipilih akan muncul di dasbor Anda.

8. (Opsional) Jika Anda ingin memodifikasi atau menghapus dasbor, pilih ikon pengaturan di kanan atas widget, lalu pilih salah satu tindakan berikut di menu.

   - Edit - Memodifikasi daftar metrik di jendela. Pilih Perbarui widget setelah Anda memilih metrik untuk dasbor Anda.
   - Hapus - Menghapus widget. Pilih Hapus di jendela konfirmasi.

Memilih dasbor yang telah dikonfigurasi sebelumnya dengan Wawasan Performa di panel navigasi

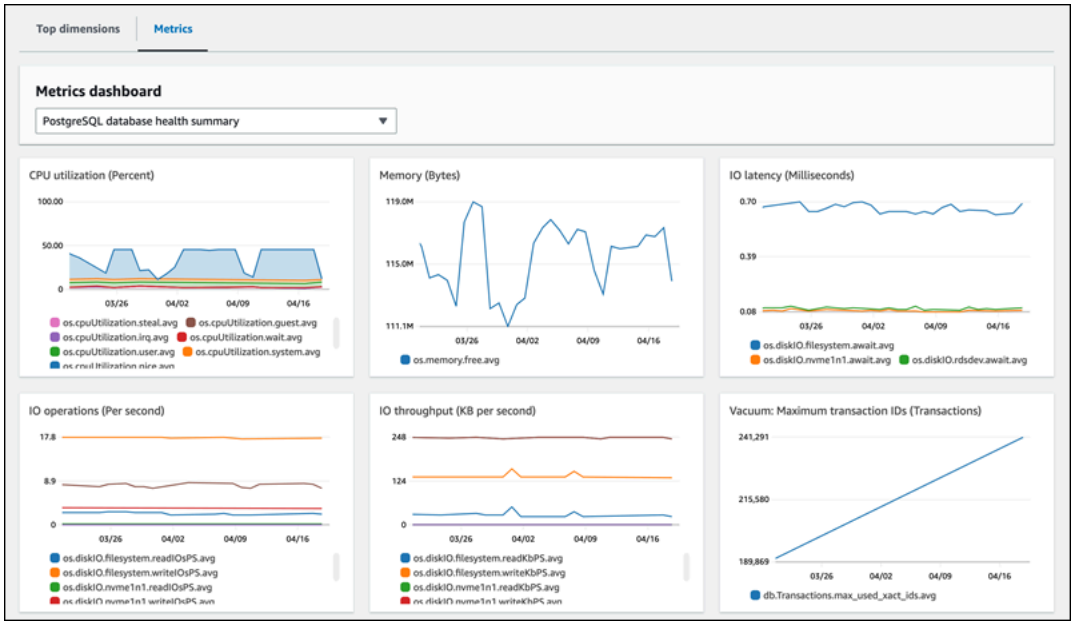
Anda dapat melihat metrik yang paling umum digunakan dengan dasbor yang telah dikonfigurasi sebelumnya. Dasbor ini membantu mendiagnosis masalah performa mesin basis data dan mengurangi waktu pemulihan rata-rata dari jam menjadi menit.

> **Note**
>
> Dasbor ini tidak dapat diedit.

Untuk memilih dasbor yang telah dikonfigurasi sebelumnya dengan Wawasan Performa di panel navigasi:

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.
4. Gulir ke tab Metrik di jendela.
5. Pilih dasbor yang telah dikonfigurasi sebelumnya dari daftar drop-down.

   Anda dapat melihat metrik untuk instans DB di dasbor. Contoh berikut menampilkan dasbor metrik yang telah dikonfigurasi sebelumnya.


Top dimensions
Metrics
Metrics dashboard
PostgreSQL database health summary
CPU utilization (Percent)
100.00
50.00
0
03/26
04/02
04/09
04/16
os.cpuUtilization.steal.avg
os.cpuUtilization.guest.avg
os.cpuUtilization.irq.avg
os.cpuUtilization.wait.avg
os.cpuUtilization.user.avg
os.cpuUtilization.system.avg
Memory (Bytes)
119.0M
115.0M
111.1M
os.memory.free.avg
IO latency (Milliseconds)
0.70
0.39
0.08
os.diskIO.filesystem.await.avg
os.diskIO.nvme1n1.await.avg
os.diskIO.rdsdev.await.avg
IO operations (Per second)
17.8
8.9
os.diskIO.filesystem.readIOsPS.avg
os.diskIO.filesystem.writeIOsPS.avg
os.diskIO.nvme1n1.readIOsPS.avg
IO throughput (KB per second)
248
124
os.diskIO.filesystem.readKbPS.avg
os.diskIO.filesystem.writeKbPS.avg
os.diskIO.nvme1n1.readKbPS.avg
Vacuum: Maximum transaction IDs (Transactions)
241,291
215,580
189,869
db.Transactions.max_used_xact_ids.avg

Memantau metrik Amazon Aurora dengan Amazon CloudWatch

Amazon CloudWatch adalah sebuah repositori metrik. Repositori ini mengumpulkan dan mengolah data mentah dari Amazon Aurora menjadi metrik-metrik waktu nyaris nyata yang dapat dibaca. Lihat daftar lengkap metrik-metrik Amazon Aurora yang dikirim ke CloudWatch di Referensi metrik untuk Amazon Aurora.

Topik

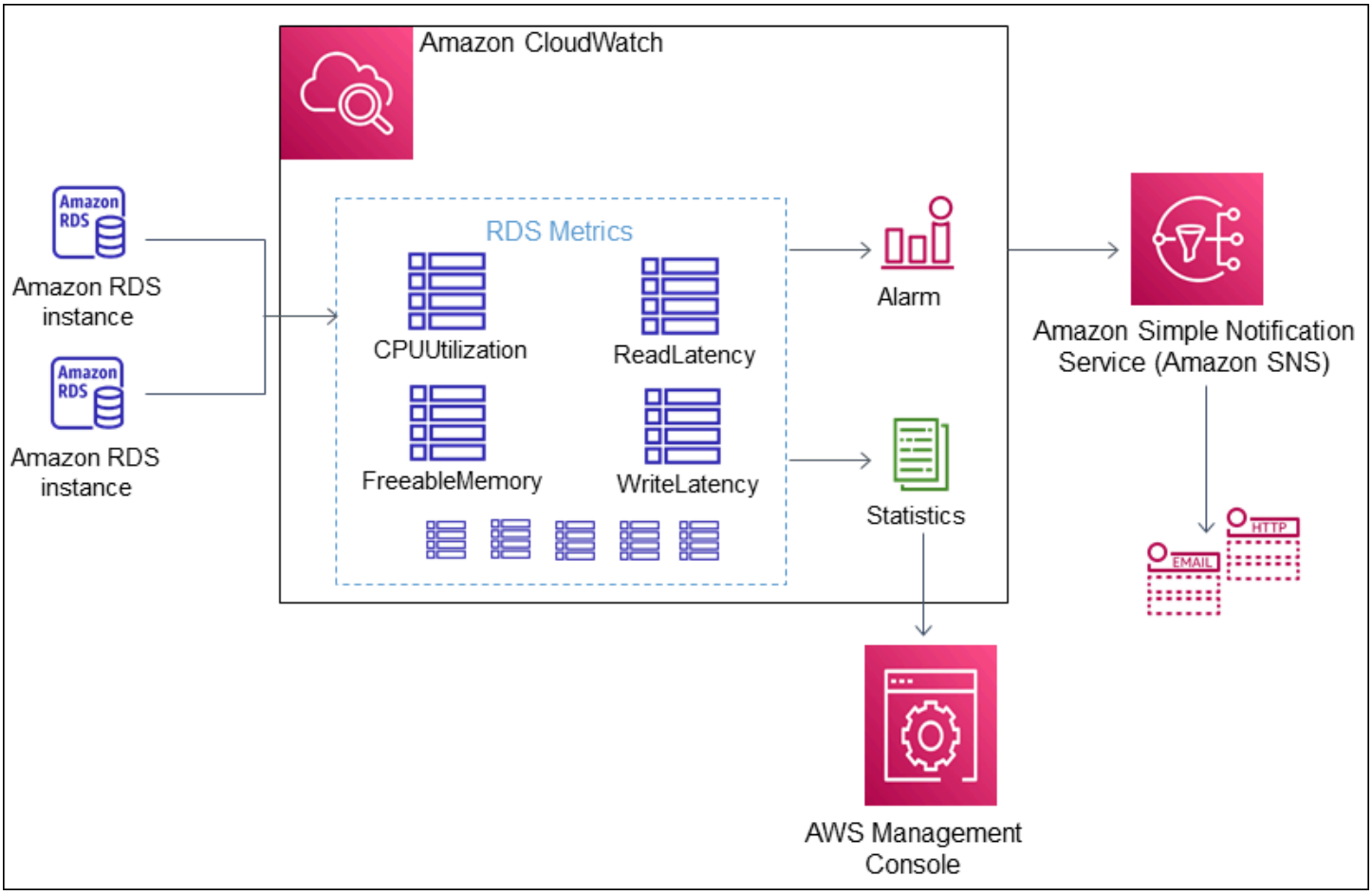
- Ikhtisar Amazon Aurora dan Amazon CloudWatch
- Melihat metrik cluster DB di CloudWatch konsol dan AWS CLI
- Mengekspor metrik Performance Insights ke CloudWatch
- Membuat alarm CloudWatch untuk memantau Amazon Aurora

Ikhtisar Amazon Aurora dan Amazon CloudWatch

Secara bawaan, Amazon Aurora mengirim secara otomatis data metrik ke CloudWatch dalam beberapa periode 1 menit. Misalnya, metrik `CPUUtilization` mencatat persentase pemanfaatan CPU untuk instans basis data dari waktu ke waktu. Titik data dengan periode 60 detik (1 menit) tersedia selama 15 hari Ini berarti bahwa Anda dapat mengakses informasi historis dan melihat bagaimana aplikasi web atau layanan Anda berkinerja.

Anda kini mengekspor dasbor metrik Wawasan Performa dari Amazon RDS ke Amazon CloudWatch. Anda dapat mengekspor dasbor metrik yang telah dikonfigurasikan atau yang disesuaikan sebagai dasbor baru atau menambahkannya ke dasbor CloudWatch yang ada. Dasbor yang diekspor tersedia untuk dilihat di konsol CloudWatch. Lihat informasi selengkapnya tentang cara mengekspor dasbor metrik Wawasan Performa ke CloudWatch di Mengekspor metrik Performance Insights ke CloudWatch.

Seperti yang ditunjukkan pada diagram berikut, Anda dapat menyiapkan alarm untuk metrik CloudWatch. Misalnya, Anda dapat membuat alarm yang memberi sinyal ketika penggunaan CPU untuk sebuah instans melebihi 70%. Anda dapat mengonfigurasikan Amazon Simple Notification Service agar mengirimi Anda email saat ambang batas itu dilewati.



Amazon RDS menerbitkan berbagai tipe metrik berikut ke Amazon CloudWatch:

- Metrik-metrik Aurora di tingkat klaster dan instans

  Lihat tabel metrik-metrik ini di CloudWatch Metrik Amazon untuk Amazon Aurora.
- Metrik-metrik Wawasan Performa

  Lihat tabel metrik-metrik ini di CloudWatch Metrik Amazon untuk Performance Insights dan Metrik penghitung Wawasan Performa.
- Metrik-metrik Pemantauan Disempurnakan (dipublikasikan ke Log Amazon CloudWatch)

  Lihat tabel metrik-metrik ini di Metrik OS dalam Pemantauan yang Disempurnakan.
- Metrik-metrik penggunaan untuk kuota layanan Amazon RDS di Akun AWS Anda

  Lihat tabel metrik-metrik ini di Metrik CloudWatch penggunaan Amazon untuk . Lihat informasi selengkapnya tentang kuota Amazon RDS di Kuota dan batasan untuk Amazon Aurora.

Lihat informasi selengkapnya tentang CloudWatch di Apakah Amazon CloudWatch? dalam Panduan Pengguna Amazon CloudWatch. Lihat informasi selengkapnya tentang retensi metrik CloudWatch di Retensi metrik.

Melihat metrik cluster DB di CloudWatch konsol dan AWS CLI

Setelah itu, Anda dapat menemukan detail tentang cara melihat metrik untuk instans DB Anda menggunakan CloudWatch. Untuk informasi tentang metrik pemantauan untuk sistem operasi instans DB Anda secara real time menggunakan CloudWatch Log, lihatMemantau metrik OS dengan Pemantauan yang Disempurnakan.

Saat Anda menggunakan sumber Amazon Aurora, Aurora mengirimkan metrik dan dimensi ke Amazon setiap menit. CloudWatch

Sekarang Anda dapat mengekspor dasbor metrik Performance Insights dari Amazon RDS ke CloudWatch Amazon dan melihat metrik ini di konsol. CloudWatch Untuk informasi selengkapnya tentang cara mengekspor dasbor metrik Performance Insights ke, lihat. CloudWatch Mengekspor metrik Performance Insights ke CloudWatch

Gunakan prosedur berikut untuk melihat metrik Aurora di CloudWatch konsol dan CLI.

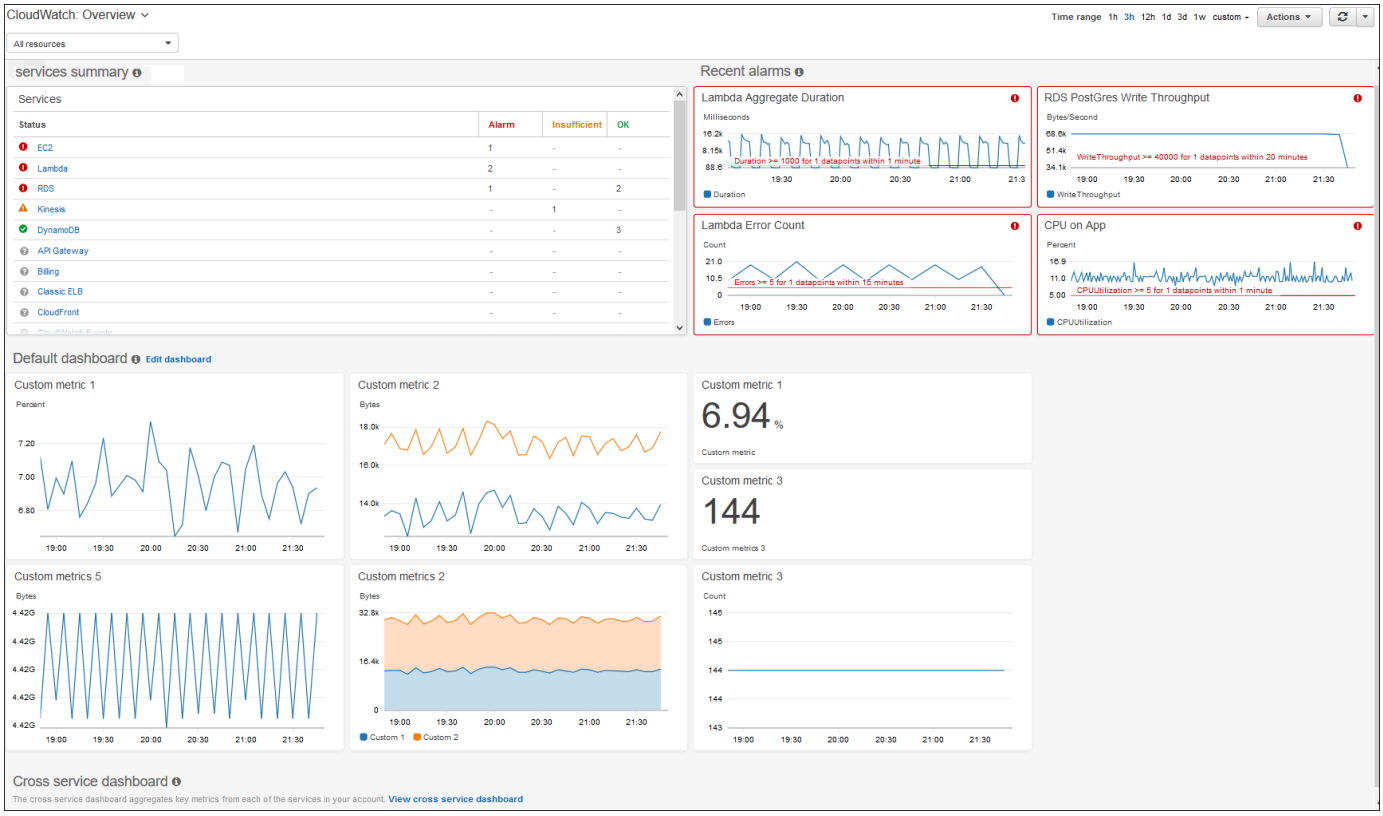
Konsol

Untuk melihat metrik menggunakan konsol Amazon CloudWatch

Metrik dikelompokkan berdasarkan ruang nama layanan dahulu, lalu berdasarkan berbagai kombinasi dimensi dalam setiap ruang nama.

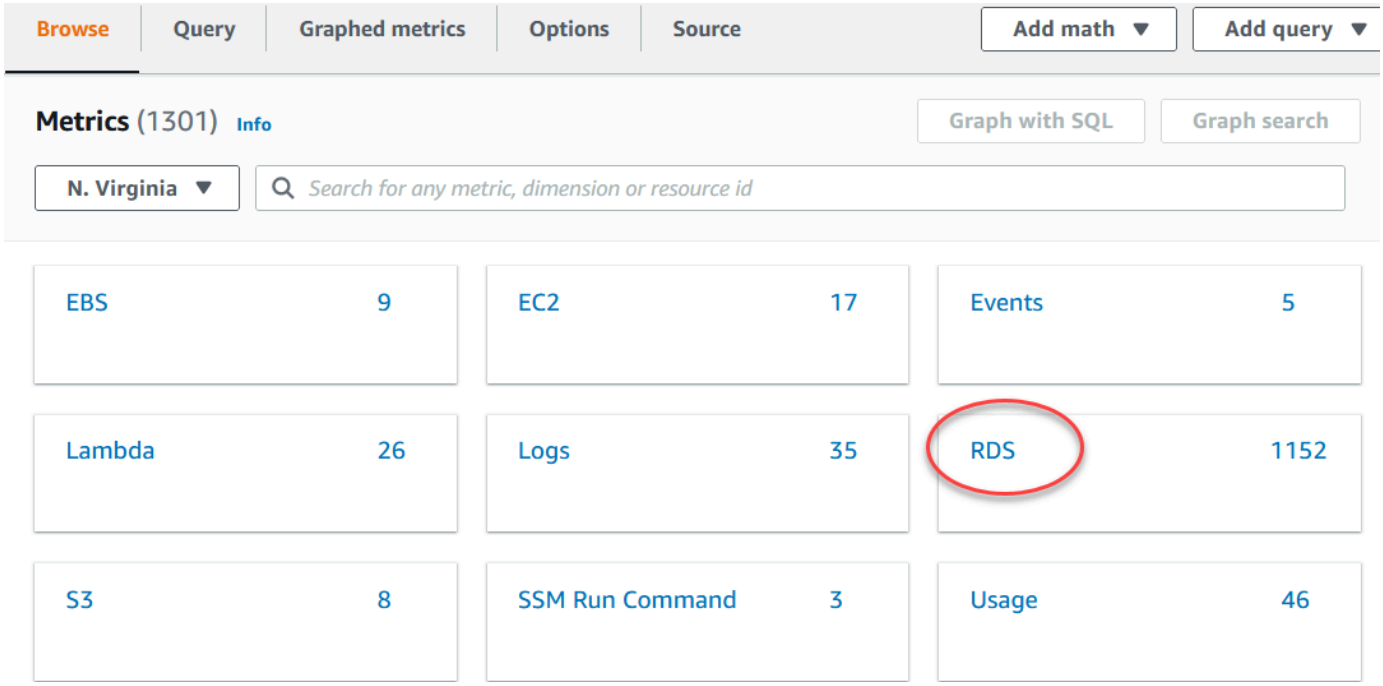
1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.

   Halaman beranda CloudWatch ikhtisar muncul.



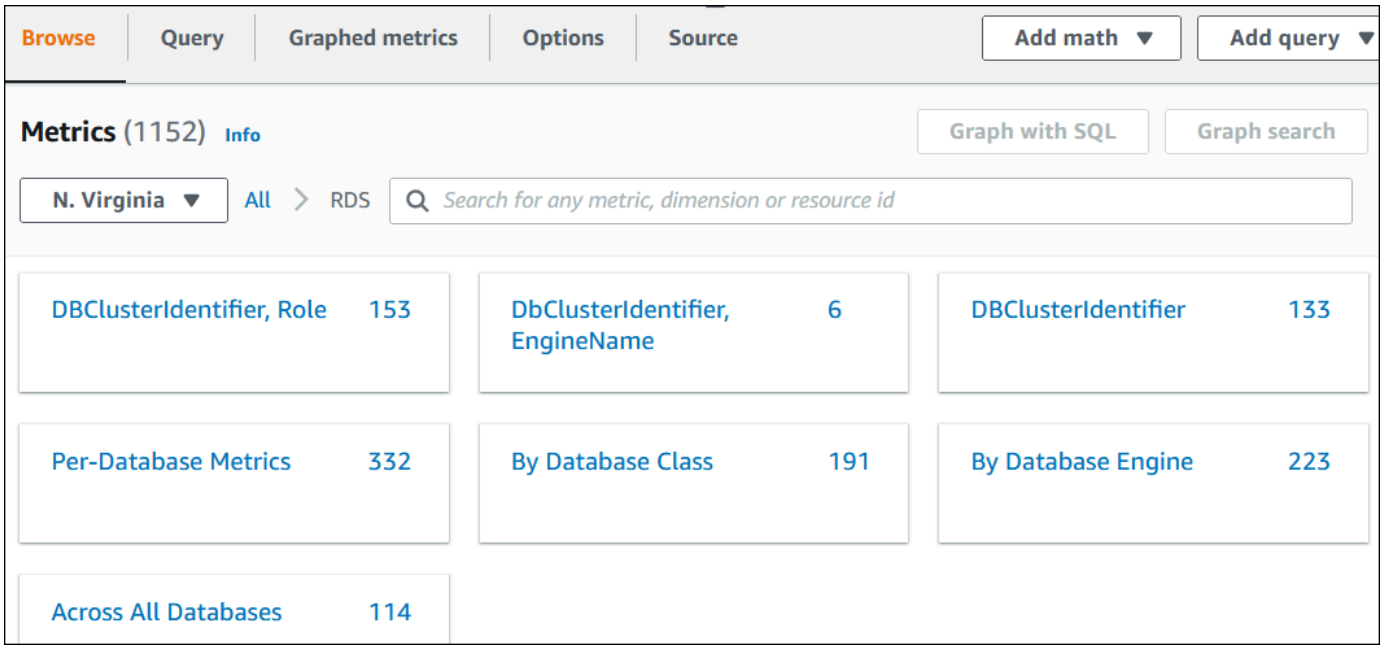
2. Jika perlu, ubah Wilayah AWS. Dari bilah navigasi, pilih Wilayah AWS tempat sumber daya AWS Anda berada. Lihat informasi yang lebih lengkap di Kawasan dan titik akhir.

3. Di panel navigasi, pilih Metrik, dan lalu Semua metrik.

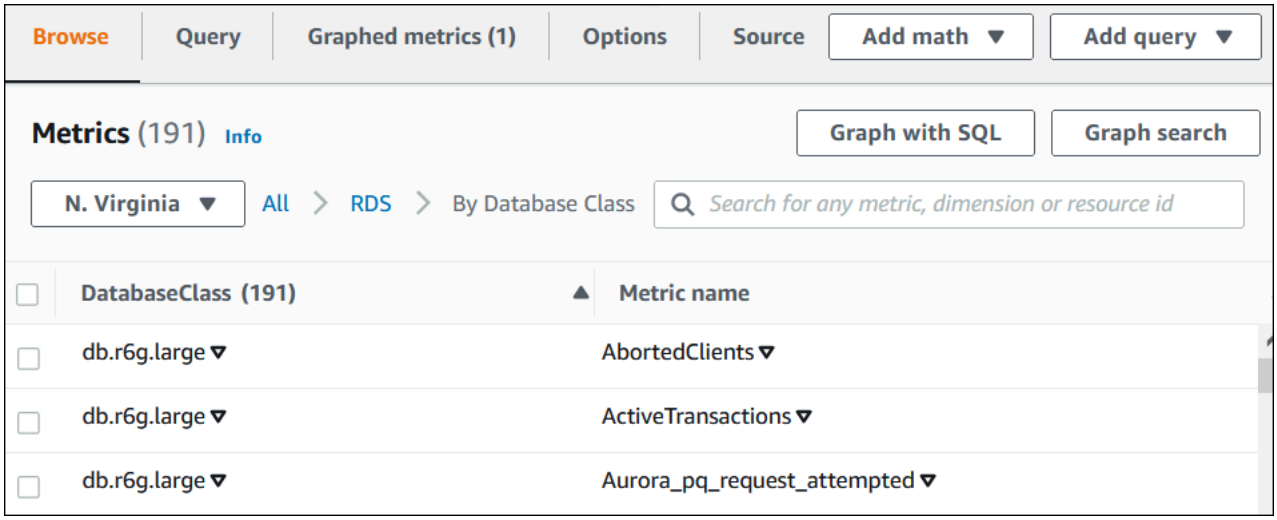


4. Gulir turun dan pilih ruang nama metrik RDS.

Halaman ini menampilkan dimensi-dimensi Amazon Aurora. Untuk deskripsi semua dimensi ini, lihat Dimensi-dimensi Amazon CloudWatch untuk Aurora.



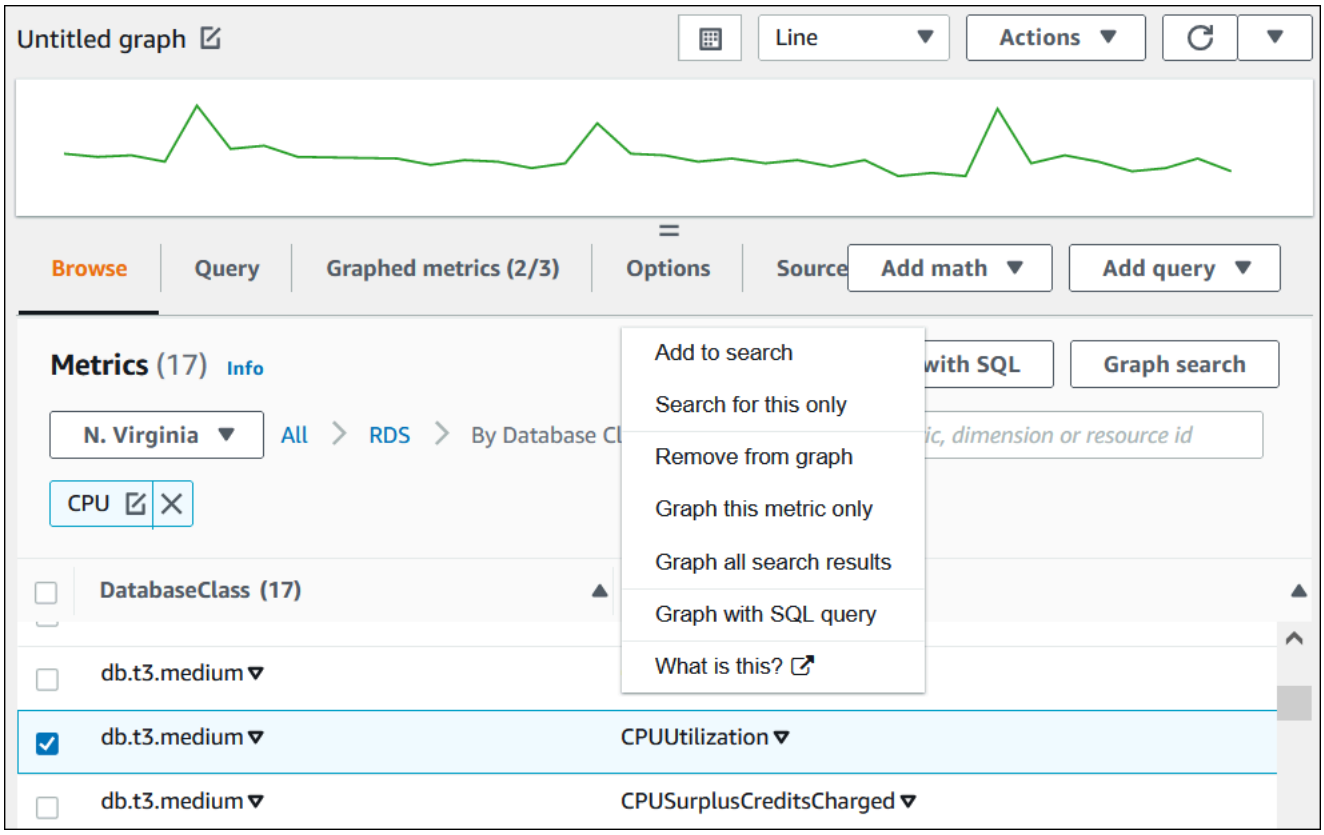
5. Pilih dimensi metrik, misalnya Berdasarkan Kelas Basis Data.



6. Lakukan salah satu tindakan berikut:

   - Untuk mengurutkan metrik, gunakan judul kolom.
   - Untuk membuat grafik metrik, pilih kotak centang di samping metrik.
   - Untuk memfilter berdasarkan sumber daya, pilih ID sumber daya, lalu pilih Tambahkan ke pencarian.
   - Untuk memfilter berdasarkan metrik, pilih nama metrik, lalu pilih Tambahkan ke pencarian.

Contoh berikut memfilter kelas db.t3.medium dan membuat grafik metrik CPUUtilization.



Anda dapat menemukan detail tentang cara menganalisis penggunaan sumber daya untuk Aurora PostgreSQL menggunakan metrik. CloudWatch Lihat informasi yang lebih lengkap di Menggunakan CloudWatch metrik Amazon untuk menganalisis penggunaan sumber daya untuk Aurora PostgreSQL

AWS CLI

Untuk mendapatkan informasi metrik dengan menggunakanAWS CLI, gunakan CloudWatch perintah `list-metrics`. Dalam contoh berikut, Anda memerinci semua metrik di ruang nama `AWS/RDS`.

```
aws cloudwatch list-metrics --namespace AWS/RDS
```

Untuk mendapatkan data metrik, gunakan perintah `get-metric-data`.

Contoh berikut mendapatkan `CPUUtilization` statistik misalnya `my-instance` selama periode 24 jam tertentu, dengan perincian 5 menit.

Buat file JSON `CPU_metric.json` dengan konten berikut.

```
{
   "StartTime" : "2023-12-25T00:00:00Z",
   "EndTime" : "2023-12-26T00:00:00Z",
   "MetricDataQueries" : [{
     "Id" : "cpu",
     "MetricStat" : {
    "Metric" : {
        "Namespace" : "AWS/RDS",
        "MetricName" : "CPUUtilization",
        "Dimensions" : [{ "Name" : "DBInstanceIdentifier" , "Value" : my-instance}]
    },
       "Period" : 360,
       "Stat" : "Minimum"
     }
   }]
}
```

Example

Untuk Linux, macOS, atau Unix:

```
aws cloudwatch get-metric-data \
    --cli-input-json file://CPU_metric.json
```

Untuk Windows:

```
aws cloudwatch get-metric-data ^
     --cli-input-json file://CPU_metric.json
```

Contoh output tampil sebagai berikut:

```
{
    "MetricDataResults": [
        {
            "Id": "cpu",
            "Label": "CPUUtilization",
            "Timestamps": [
                "2023-12-15T23:48:00+00:00",
                "2023-12-15T23:42:00+00:00",
                "2023-12-15T23:30:00+00:00",
                "2023-12-15T23:24:00+00:00",
```

```
                ...
            ],
            "Values": [
                13.299778337027714,
                13.677507543049558,
                14.24976250395827,
                13.02521708695145,
                ...
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

Untuk informasi selengkapnya, lihat Mendapatkan statistik untuk metrik di Panduan CloudWatch Pengguna Amazon.

Mengekspor metrik Performance Insights ke CloudWatch

Performance Insights memungkinkan Anda mengekspor dasbor metrik yang telah dikonfigurasi sebelumnya atau kustom untuk instans DB Anda ke Amazon. CloudWatch Anda dapat mengekspor dasbor metrik sebagai dasbor baru atau menambahkannya ke CloudWatch dasbor yang ada. Saat Anda memilih untuk menambahkan dasbor ke CloudWatch dasbor yang ada, Anda dapat membuat label header sehingga metrik muncul di bagian terpisah di CloudWatch dasbor.

Anda dapat melihat dasbor metrik yang diekspor di konsol. CloudWatch Jika menambahkan metrik baru ke dasbor metrik Performance Insights setelah mengekspornya, Anda harus mengekspor dasbor ini lagi untuk melihat metrik baru di konsol. CloudWatch

Anda juga dapat memilih widget metrik di dasbor Performance Insights dan melihat data metrik di konsol. CloudWatch

Untuk informasi selengkapnya tentang melihat metrik di CloudWatch konsol, lihatMelihat metrik cluster DB di CloudWatch konsol dan AWS CLI.

Mengekspor metrik Performance Insights sebagai dasbor baru CloudWatch

Pilih dasbor metrik yang telah dikonfigurasi sebelumnya atau kustom dari dasbor Performance Insights dan ekspor sebagai dasbor baru. CloudWatch Anda dapat melihat dasbor yang diekspor di CloudWatch konsol.

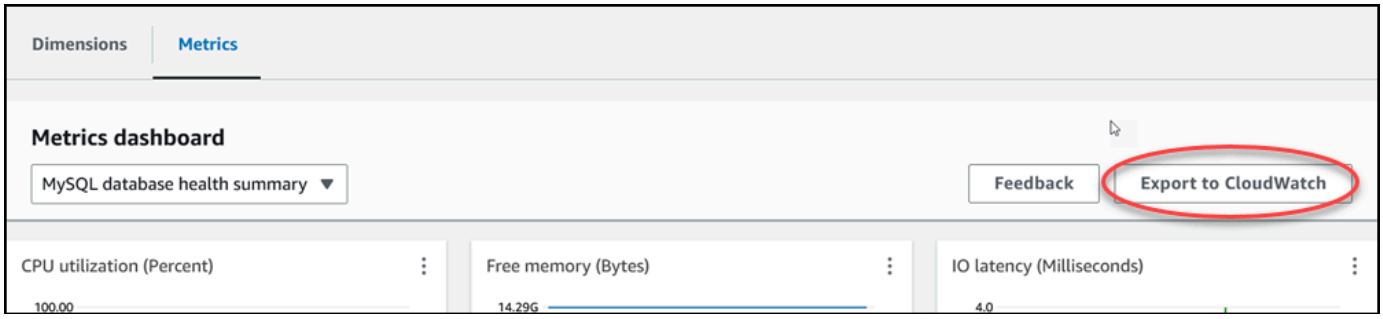
Untuk mengekspor dasbor metrik Performance Insights sebagai dasbor baru ke CloudWatch

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.

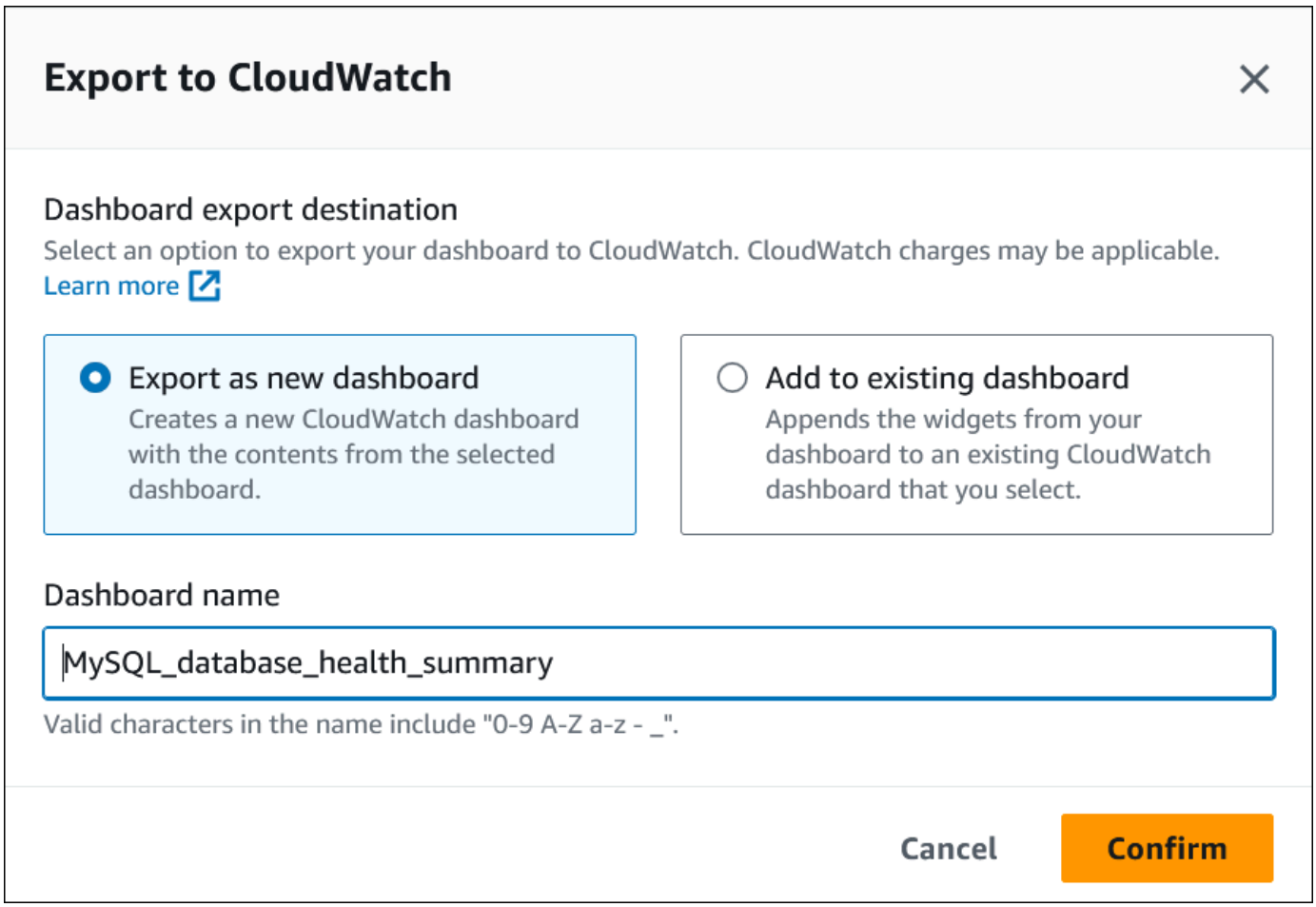
   Dasbor Wawasan Performa ditampilkan untuk instans DB.
4. Gulir ke bawah dan pilih Metrik.

   Secara default, dasbor yang telah dikonfigurasi sebelumnya dengan metrik Wawasan Performa akan muncul.
5. Pilih dasbor yang telah dikonfigurasi atau kustom, lalu pilih Ekspor ke CloudWatch.

   CloudWatchJendela Ekspor ke muncul.

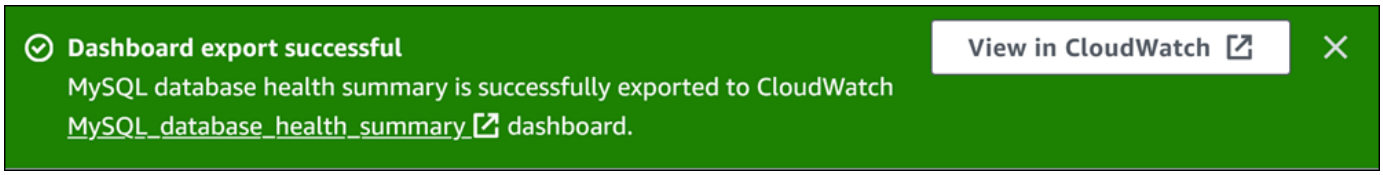


6. Pilih Ekspor sebagai dasbor baru.



7. Masukkan nama dasbor baru di kolom Nama dasbor dan pilih Konfirmasi.

   Banner menampilkan pesan setelah ekspor dasbor berhasil.



8. Pilih tautan atau Lihat CloudWatch di spanduk untuk melihat dasbor metrik di CloudWatch konsol.

Menambahkan metrik Performance Insights ke dasbor yang ada CloudWatch

Tambahkan dasbor metrik yang telah dikonfigurasi sebelumnya atau kustom ke dasbor yang ada CloudWatch . Anda dapat menambahkan label ke dasbor metrik untuk muncul di bagian terpisah di CloudWatch dasbor.

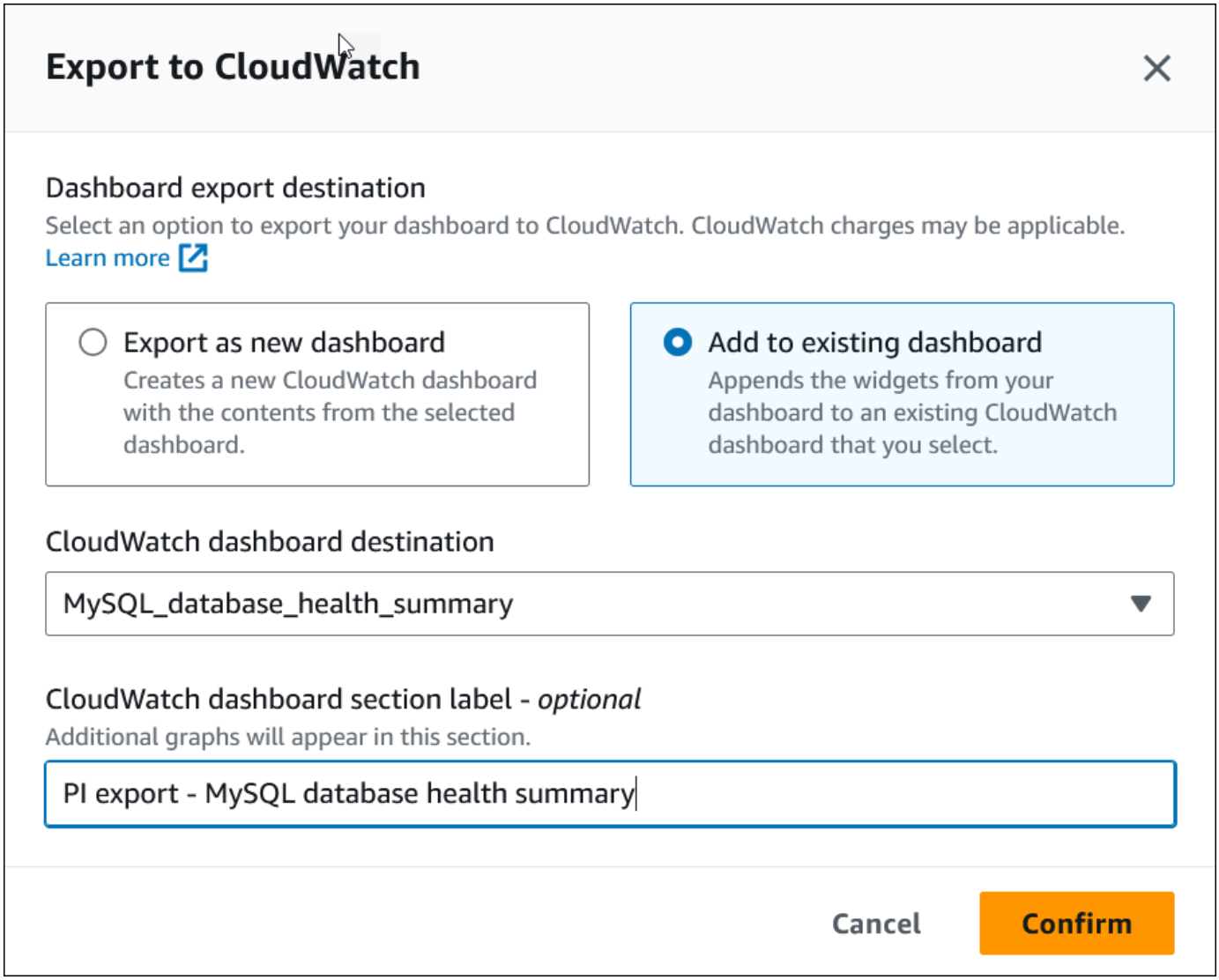
Untuk mengekspor metrik ke dasbor yang ada CloudWatch

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.

   Dasbor Wawasan Performa ditampilkan untuk instans DB.
4. Gulir ke bawah dan pilih Metrik.

   Secara default, dasbor yang telah dikonfigurasi sebelumnya dengan metrik Wawasan Performa akan muncul.
5. Pilih dasbor yang telah dikonfigurasi atau kustom, lalu pilih Ekspor ke CloudWatch.

   CloudWatchJendela Ekspor ke muncul.
6. Pilih Tambahkan ke dasbor yang ada.



7. Tentukan tujuan dan label dasbor, lalu pilih Konfirmasi.

   - CloudWatch tujuan dasbor - Pilih CloudWatch dasbor yang ada.
   - CloudWatch label bagian dasbor - opsional - Masukkan nama untuk metrik Performance Insights untuk muncul di bagian ini di dasbor. CloudWatch

   Banner menampilkan pesan setelah ekspor dasbor berhasil.

8. Pilih tautan atau Lihat CloudWatch di spanduk untuk melihat dasbor metrik di CloudWatch konsol.

Melihat widget metrik Performance Insights di CloudWatch

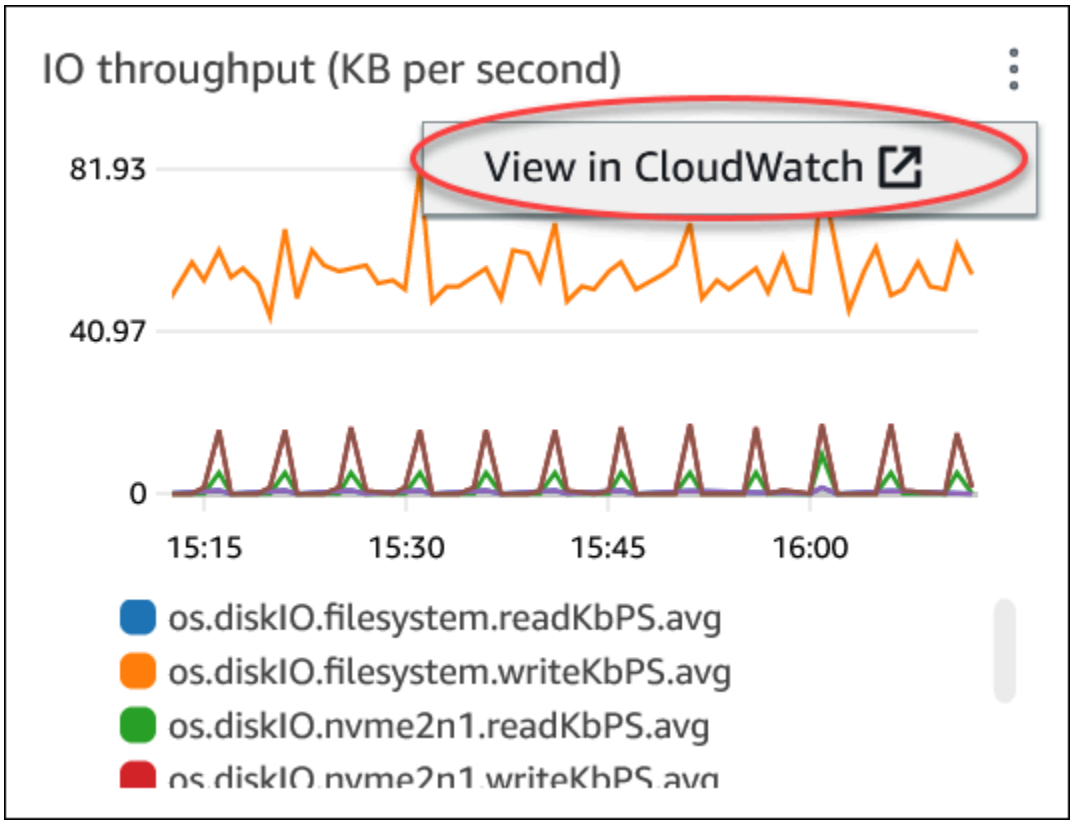
Pilih widget metrik Performance Insights di dasbor Amazon RDS Performance Insights dan lihat data metrik di konsol. CloudWatch

Untuk mengekspor widget metrik dan melihat data metrik di konsol CloudWatch

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.

   Dasbor Wawasan Performa ditampilkan untuk instans DB.
4. Gulir ke bawah ke Metrik.

   Secara default, dasbor yang telah dikonfigurasi sebelumnya dengan metrik Wawasan Performa akan muncul.
5. Pilih widget metrik dan kemudian pilih Lihat CloudWatch di dalam menu.



   Data metrik muncul di CloudWatch konsol.

Membuat alarm CloudWatch untuk memantau Amazon Aurora

Anda dapat membuat alarm CloudWatch yang mengirimkan pesan Amazon SNS ketika status alarm berubah. Alarm mengawasi metrik tunggal selama periode waktu yang Anda tentukan. Alarm tersebut juga dapat melakukan satu atau beberapa tindakan berdasarkan nilai metrik yang relatif terhadap ambang batas tertentu selama beberapa periode waktu. Tindakannya adalah notifikasi yang dikirim ke topik Amazon SNS atau kebijakan Amazon EC2 Auto Scaling.

Alarm hanya menginvokasi tindakan untuk status dengan perubahan berkelanjutan. Alarm CloudWatch tidak menginvokasi tindakan karena alarm tersebut berada dalam status tertentu. Status harus diubah dan dipertahankan selama jangka waktu tertentu.

> **Note**
>
> Untuk Aurora, gunakan metrik peran `WRITER` atau `READER` untuk menyiapkan alarm, bukan mengandalkan metrik untuk instans basis data tertentu. Peran instans basis data Aurora dapat mengubah peran dari waktu ke waktu. Anda dapat menemukan metrik berbasis peran tersebut dalam konsol CloudWatch.
> Aurora Auto Scaling mengatur alarm secara otomatis berdasarkan metrik peran `READER`. Lihat informasi selengkapnya tentang Aurora Auto Scaling di Menggunakan Amazon Aurora Auto Scaling dengan Replika Aurora.

Anda dapat menggunakan fungsi matematika metrik DB_PERF_INSIGHTS di konsol CloudWatch guna melakukan kueri Amazon RDS untuk metrik penghitung Wawasan Kinerja. Fungsi DB_PERF_INSIGHTS juga menyertakan metrik DBLoad pada interval submenit. Anda dapat mengatur alarm CloudWatch berdasarkan metrik ini.

Lihat detail selengkapnya tentang cara membuat alarm di Membuat alarm di metrik penghitung Wawasan Kinerja dari basis data AWS.

Mengatur alarm menggunakan AWS CLI

- Panggil `put-metric-alarm`. Lihat informasi selengkapnya di Referensi Perintah AWS CLI.

Mengatur alarm menggunakan API CloudWatch

- Panggil `PutMetricAlarm`. Lihat informasi selengkapnya di Referensi API Amazon CloudWatch

Lihat informasi selengkapnya tentang mengatur topik Amazon SNS dan membuat alarm di Menggunakan alarm Amazon CloudWatch.

Memantau muatan DB dengan Wawasan Performa di Amazon Aurora

Wawasan Performa memperluas fitur pemantauan Amazon Aurora yang sudah ada untuk mengilustrasikan dan membantu Anda menganalisis performa klaster. Dengan dasbor Wawasan Performa, Anda dapat memvisualisasikan muatan basis data pada muatan klaster Amazon Aurora dan memfilter muatan menurut peristiwa tunggu, pernyataan SQL, host, atau pengguna. Untuk informasi tentang cara menggunakan Wawasan Performa dengan Amazon DocumentDB, lihat Panduan Developer Amazon DocumentDB.

Topik

- Ikhtisar Wawasan Performa tentang Amazon Aurora
- Mengaktifkan dan menonaktifkan Wawasan Performa
- Mengaktifkan Skema Performa untuk Wawasan Performa di Aurora MySQL
- Mengonfigurasi kebijakan akses untuk Wawasan Performa
- Menganalisis metrik dengan dasbor Wawasan Performa
- Melihat rekomendasi proaktif Performance Insights
- Mengambil metrik dengan API Wawasan Performa
- Mencatat panggilan Wawasan Performa menggunakan AWS CloudTrail

Ikhtisar Wawasan Performa tentang Amazon Aurora

Secara default, Wawasan Performa diaktifkan di wizard pembuatan konsol untuk semua mesin Amazon RDS. Jika Anda mengaktifkan Wawasan Performa di tingkat klaster DB, Wawasan Performa akan diaktifkan untuk setiap instans DB di klaster tersebut. Jika Anda memiliki lebih dari satu basis data di instans DB, Wawasan Performa akan menggabungkan data performa.

Anda dapat menemukan ikhtisar Wawasan Performa untuk Amazon Aurora dalam video berikut.

Menggunakan Wawasan Performa untuk Menganalisis Kinerja Amazon Aurora PostgreSQL

Topik

- Muatan basis data
- CPU Maksimum
- Dukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk Wawasan Performa

- Harga dan retensi data untuk Wawasan Performa

Muatan basis data

Muatan basis data (muatan DB) mengukur tingkat aktivitas sesi dalam basis data Anda. Metrik utama dalam Wawasan Performa adalah `DBLoad`, yang dikumpulkan setiap detik.

Topik

- Sesi aktif
- Sesi aktif rata-rata
- Eksekusi aktif rata-rata
- Dimensi

Sesi aktif

Sesi basis data mewakili dialog aplikasi dengan basis data relasional. Sesi aktif adalah koneksi yang mengirimkan tugas ke mesin DB dan sedang menunggu tanggapan.

Sesi dianggap aktif jika berjalan di CPU atau menunggu sumber daya tersedia sehingga dapat dilanjutkan. Misalnya, sesi aktif mungkin menunggu halaman (atau blok) dibaca ke dalam memori, dan kemudian menggunakan CPU saat membaca data dari halaman.

Sesi aktif rata-rata

Sesi aktif rata-rata (AAS) adalah unit untuk metrik `DBLoad` dalam Wawasan Performa. Ini mengukur berapa banyak sesi yang aktif secara bersamaan di basis data.

Setiap detik, Wawasan Performa mengambil sampel jumlah sesi yang secara bersamaan menjalankan kueri. Untuk setiap sesi aktif, Wawasan Performa mengumpulkan data berikut:

- Pernyataan SQL
- Status sesi (berjalan pada CPU atau menunggu)
- Host
- Pengguna yang menjalankan SQL

Wawasan Performa menghitung AAS dengan membagi jumlah total sesi dengan jumlah sampel selama periode waktu tertentu. Misalnya, tabel berikut menunjukkan 5 sampel berturut-turut dari kueri yang berjalan yang diambil dengan interval 1 detik.

Sampel	Jumlah sesi yang menjalankan kueri	AAS	Penghitungan
1	2	2	Total 2 sesi/1 sampel
2	0	1	Total 2 sesi/2 sampel
3	4	2	Total 6 sesi/3 sampel
4	0	1,5	Total 6 sesi/4 sampel
5	4	2	Total 10 sesi/5 sampel

Pada contoh sebelumnya, muatan DB untuk interval waktu tersebut adalah 2 AAS. Pengukuran ini berarti bahwa rata-rata ada 2 sesi aktif pada waktu tertentu selama interval tersebut ketika 5 sampel diambil.

Analogi untuk muatan DB adalah aktivitas pekerja di gudang. Misalkan gudang mempekerjakan 100 pekerja. Jika 1 pesanan masuk, 1 pekerja memenuhi pesanan sedangkan 99 pekerja menganggur. Jika 100 pesanan masuk, 100 pekerja semuanya memenuhi pesanan secara bersamaan. Jika setiap 15 menit seorang manajer menuliskan berapa banyak pekerja yang aktif secara bersamaan, menambahkan angka-angka ini di penghujung hari, dan kemudian membagi totalnya dengan jumlah sampel, manajer menghitung jumlah rata-rata pekerja yang aktif pada waktu tertentu. Jika rata-rata 50 pekerja kemarin dan 75 pekerja hari ini, maka tingkat aktivitas rata-rata di gudang meningkat. Demikian pula, muatan DB meningkat seiring dengan meningkatnya aktivitas sesi basis data.

Eksekusi aktif rata-rata

Eksekusi aktif rata-rata (AAE) per detik berkaitan dengan AAS. Untuk menghitung AAE, Wawasan Performa membagi total waktu eksekusi kueri dengan interval waktu. Tabel berikut menunjukkan penghitungan AAE untuk kueri yang sama dalam tabel sebelumnya.

Waktu yang telah berlalu (dtk)	Total waktu eksekusi (detik)	AAE	Penghitungan
60	120	2	120 detik eksekusi/60 detik berlalu
120	120	1	120 detik eksekusi/120 detik berlalu
180	380	2,11	380 detik eksekusi/180 detik berlalu
240	380	1,58	380 detik eksekusi/240 detik berlalu
300	600	2	600 detik eksekusi/300 detik berlalu

Dalam kebanyakan kasus, AAS dan AAE untuk sebuah kueri kira-kira sama. Namun, karena input ke penghitungan berupa sumber data yang berbeda, penghitungannya sering sedikit berbeda.

Dimensi

Metrik `db.load` berbeda dengan metrik seri waktu lainnya karena Anda dapat membaginya menjadi beberapa sub-komponen yang disebut dimensi. Anda dapat menganggap dimensi sebagai kategori “potong menurut” untuk berbagai karakteristik metrik `DBLoad`.

Saat Anda mendiagnosis masalah performa, dimensi berikut sering kali paling berguna:

Topik

- Peristiwa tunggu
- SQL Teratas

Untuk daftar lengkap dimensi untuk mesin Aurora, lihat Muatan DB diiris berdasarkan dimensi.

Peristiwa tunggu

Peristiwa tunggu menyebabkan pernyataan SQL menunggu peristiwa tertentu terjadi sebelum dapat terus berjalan. Peristiwa tunggu adalah dimensi penting, atau kategori, untuk muatan DB karena menunjukkan di mana pekerjaan terhambat.

Setiap sesi aktif berjalan di CPU atau menunggu. Misalnya, sesi menggunakan CPU ketika mencari memori untuk buffer, melakukan penghitungan, atau menjalankan kode prosedural. Ketika tidak menggunakan CPU, sesi mungkin menunggu buffer memori menjadi kosong, file data dibaca, atau log untuk ditulis. Semakin banyak waktu untuk sesi menunggu sumber daya, semakin sedikit waktu untuk sesi dijalankan di CPU.

Ketika Anda menyetel basis data, Anda sering mencoba mencari tahu sumber daya yang sedang menunggu sesi. Misalnya, dua atau tiga peristiwa tunggu mungkin menyumbang 90 persen dari muatan DB. Ukuran ini berarti bahwa, rata-rata, sesi aktif menghabiskan sebagian besar waktunya menunggu sejumlah kecil sumber daya. Jika Anda dapat mengetahui penyebab peristiwa tunggu ini, Anda dapat mencoba solusinya.

Pertimbangkan analogi pekerja gudang. Pesanan masuk. Pekerja mungkin terlambat memenuhi pesanan. Misalnya, pekerja lain mungkin sedang mengisi ulang rak, troli mungkin tidak tersedia. Atau sistem yang digunakan untuk memasukkan status pesanan lambat. Semakin lama pekerja menunggu, semakin lama waktu yang dibutuhkan untuk memenuhi pesanan. Menunggu adalah bagian alami dari alur kerja gudang, tetapi jika waktu tunggunya berlebihan, produktivitasnya menurun. Sama halnya, menunggu sesi berulang atau panjang dapat menurunkan performa basis data. Untuk informasi selengkapnya, lihat Menyetel peristiwa tunggu untuk Aurora PostgreSQL dan Menyetel peristiwa tunggu untuk Aurora MySQL di Panduan Pengguna Amazon Aurora.

Peristiwa tunggu bervariasi berdasarkan mesin DB:

- Untuk daftar kejadian tunggu yang umum terjadi di Aurora MySQL, lihat Peristiwa tunggu Aurora MySQL. Untuk mempelajari cara menyetel menggunakan peristiwa tunggu ini, lihat Menyesuaikan Aurora MySQL.
- Untuk informasi tentang semua kejadian tunggu MySQL, lihat Tabel Ringkasan Peristiwa Tunggu dalam dokumentasi MySQL.
- Untuk daftar peristiwa tunggu yang umum terjadi di Aurora PostgreSQL, lihat Peristiwa tunggu Amazon Aurora PostgreSQL. Untuk mempelajari cara menyetel menggunakan peristiwa tunggu ini, lihat Menyetel dengan peristiwa tunggu di Aurora PostgreSQL.
- Untuk informasi tentang semua kejadian tunggu PostgreSQL, lihat Tabel Pengumpul Statistik > Peristiwa Tunggu dalam dokumentasi PostgreSQL.

SQL Teratas

Saat kejadian tunggu menunjukkan kemacetan, SQL teratas menunjukkan kueri mana yang paling berkontribusi pada pemuatan DB. Misalnya, saat ini mungkin ada banyak kueri yang berjalan di basis data, tetapi kueri tunggal mungkin menggunakan 99 persen dari muatan DB. Dalam hal ini, muatan tinggi mungkin menunjukkan masalah dalam kueri.

Secara default, konsol Wawasan Performa menampilkan kueri SQL teratas yang berkontribusi pada muatan basis data. Konsol juga menunjukkan statistik yang relevan untuk setiap pernyataan. Untuk mendiagnosis masalah performa untuk pernyataan tertentu, Anda dapat memeriksa rencana pelaksanaannya.

CPU Maksimum

Di dasbor, bagan Basis data muatan mengumpulkan, menggabungkan, dan menampilkan informasi sesi. Untuk mengetahui apakah sesi aktif melebihi CPU maksimum, lihat hubungannya dengan baris vCPU Maks. Nilai vCPU Maks ditentukan oleh jumlah inti vCPU (CPU virtual) untuk instans DB Anda. Untuk Aurora Nirserver v2, vCPU Maks mencerminkan perkiraan jumlah vCPU.

Satu proses dapat berjalan pada vCPU pada satu waktu. Jika jumlah proses melebihi jumlah vCPU, proses ini akan mulai mengantre. Jika antrean meningkat, performanya akan terpengaruh. Jika muatan DB sering melampaui baris vCPU Maks dan status tunggu utamanya adalah CPU, CPU akan kelebihan muatan. Dalam kasus ini, sebaiknya Anda membatasi koneksi ke instans, menyesuaikan kueri SQL apa pun dengan muatan CPU yang tinggi, atau mempertimbangkan kelas instans yang lebih besar. Instans yang tinggi dan konsisten dari setiap status tunggu menunjukkan bahwa mungkin terjadi kemacetan atau masalah ketidakcocokan sumber daya yang perlu diselesaikan. Hal ini bisa terjadi meski muatan DB tidak melampaui baris vCPU Maks.

Dukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk Wawasan Performa

Tabel berikut berisi mesin DB Amazon Aurora yang mendukung Wawasan Performa.

Mesin DB Amazon Aurora	Versi mesin dan Wilayah yang didukung	Pembatasan kelas instans
Edisi yang Kompatibel dengan	Untuk informasi selengkapnya tentang versi dan ketersediaan Wilayah Wawasan Performa	Wawasan Performa memiliki batasan kelas mesin berikut:

Mesin DB Amazon Aurora	Versi mesin dan Wilayah yang didukung	Pembatasan kelas instans
Amazon Aurora MySQL	dengan Aurora MySQL, lihat Wawasan Performa dengan Aurora MySQL.	• db.t2 – Tidak didukung • db.t3 – Tidak didukung • db.t4g.micro dan db.t4g.small – Tidak didukung
Edisi yang Kompatibel dengan Amazon Aurora PostgreSQL	Untuk informasi selengkapnya tentang versi dan ketersediaan Wilayah Wawasan Performa dengan Aurora MySQL, lihat Wawasan Performa dengan Aurora PostgreSQL.	N/A

Dukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk fitur Wawasan Performa

Tabel berikut berisi mesin DB Amazon Aurora yang mendukung fitur Wawasan Performa.

Fitur	Tingkat harga	Wilayah yang didukung	Mesin DB yang didukung	Kelas instans yang didukung
Statistik SQL untuk Wawasan Performa	Semua	Semua	Semua	Semua
Menganalisis performa basis data selama periode waktu tertentu	Khusus tingkat berbayar	• AS Timur (Ohio) • AS Timur (Virginia Utara) • AS Barat (California Utara)	Semua	Semua kecuali db.server less (Aurora Serverless v2)

Fitur	Tingkat harga	Wilayah yang didukung	Mesin DB yang didukung	Kelas instans yang didukung
		• AS Barat (Oregon) • Asia Pasifik (Mumbai) • Asia Pasifik (Seoul) • Asia Pasifik (Singapura) • Asia Pasifik (Sydney) • Asia Pasifik (Tokyo) • Kanada (Pusat) • Eropa (Frankfurt) • Eropa (Irlandia) • Eropa (London) • Eropa (Paris) • Eropa (Stockholm)		

Fitur	Tingkat harga	Wilayah yang didukung	Mesin DB yang didukung	Kelas instans yang didukung
Melihat rekomendasi proaktif Performance Insights	Khusus tingkat berbayar	• AS Timur (Ohio) • AS Timur (Virginia Utara) • AS Barat (California Utara) • AS Barat (Oregon) • Asia Pasifik (Mumbai) • Asia Pasifik (Seoul) • Asia Pasifik (Singapura) • Asia Pasifik (Sydney) • Asia Pasifik (Tokyo) • Kanada (Pusat) • Eropa (Frankfurt) • Eropa (Irlandia) • Eropa (London) • Eropa (Paris)	Semua	Semua kecuali db.serverless (Aurora Serverless v2)

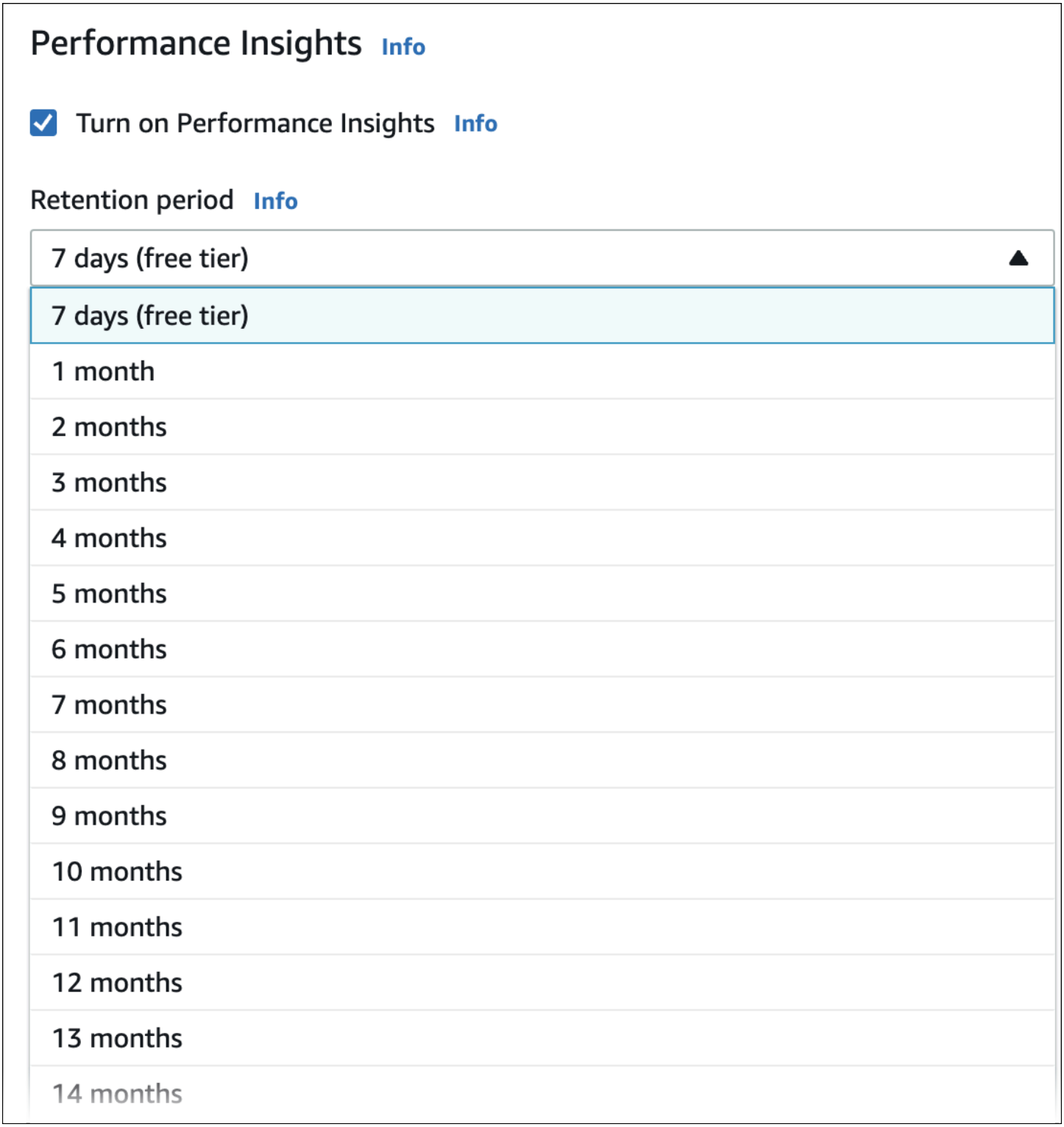
Fitur	Tingkat harga	Wilayah yang didukung	Mesin DB yang didukung	Kelas instans yang didukung
		• Eropa (Stockholm) • Amerika Selatan (Sao Paulo)		

Harga dan retensi data untuk Wawasan Performa

Secara default, Wawasan Performa menawarkan tingkat gratis yang mencakup riwayat data performa selama 7 hari dan 1 juta permintaan API per bulan. Anda juga dapat membeli periode retensi yang lebih lama. Untuk informasi harga selengkapnya, lihat Harga Wawasan Performa.

Di konsol RDS, Anda dapat memilih salah satu periode retensi berikut untuk data Wawasan Performa:

- Default (7 hari)
- ***n*** bulan, di mana ***n*** adalah angka dari 1—24



Untuk mempelajari cara menetapkan periode retensi menggunakanAWS CLI, lihat AWS CLI.

Mengaktifkan dan menonaktifkan Wawasan Performa

Anda dapat mengaktifkan Wawasan Performa untuk klaster saat Anda membuatnya. Jika diperlukan, Anda dapat menonaktifkannya nanti di tingkat instans untuk instans apa pun di klaster DB Anda. Mengaktifkan dan menonaktifkan Wawasan Performa tidak menyebabkan waktu henti, reboot, atau failover.

> **Note**
>
> Skema Performa adalah alat performa opsional yang digunakan oleh Aurora MySQL . Jika Anda mengaktifkan atau menonaktifkan Skema Performa, Anda perlu me-reboot. Namun, jika Anda mengaktifkan atau menonaktifkan Wawasan Performa, Anda tidak perlu me-reboot. Untuk informasi selengkapnya, lihat Mengaktifkan Skema Performa untuk Wawasan Performa di Aurora MySQL.

Jika Anda menggunakan Wawasan Performa dengan basis data global Aurora, aktifkan Wawasan Performa satu per satu untuk instans DB di setiap Wilayah AWS. Untuk detailnya, lihat Memantau basis data global Amazon Aurora dengan Wawasan Performa Amazon RDS.

Agen Wawasan Performa menggunakan CPU dan memori terbatas di host DB. Ketika muatan DB tinggi, agen membatasi dampak performa dengan mengurangi frekuensi pengumpulan data.

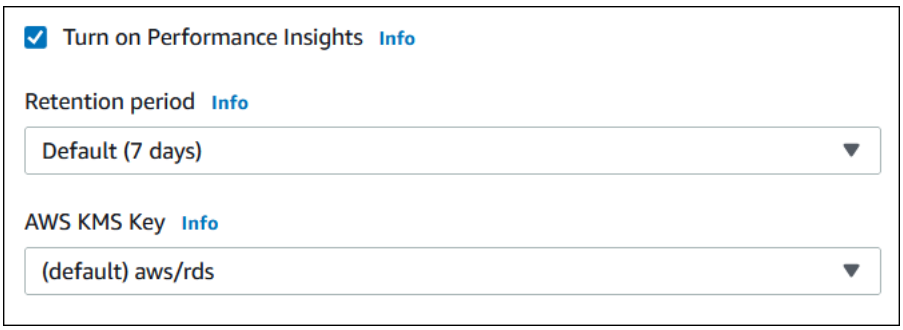
Konsol

Di konsol, Anda dapat mengaktifkan atau menonaktifkan Wawasan Performa saat membuat klaster DB. Anda dapat mengubah instans DB di klaster untuk mengaktifkan atau menonaktifkan Wawasan Performa untuk instans.

Mengaktifkan atau menonaktifkan Wawasan Performa saat membuat klaster DB

Saat Anda membuat klaster DB baru, aktifkan Wawasan Performa dengan memilih Aktifkan Wawasan Performa di bagian Wawasan Performa. Atau pilih Nonaktifkan Wawasan Performa. Untuk membuat klaster DB, ikuti petunjuk untuk mesin DB Anda di Membuat klaster DB Amazon Aurora.

Tangkapan layar berikut menunjukkan bagian Wawasan Performa.



Jika memilih Aktifkan Wawasan Performa, Anda akan memiliki opsi berikut:

- Retensi – Jumlah waktu untuk mempertahankan data Wawasan Performa. Pengaturan retensi di tingkat gratis adalah Default (7 hari). Untuk mempertahankan data kinerja Anda lebih lama, tetapkan 1–24 bulan. Untuk informasi selengkapnya tentang periode retensi, lihat Harga dan retensi data untuk Wawasan Performa.
- AWS KMS key – Tentukan AWS KMS key Anda. Wawasan Performa mengenkripsi semua data yang berpotensi sensitif menggunakan kunci KMS Anda. Data dienkripsi saat dipindahkan dan saat tidak aktif. Untuk informasi selengkapnya, lihat Mengonfigurasi kebijakan AWS KMS untuk Wawasan Performa.

Mengaktifkan atau menonaktifkan Wawasan Performa saat memodifikasi instans DB di klaster DB

Di konsol, Anda dapat memodifikasi instans DB di klaster DB untuk mengaktifkan atau menonaktifkan Wawasan Performa. Anda tidak dapat mengaktifkan atau menonaktifkan Wawasan Performa di tingkat klaster: Anda harus melakukannya untuk setiap instans di klaster.

Untuk mengaktifkan atau menonaktifkan Wawasan Performa untuk instans DB di klaster DB Anda menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih instans DB , dan pilih Modifikasi.
4. Di bagian Wawasan Performa, pilih Aktifkan Wawasan Performa atau Nonaktifkan Wawasan Performa.

   Jika memilih Aktifkan Wawasan Performa, Anda akan memiliki opsi berikut:

- Retensi – Jumlah waktu untuk mempertahankan data Wawasan Performa. Pengaturan retensi di tingkat gratis adalah Default (7 hari). Untuk mempertahankan data kinerja Anda lebih lama, tetapkan 1–24 bulan. Untuk informasi selengkapnya tentang periode retensi, lihat Harga dan retensi data untuk Wawasan Performa.
- AWS KMS key – Tentukan kunci KMS Anda. Wawasan Performa mengenkripsi semua data yang berpotensi sensitif menggunakan kunci KMS Anda. Data dienkripsi saat dipindahkan dan saat tidak aktif. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.

5. Pilih Lanjutkan.
6. Untuk Penjadwalan Modifikasi, pilih Terapkan langsung. Jika Anda memilih Terapkan selama periode pemeliharaan terjadwal berikutnya, instans Anda akan mengabaikan pengaturan ini dan segera mengaktifkan Wawasan Performa.
7. Pilih Modifikasi instans.

AWS CLI

Saat Anda menggunakan create-db-instanceAWS CLIperintah, aktifkan Performance Insights dengan menentukan. `--enable-performance-insights` Atau nonaktifkan Wawasan Performa dengan menentukan `--no-enable-performance-insights`.

Anda juga dapat menentukan nilai ini menggunakan perintah AWS CLI berikut:

- create-db-instance-read-replika
- modify-db-instance
- restore-db-instance-from-s3

Prosedur berikut menjelaskan cara mengaktifkan atau menonaktifkan Wawasan Performa untuk instans DB yang ada di klaster DB Anda menggunakan AWS CLI.

Untuk mengaktifkan atau menonaktifkan Wawasan Performa untuk instans DB di klaster DB Anda menggunakan AWS CLI

- Panggil modify-db-instanceAWS CLIperintah dan berikan nilai-nilai berikut:
  - `--db-instance-identifier` – Nama instans DB di klaster DB Anda.

- `--enable-performance-insights` untuk mengaktifkan atau `--no-enable-performance-insights` untuk menonaktifkan

Contoh berikut mengaktifkan Wawasan Performa untuk `sample-db-instance`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier sample-db-instance \
    --enable-performance-insights
```

Untuk Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier sample-db-instance ^
    --enable-performance-insights
```

Saat mengaktifkan Wawasan Performa di CLI, Anda dapat secara opsional menentukan jumlah hari untuk mempertahankan data Wawasan Performa dengan opsi `--performance-insights-retention-period`. Anda dapat menentukan 7, *`month`* * 31 (di mana *`month`* adalah jumlah dari 1-23), atau `731`. Misalnya, jika Anda ingin mempertahankan data performa selama 3 bulan, tentukan 93, yakni 3 * 31. Nilai default-nya adalah 7 hari. Untuk informasi selengkapnya tentang periode retensi, lihat Harga dan retensi data untuk Wawasan Performa.

Contoh berikut mengaktifkan Wawasan Performa untuk `sample-db-instance` dan menentukan bahwa data Wawasan Performa dipertahankan selama 93 hari (3 bulan).

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier sample-db-instance \
    --enable-performance-insights \
    --performance-insights-retention-period 93
```

Untuk Windows:

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

```
    --enable-performance-insights ^
    --performance-insights-retention-period 93
```

Jika Anda menentukan periode retensi seperti 94 hari, yang bukan merupakan nilai yang valid, RDS akan mengeluarkan kesalahan.

```
An error occurred (InvalidParameterValue) when calling the CreateDBInstance operation:
Invalid Performance Insights retention period. Valid values are: [7, 31, 62, 93, 124,
 155, 186, 217,
248, 279, 310, 341, 372, 403, 434, 465, 496, 527, 558, 589, 620, 651, 682, 713, 731]
```

API RDS

Saat Anda membuat instans DB baru di klaster DB Anda menggunakan operasi Amazon RDS API operasi CreateDBInstance, aktifkan Wawasan Performa dengan mengatur ke `EnablePerformanceInsights` ke `True`. Untuk menonaktifkan Wawasan Performa, atur `EnablePerformanceInsights` ke `False`.

Anda juga dapat menentukan nilai `EnablePerformanceInsights` menggunakan operasi API berikut:

- ModifyDBInstance
- dibuatB InstanceReadReplica
- DirestoredB S3 InstanceFrom

Saat mengaktifkan Wawasan Performa, Anda dapat secara opsional menentukan jumlah waktu, dalam hari, untuk mempertahankan data Wawasan Performa dengan parameter `PerformanceInsightsRetentionPeriod`. Anda dapat menentukan 7, *`month`* * 31 (di mana *`month`* adalah jumlah dari 1-23), atau 731. Misalnya, jika Anda ingin mempertahankan data performa selama 3 bulan, tentukan 93, yakni 3 * 31. Nilai default-nya adalah 7 hari. Untuk informasi selengkapnya tentang periode retensi, lihat Harga dan retensi data untuk Wawasan Performa.

Mengaktifkan Skema Performa untuk Wawasan Performa di Aurora MySQL

Skema Performa adalah fitur opsional untuk memantau performa runtime Aurora MySQL dengan tingkat detail rendah. Skema Performa dirancang untuk memiliki dampak minimal terhadap performa basis data. Wawasan Performa adalah fitur terpisah yang dapat digunakan dengan atau tanpa Skema Performa.

Topik

- Ringkasan Skema Performa
- Wawasan Performa dan Skema Performa
- Manajemen Skema Performa otomatis berdasarkan Wawasan Performa
- Pengaruh reboot pada Skema Performa
- Menentukan apakah Wawasan Performa mengelola Skema Performa
- Mengkonfigurasi Skema Performa untuk manajemen otomatis

Ringkasan Skema Performa

Skema Performa memantau peristiwa dalam basis data Aurora MySQL. Peristiwa adalah tindakan server basis data yang memakan waktu dan telah diinstrumentasi sehingga informasi waktu dapat dikumpulkan. Contoh peristiwa antara lain:

- Panggilan fungsi
- Peristiwa tunggu untuk sistem operasi
- Tahapan eksekusi SQL
- Grup pernyataan SQL

Mesin penyimpanan `PERFORMANCE_SCHEMA` adalah mekanisme untuk mengimplementasikan fitur Skema Performa. Mesin ini mengumpulkan data peristiwa menggunakan instrumentasi dalam kode sumber basis data. Mesin menyimpan peristiwa dalam tabel hanya memori di basis data `performance_schema`. Anda dapat mengkueri `performance_schema` sama seperti Anda mengkueri tabel lainnya. Untuk informasi selengkapnya, lihat Skema Performa MySQL di Panduan Referensi MySQL.

Wawasan Performa dan Skema Performa

Wawasan Performa dan Skema Performa adalah fitur terpisah, tetapi terhubung. Perilaku Wawasan Performa untuk Aurora MySQL bergantung pada apakah Skema Performa diaktifkan, dan jika demikian, apakah Wawasan Performa mengelola Skema Performa secara otomatis. Tabel berikut menjelaskan perilaku tersebut.

Skema Performa diaktifkan	Mode manajemen Wawasan Performa	Perilaku Wawasan Performa
Ya	Otomatis	• Mengumpulkan informasi pemantauan tingkat rendah yang mendetail • Mengumpulkan metrik sesi aktif setiap detik • Menampilkan muatan DB yang dikategorikan berdasarkan peristiwa tunggu mendetail, yang dapat digunakan untuk mengidentifikasi kemacetan
Ya	Manual	• Mengumpulkan peristiwa tunggu dan metrik per SQL • Mengumpulkan metrik sesi aktif setiap lima detik, bukan setiap detik • Melaporkan status pengguna seperti memasukkan dan mengirim, yang tidak membantu Anda mengidentifikasi kemacetan
Tidak	N/A	• Tidak mengumpulkan peristiwa tunggu, metrik per SQL, atau informasi pemantauan tingkat rendah mendetail lainnya • Mengumpulkan metrik sesi aktif setiap lima detik, bukan setiap detik • Melaporkan status pengguna seperti memasukkan dan mengirim, yang tidak membantu Anda mengidentifikasi kemacetan

Manajemen Skema Performa otomatis berdasarkan Wawasan Performa

Ketika Anda membuat instans DB Aurora MySQL dengan Wawasan Performa diaktifkan, Skema Performa juga diaktifkan. Dalam kasus ini, Wawasan Performa secara otomatis mengelola parameter Skema Performa Anda. Ini adalah konfigurasi yang disarankan.

> **Note**
>
> Manajemen Skema Performa otomatis tidak didukung untuk kelas instans t4g.medium.

Agar Wawasan Performa dapat mengelola Skema Performa secara otomatis, skema `performance_schema` harus diatur ke `0`. Secara default, nilai Sumber adalah `system`.

Anda juga dapat mengelola Skema Performa secara manual. Jika Anda memilih opsi ini, atur parameter sesuai dengan nilai dalam tabel berikut.

Nama parameter	Nilai parameter
`performance_schema`	1 (Kolom Sumber memiliki nilai `system`)
`performance-schema-consumer-events-waits-current`	`ON`
`performance-schema-instrument`	`wait/%=ON`
`performance_schema_consumer_global_instrumentation`	`1`
`performance_schema_consumer_thread_instrumentation`	`1`

Jika Anda mengubah nilai parameter `performance_schema` secara manual, dan kemudian ingin mengubah ke manajemen otomatis di lain waktu, lihat Mengkonfigurasi Skema Performa untuk manajemen otomatis.

> **Important**
>
> Saat Wawasan Performa mengaktifkan Skema Performa, nilai grup parameter tidak akan diubah. Namun, nilainya diubah pada instans DB yang sedang berjalan. Satu-satunya cara untuk melihat nilai yang diubah adalah dengan menjalankan perintah `SHOW GLOBAL VARIABLES`.

Pengaruh reboot pada Skema Performa

Wawasan Performa dan Skema Performa berbeda dalam persyaratannya untuk reboot instans DB:

Skema Performa

Untuk mengaktifkan atau menonaktifkan fitur ini, Anda harus me-reboot instans DB.

Wawasan Performa

Untuk mengaktifkan atau menonaktifkan fitur ini, Anda tidak harus me-reboot instans DB.

Jika Skema Performa saat ini tidak diaktifkan, dan Anda mengaktifkan Wawasan Performa tanpa me-reboot instans DB, Skema Performa tidak akan diaktifkan.

Menentukan apakah Wawasan Performa mengelola Skema Performa

Untuk mengetahui apakah Wawasan Performa saat ini mengelola Skema Performa untuk mesin utama versi 5.6, 5.7, dan 8.0, tinjau tabel berikut ini.

Pengaturan parameter performance_schema	Pengaturan kolom Source	Wawasan Performa mengelola Skema Performa?
`0`	`system`	Ya
`0` atau `1`	`user`	Tidak

Untuk menentukan apakah Wawasan Performa mengelola Skema Performa secara otomatis

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

2. Pilih Grup parameter.
3. Pilih nama grup parameter untuk instans DB Anda.
4. Masukkan **`performance_schema`** ke bilah pencarian.
5. Periksa apakah Sumber adalah default sistem dan Nilai adalah 0. Jika demikian, Wawasan Performa mengelola Skema Performa secara otomatis. Jika tidak, Wawasan Performa tidak mengelola Skema Performa secara otomatis.

Mengkonfigurasi Skema Performa untuk manajemen otomatis

Asumsikan bahwa Wawasan Performa diaktifkan untuk instans DB, tetapi saat ini tidak mengelola Wawasan Performa. Jika Anda ingin mengizinkan Wawasan Performa mengelola Skema Performa secara otomatis, selesaikan langkah-langkah berikut.

Untuk mengonfigurasi Skema Performa untuk manajemen otomatis

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Grup parameter.
3. Pilih nama grup parameter untuk instans DB.
4. Masukkan **`performance_schema`** ke bilah pencarian.
5. Pilih parameter `performance_schema`.
6. Pilih Edit parameter.
7. Pilih parameter `performance_schema`.
8. Di Nilai, pilih 0.
9. Pilih Atur ulang, Atur ulang parameter.
10. Reboot instans DB.

> **Important**
>
> Setiap kali Anda mengaktifkan atau menonaktifkan Skema Performa, pastikan untuk me-reboot instans DB.

Untuk informasi tentang cara mengubah parameter instans, lihat Memodifikasi parameter dalam grup parameter DB. Untuk informasi selengkapnya tentang dasbor, lihat Menganalisis metrik dengan dasbor Wawasan Performa. Untuk informasi selengkapnya tentang skema performa MySQL, lihat Panduan Referensi MySQL 8.0.

Mengonfigurasi kebijakan akses untuk Wawasan Performa

Untuk mengakses Wawasan Performa, pengguna utama harus memiliki izin yang sesuai dari AWS Identity and Access Management (IAM). Anda dapat memberikan akses dengan cara berikut:

- Melampirkan kebijakan `AmazonRDSPerformanceInsightsReadOnly` terkelola ke kumpulan izin atau peran untuk mengakses semua operasi hanya-baca dari API Wawasan Performa.
- Melampirkan kebijakan `AmazonRDSPerformanceInsightsFullAccess` terkelola ke kumpulan izin atau peran untuk mengakses semua operasi API Wawasan Performa.
- Membuat kebijakan IAM khusus dan melampirkannya ke kumpulan izin atau peran.

Jika Anda menentukan kunci yang dikelola pelanggan saat mengaktifkan Wawasan Performa, pastikan pengguna di akun Anda memiliki izin `kms:Decrypt` dan `kms:GenerateDataKey` izin pada kunci KMS.

Melampirkan kebijakan AmazonRDSPerformanceInsightsReadOnly ke pengguna utama IAM

`AmazonRDSPerformanceInsightsReadOnly` adalah kebijakan yang dikelola AWS yang memberikan akses ke semua operasi hanya-baca API Wawasan Performa Amazon RDS.

Jika Anda melampirkan `AmazonRDSPerformanceInsightsReadOnly` ke kumpulan izin atau peran, penerima dapat menggunakan Wawasan Performa beserta fitur konsol lainnya.

Untuk informasi selengkapnya, lihat AWS kebijakan terkelola: AmazonRDS PerformanceInsightsReadOnly.

Melampirkan kebijakan AmazonRDSPerformanceInsightsFullAccess ke pengguna utama IAM

`AmazonRDSPerformanceInsightsFullAccess` adalah kebijakan yang dikelola AWS yang memberikan akses ke semua operasi API Wawasan Performa Amazon RDS.

Jika Anda melampirkan `AmazonRDSPerformanceInsightsFullAccess` ke kumpulan izin atau peran, penerima dapat menggunakan Wawasan Performa beserta fitur konsol lainnya.

Untuk informasi selengkapnya, lihat AWS kebijakan terkelola: AmazonRDS PerformanceInsightsFullAccess.

Membuat kebijakan IAM khusus untuk Wawasan Performa

Bagi pengguna yang tidak memiliki kebijakan `AmazonRDSPerformanceInsightsReadOnly` atau `AmazonRDSPerformanceInsightsFullAccess`, Anda dapat memberikan akses ke Wawasan Performa dengan membuat atau memodifikasi kebijakan IAM yang dikelola pengguna. Jika Anda melampirkan kebijakan ini ke kumpulan izin atau peran IAM, penerima dapat menggunakan Wawasan Performa.

Untuk membuat kebijakan khusus

1. Buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Di panel navigasi, pilih Kebijakan.
3. Pilih Buat kebijakan.
4. Di halaman Buat Kebijakan, pilih tab JSON.
5. Salin dan tempel teks yang disediakan di bagian dokumen kebijakan JSON di Panduan Referensi Kebijakan yang Dikelola AWS untuk kebijakan AmazonRDSPerformanceInsightsReadOnly atau AmazonRDSPerformanceInsightsFullAccess.
6. Pilih Tinjau kebijakan.
7. Berikan nama untuk kebijakan tersebut dan secara opsional deskripsi, lalu pilih Buat kebijakan.

Sekarang, Anda dapat menyisipkan kebijakan ke kumpulan izin atau peran. Prosedur berikut mengasumsikan bahwa Anda sudah memiliki pengguna yang tersedia untuk tujuan ini.

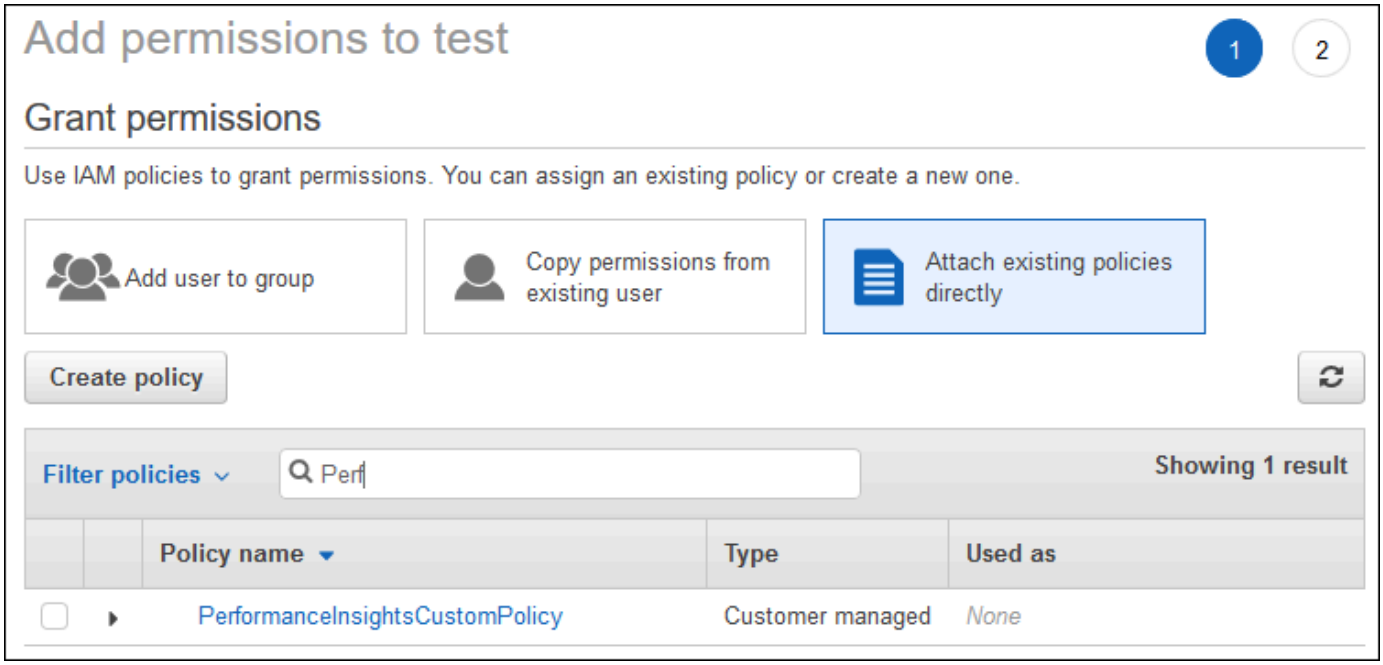
Untuk melampirkan kebijakan ini ke pengguna

1. Buka konsol IAM di https://console.aws.amazon.com/iam/.

2. Di panel navigasi, pilih Pengguna.
3. Pilih pengguna yang ada dari daftar.

> **Important**
>
> Untuk menggunakan Wawasan Performa, pastikan Anda memiliki akses ke Amazon RDS selain ke kebijakan khusus. Misalnya, kebijakan `AmazonRDSPerformanceInsightsReadOnly` yang ditentukan sebelumnya memberikan akses hanca-baca ke Amazon RDS. Untuk informasi selengkapnya, lihat Mengelola akses menggunakan kebijakan.

4. Di halaman Ringkasan, pilih Tambahkan izin.
5. Pilih Lampirkan kebijakan yang sudah ada secara langsung. Untuk Pencarian, ketik beberapa karakter pertama dari nama kebijakan Anda, seperti yang ditampilkan di bawah ini.



6. Pilih kebijakan Anda, lalu pilih Berikutnya: Tinjauan.
7. Pilih Tambahkan izin.

Mengonfigurasi kebijakan AWS KMS untuk Wawasan Performa

Wawasan Performa menggunakan AWS KMS key untuk mengenkripsi data sensitif. Saat mengaktifkan Wawasan Performa melalui API atau konsol, Anda dapat melakukan salah satu tindakan berikut:

- Memilih Kunci yang dikelola AWS default.

  Amazon RDS Kunci yang dikelola AWS untuk instans DB baru Anda. Amazon RDS membuat Kunci yang dikelola AWS untuk Akun AWS Anda. Akun AWS Anda memiliki Kunci yang dikelola AWS yang berbeda untuk Amazon RDS untuk masing-masing Wilayah AWS.
- Memilih kunci yang dikelola pelanggan.

  Jika Anda menentukan kunci yang dikelola pelanggan, pengguna di akun Anda yang memanggil API Wawasan Performa memerlukan izin `kms:Decrypt` dan `kms:GenerateDataKey` pada kunci KMS. Anda dapat mengkonfigurasi izin ini melalui kebijakan IAM. Namun, sebaiknya Anda mengelola izin ini melalui kebijakan kunci KMS Anda. Untuk informasi selengkapnya, lihat Menggunakan kebijakan kunci dalam KMS AWS.

Example

Contoh berikut menunjukkan cara menambahkan pernyataan ke kebijakan kunci KMS Anda. Pernyataan ini mengizinkan akses ke Wawasan Performa. Bergantung pada bagaimana Anda menggunakan kunci KMS, sebaiknya Anda mengubah beberapa pembatasan. Sebelum menambahkan pernyataan ke kebijakan, hapus semua komentar.

```
{
 "Version" : "2012-10-17",
 "Id" : "your-policy",
 "Statement" : [ {
    //This represents a statement that currently exists in your policy.
 }
 ....,
 //Starting here, add new statement to your policy for Performance Insights.
 //We recommend that you add one new statement for every RDS instance
{
    "Sid" : "Allow viewing RDS Performance Insights",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            //One or more principals allowed to access Performance Insights
            "arn:aws:iam::444455556666:role/Role1"
        ]
    },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
```

```
    ],
    "Resource": "*",
    "Condition" : {
        "StringEquals" : {
            //Restrict access to only RDS APIs (including Performance Insights).
            //Replace region with your AWS Region.
            //For example, specify us-west-2.
            "kms:ViaService" : "rds.region.amazonaws.com"
        },
        "ForAnyValue:StringEquals": {
            //Restrict access to only data encrypted by Performance Insights.
            "kms:EncryptionContext:aws:pi:service": "rds",
            "kms:EncryptionContext:service": "pi",

            //Restrict access to a specific RDS instance.
            //The value is a DbiResourceId.
           "kms:EncryptionContext:aws:rds:db-id": "db-AAAAABBBBBCCCCDDDDDEEEEE"
        }
    }
}
```

Bagaimana Wawasan Performa menggunakan kunci yang dikelola pelanggan AWS KMS

Wawasan Performa menggunakan kunci yang dikelola pelanggan untuk mengenkripsi data sensitif. Jika mengaktifkan Wawasan Performa, Anda dapat memberikan kunci AWS KMS melalui API. Wawasan Performa membuat izin KMS pada kunci ini. Ini menggunakan kunci dan melakukan operasi yang diperlukan untuk memproses data sensitif. Data sensitif mencakup kolom-kolom seperti pengguna, basis data, aplikasi, dan teks kueri SQL. Wawasan Performa memastikan bahwa data tetap terenkripsi baik saat tidak aktif maupun saat transit.

Cara kerja IAM Wawasan Performa dengan AWS KMS

IAM memberikan izin ke API tertentu. Wawasan Performa memiliki API publik berikut, yang dapat Anda batasi menggunakan kebijakan IAM:

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetadata`
- `GetResourceMetrics`
- `ListAvailableResourceDimensions`

- `ListAvailableResourceMetrics`

Anda dapat menggunakan permintaan API berikut untuk mendapatkan data sensitif.

- `DescribeDimensionKeys`
- `GetDimensionKeyDetails`
- `GetResourceMetrics`

Saat Anda menggunakan API untuk mendapatkan data sensitif, Wawasan Performa memanfaatkan kredensial pemanggil. Pemeriksaan ini memastikan bahwa akses ke data sensitif dibatasi pada mereka yang memiliki akses ke kunci KMS.

Saat memanggil API ini, Anda memerlukan izin untuk memanggil API melalui kebijakan IAM dan izin untuk menginvokasi tindakan `kms:decrypt` melalui kebijakan kunci AWS KMS.

API `GetResourceMetrics` dapat menampilkan data sensitif dan non-sensitif. Parameter permintaan menentukan apakah respons harus menyertakan data sensitif. API menampilkan data sensitif ketika permintaan menyertakan dimensi sensitif baik dalam parameter filter atau kelompokkan-menurut.

Untuk informasi selengkapnya tentang dimensi yang dapat digunakan dengan API `GetResourceMetrics`, lihat DimensionGroup.

Example Contoh

Contoh berikut meminta data sensitif untuk grup `db.user`:

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
  "ServiceType": "RDS",
```

```
  "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
  "MetricQueries": [
    {
      "Metric": "db.load.avg",
      "GroupBy": {
        "Group": "db.user",
        "Limit": 2
      }
    }
  ],
  "StartTime": 1693872000,
  "EndTime": 1694044800,
  "PeriodInSeconds": 86400
}
```

Example

Contoh berikut meminta data non-sensitif untuk metrik `db.load.avg`:

```
POST / HTTP/1.1
Host: <Hostname>
Accept-Encoding: identity
X-Amz-Target: PerformanceInsightsv20180227.GetResourceMetrics
Content-Type: application/x-amz-json-1.1
User-Agent: <UserAgentString>
X-Amz-Date: <Date>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>
{
    "ServiceType": "RDS",
    "Identifier": "db-ABC1DEFGHIJKL2MNOPQRSTUV3W",
    "MetricQueries": [
        {
            "Metric": "db.load.avg"
        }
    ],
    "StartTime": 1693872000,
    "EndTime": 1694044800,
    "PeriodInSeconds": 86400
}
```

Menganalisis metrik dengan dasbor Wawasan Performa

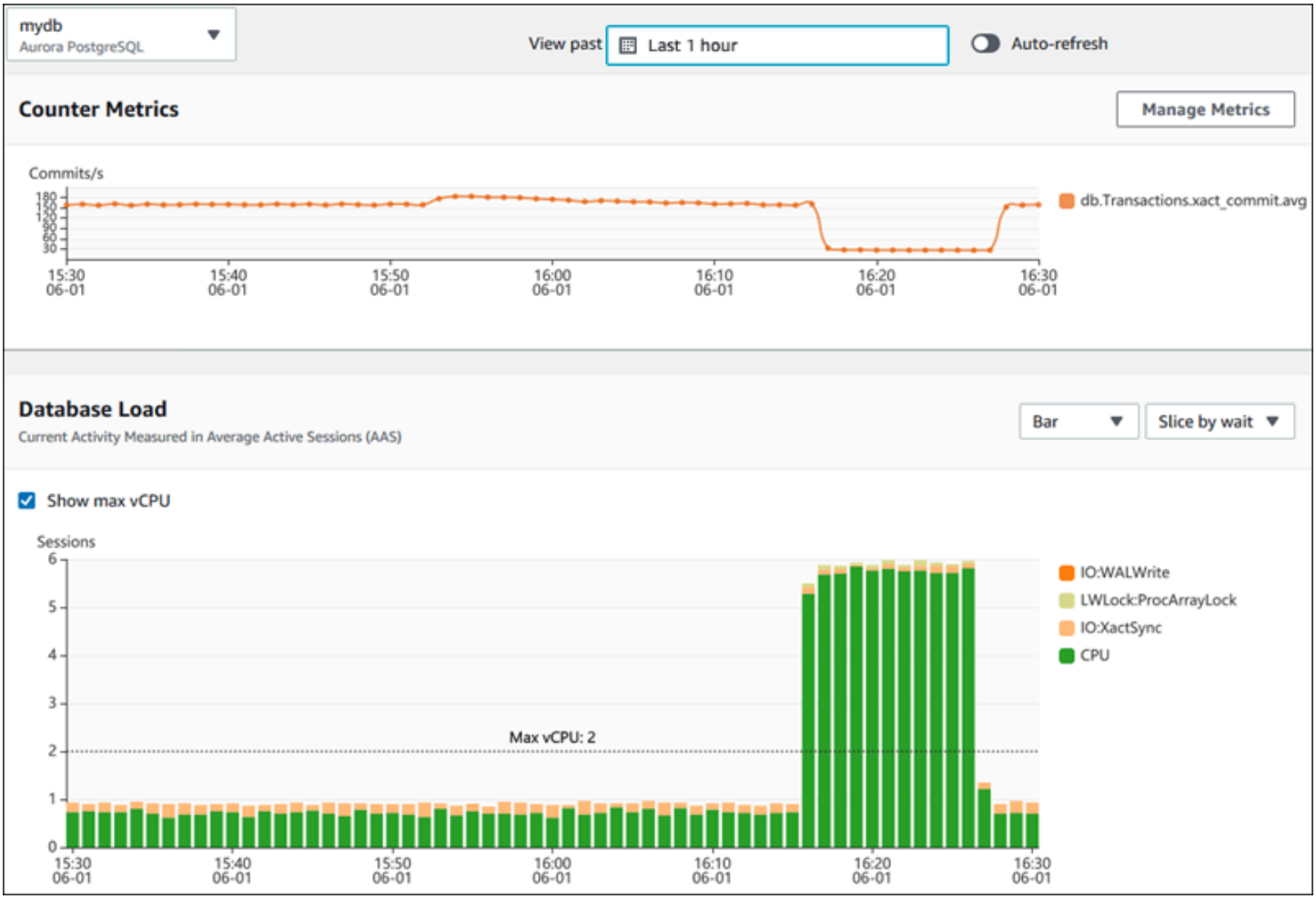
Dasbor Wawasan Performa berisi informasi performa basis data untuk membantu Anda menganalisis dan memecahkan masalah performa. Di halaman dasbor utama, Anda dapat melihat informasi tentang muatan basis data. Anda dapat “mengiris” muatan DB berdasarkan dimensi seperti peristiwa tunggu atau SQL.

Dasbor Wawasan Performa

- Ikhtisar dasbor Wawasan Performa
- Mengakses dasbor Wawasan Performa
- Menganalisis muatan DB menurut peristiwa tunggu
- Menganalisis performa basis data selama periode waktu tertentu
- Menganalisis kueri di dasbor Wawasan Performa

Ikhtisar dasbor Wawasan Performa

Dasbor adalah cara termudah untuk berinteraksi dengan Wawasan Performa. Contoh berikut menunjukkan dasbor untuk instans DB MySQL.



Topik

- Filter rentang waktu
- Bagan metrik penghitung
- Bagan muatan basis data
- Tabel Dimensi teratas

Filter rentang waktu

Secara default, dasbor Wawasan Performa menampilkan muatan DB selama satu jam terakhir. Anda dapat menyesuaikan rentang ini menjadi sesingkat 5 menit atau selama 2 tahun. Anda juga dapat memilih rentang relatif kustom.

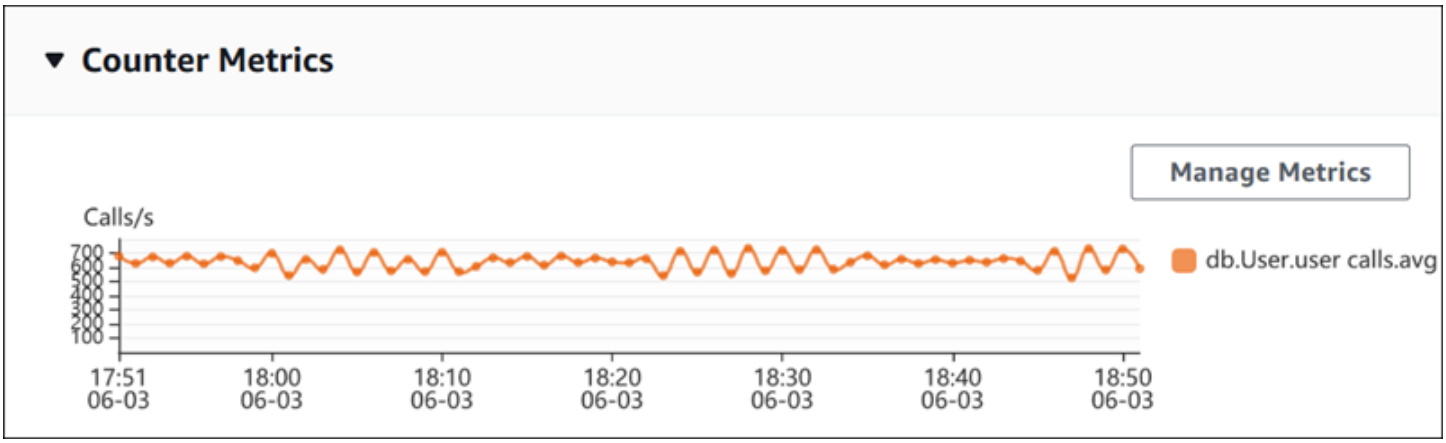
Anda dapat memilih rentang absolut dengan tanggal dan waktu awal dan akhir. Contoh berikut menunjukkan rentang waktu yang dimulai tengah malam pada 11/4/22 dan berakhir pukul 23.59 pada 14/4/22.

Bagan metrik penghitung

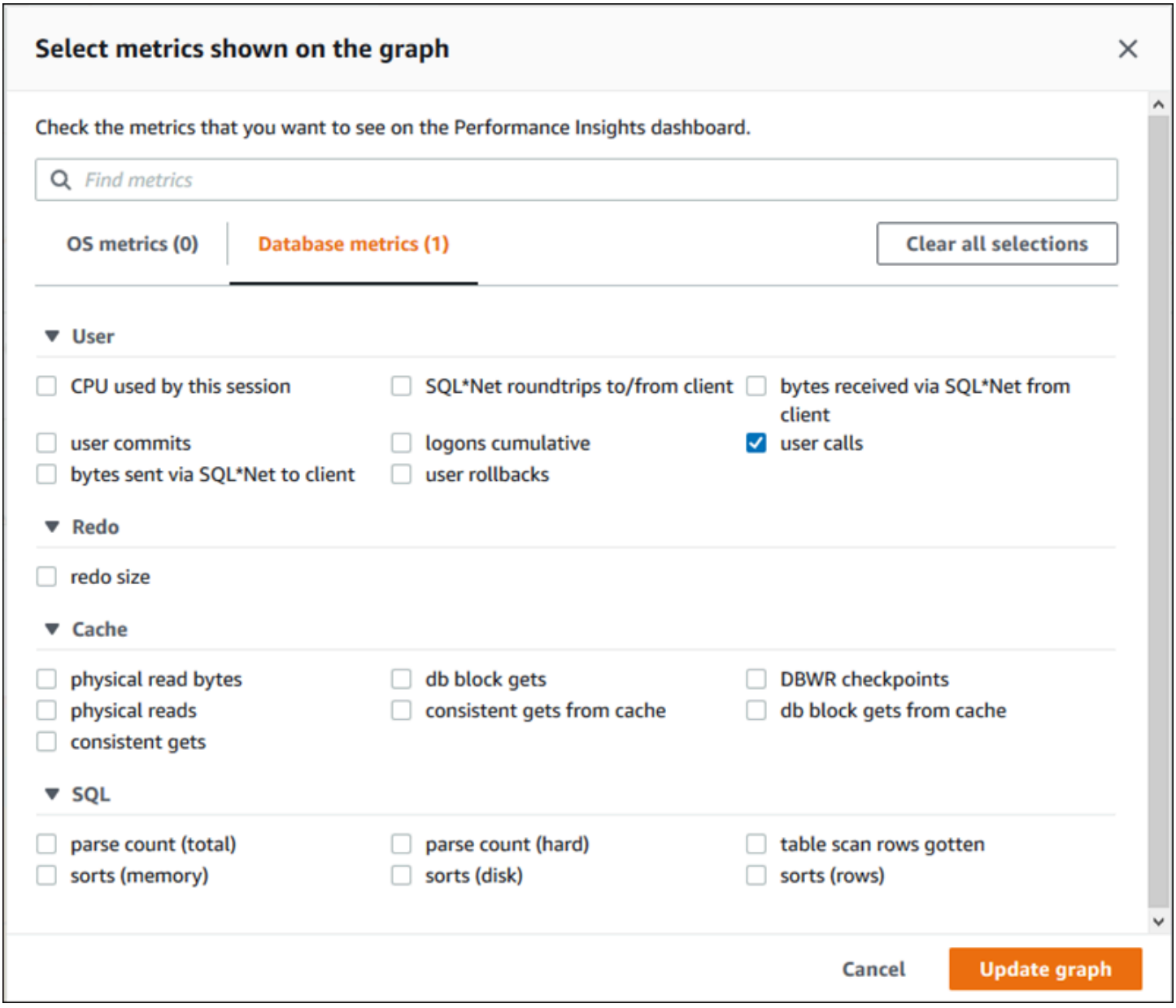
Dengan metrik penghitung, Anda dapat menyesuaikan dasbor Wawasan Performa untuk menyertakan hingga 10 grafik tambahan. Grafik ini menunjukkan pilihan dari sejumlah sistem operasi dan metrik performa basis data. Anda dapat menghubungkan informasi ini dengan muatan DB untuk membantu mengidentifikasi dan menganalisis masalah performa.

Bagan Metrik Penghitung menampilkan data untuk penghitung performa. Metrik default bergantung pada mesin DB:

- Aurora MySQL– `db.SQL.Innodb_rows_read.avg`
- Aurora PostgreSQL – `db.Transactions.xact_commit.avg`



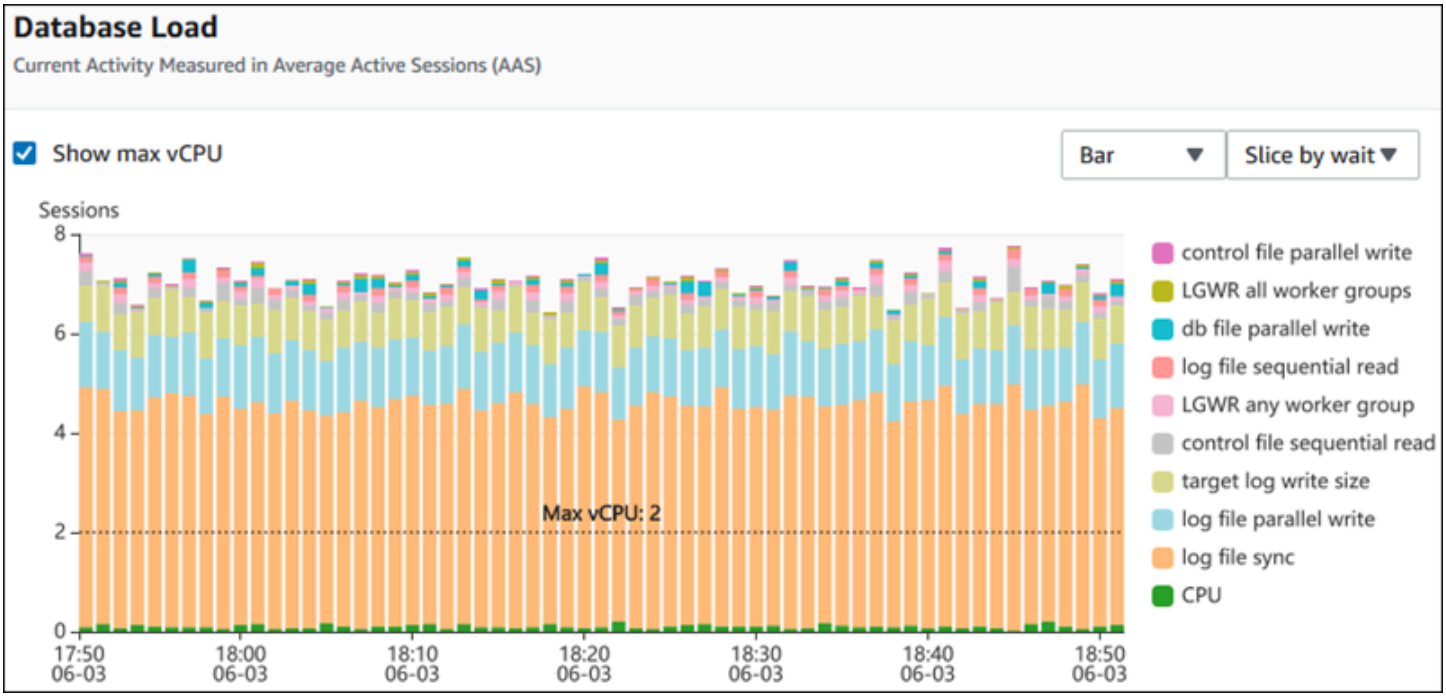
Untuk mengubah penghitung performa, pilih Kelola Metrik. Anda dapat memilih beberapa Metrik OS atau Metrik basis data, seperti yang ditunjukkan di tangkapan layar berikut. Untuk melihat detail setiap metrik, arahkan kursor ke nama metrik.



Untuk deskripsi metrik penghitung yang dapat ditambahkan untuk setiap mesin DB, lihat Metrik penghitung Wawasan Performa.

Bagan muatan basis data

Bagan Muatan basis data menunjukkan perbandingan aktivitas basis data dengan kapasitas instans DB seperti yang ditunjukkan oleh baris vCPU Maks. Secara default, bagan garis bertumpuk mewakili muatan DB sebagai sesi aktif rata-rata per unit waktu. Muatan DB diiris (dikelompokkan) berdasarkan status tunggu.

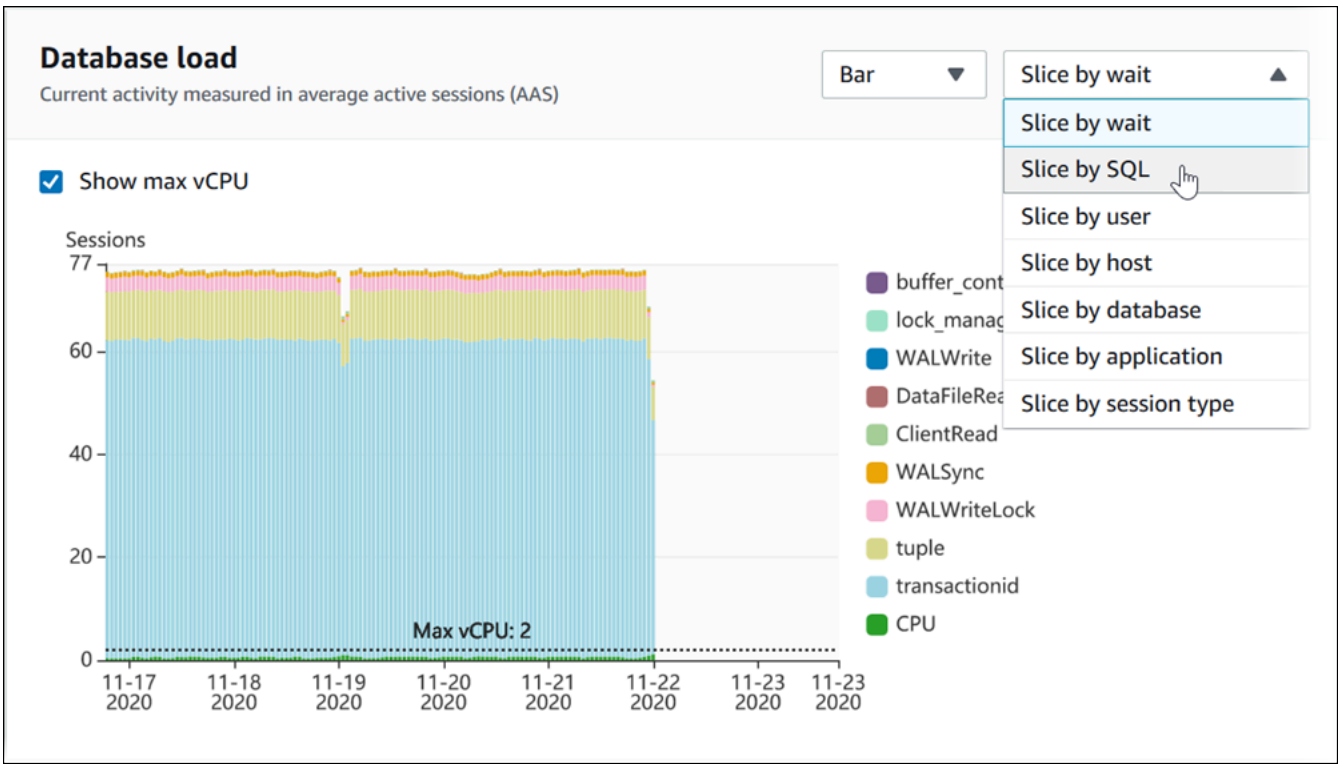


Muatan DB diiris berdasarkan dimensi

Anda dapat memilih untuk menampilkan muatan sebagai sesi aktif yang dikelompokkan berdasarkan dimensi yang didukung. Tabel berikut menunjukkan dimensi yang didukung untuk mesin yang berbeda.

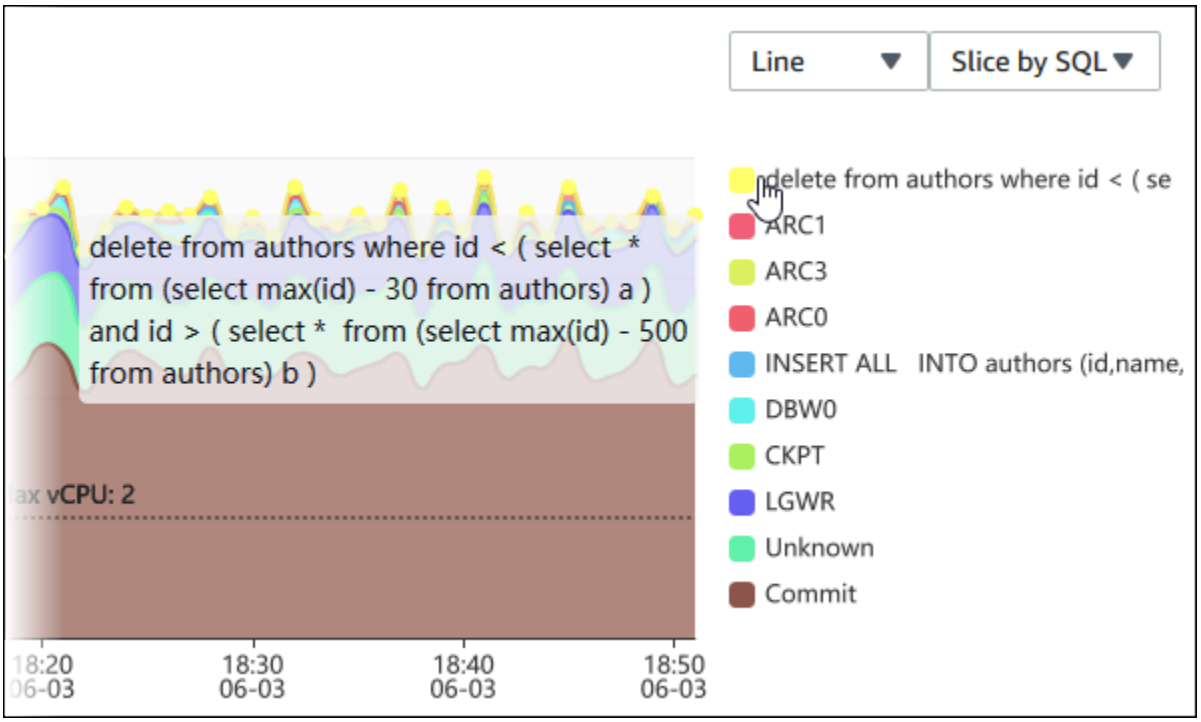
Dimensi	Aurora PostgreSQL	Aurora MySQL
Host	Ya	Ya
SQL	Ya	Ya
Pengguna	Ya	Ya
Tunggu	Ya	Ya
Aplikasi	Ya	Tidak
Basis data	Ya	Ya
Jenis sesi	Ya	Tidak

Gambar berikut menunjukkan dimensi untuk instans DB PostgreSQL.

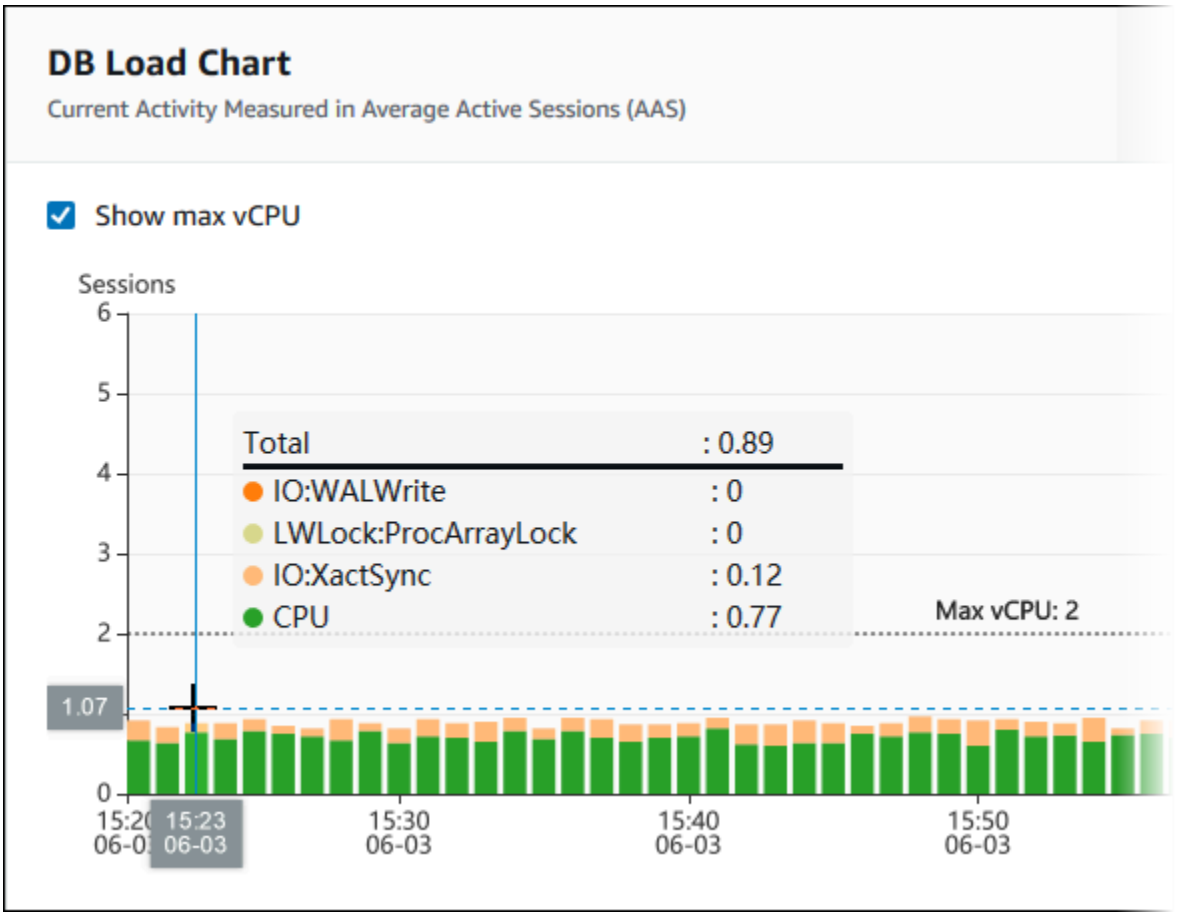


Detail muatan DB untuk item dimensi

Untuk melihat detail tentang item muatan DB dalam dimensi, arahkan kursor ke nama item. Gambar berikut menunjukkan rincian untuk pernyataan SQL.

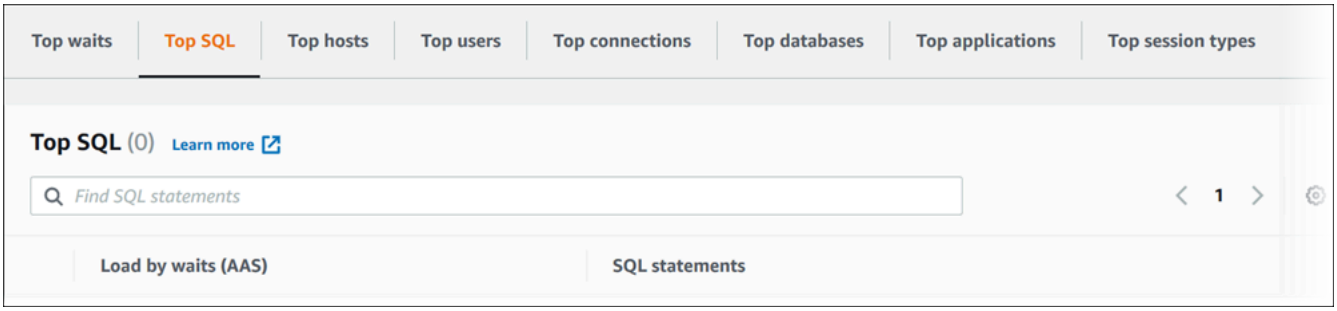


Untuk melihat detail setiap item selama periode waktu yang dipilih dalam legenda, arahkan kursor ke item tersebut.



Tabel Dimensi teratas

Tabel Dimensi teratas mengiris muatan DB dengan dimensi yang berbeda. Dimensi adalah kategori atau “irisan menurut” untuk karakteristik muatan DB yang berbeda. Jika dimensinya adalah SQL, SQL Teratas menunjukkan pernyataan SQL yang berkontribusi paling besar terhadap muatan DB.



Pilih salah satu tab dimensi berikut.

Tab	Deskripsi	Mesin yang didukung
SQL Teratas	Pernyataan SQL yang saat ini sedang berjalan	Semua
Tunggu teratas	Peristiwa di mana backend basis data sedang menunggu	Semua
Host teratas	Nama host klien yang terhubung	Semua
Pengguna teratas	Pengguna yang masuk ke basis data	Semua
Nama basis data yang terhubung ke klien		
Aplikasi teratas	Nama aplikasi yang terhubung ke basis data	Khusus Aurora PostgreSQL
Jenis sesi teratas	Jenis sesi saat ini	Khusus Aurora PostgreSQL

Untuk mempelajari cara menganalisis kueri dengan menggunakan tab SQL Teratas, lihat Ikhtisar tab SQL Teratas.

Mengakses dasbor Wawasan Performa

Amazon RDS menyediakan tampilan Wawasan Performa dan metrik CloudWatch terpadu di dasbor Wawasan Performa.

Untuk mengakses dasbor Wawasan Performa, gunakan prosedur berikut.

Untuk melihat dasbor Wawasan Performa di Konsol Manajemen AWS

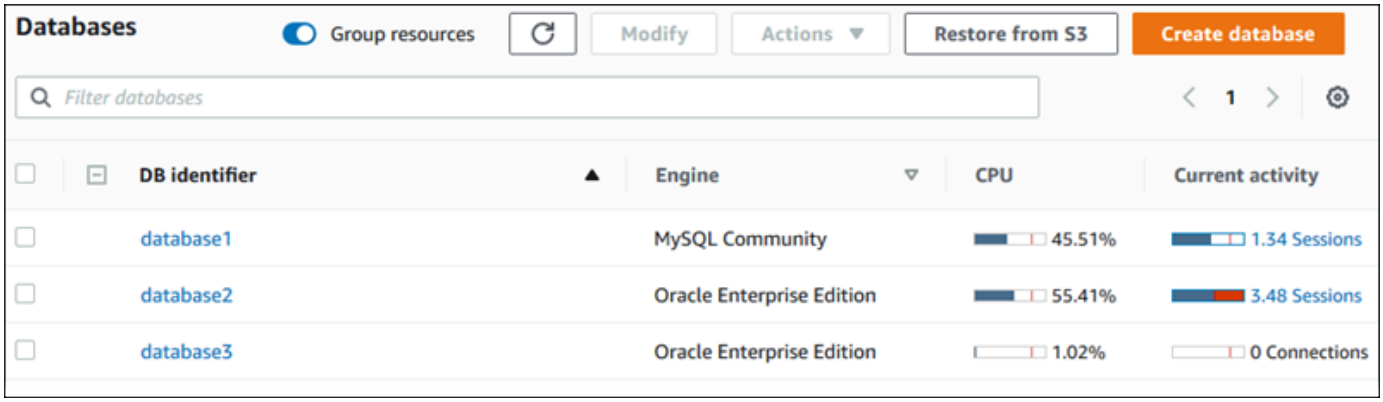
1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.
4. Pilih tampilan pemantauan default di jendela yang ditampilkan.

- Pilih opsi Tampilan Wawasan Performa dan metrik CloudWatch (Baru), lalu pilih Lanjutkan untuk melihat Wawasan Performa dan metrik CloudWatch.
- Pilih opsi Tampilan Wawasan Performa, lalu pilih Lanjutkan untuk tampilan pemantauan lama. Kemudian, lanjutkan dengan prosedur ini.

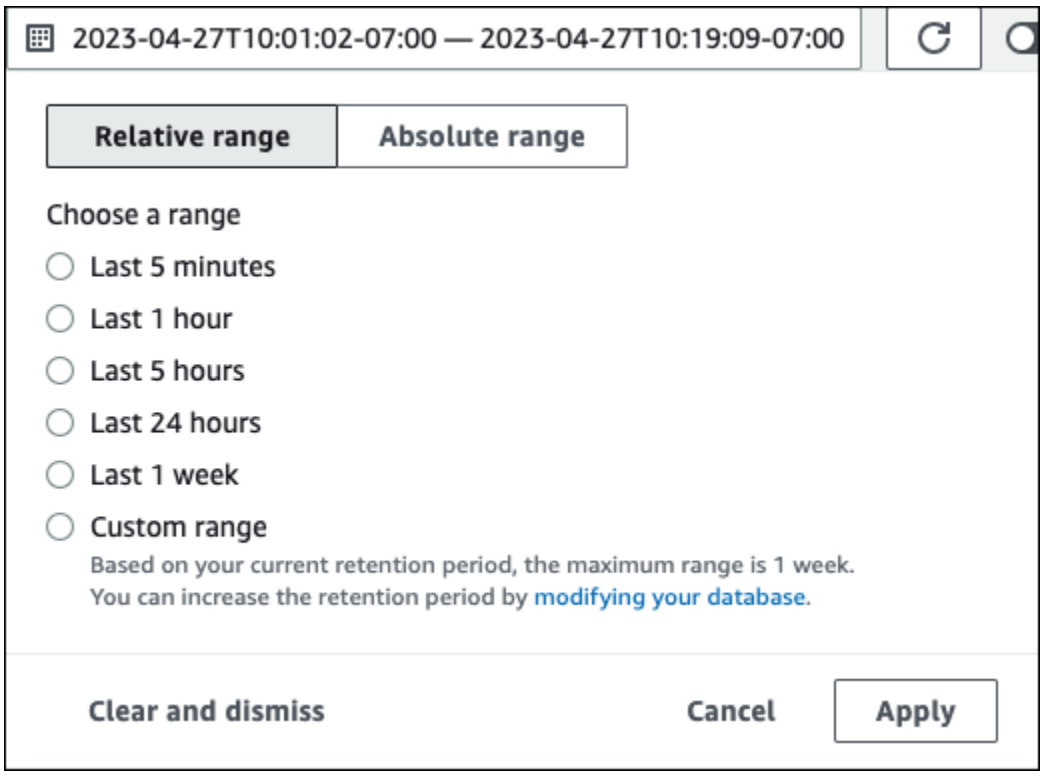
> **Note**
>
> Tampilan ini akan dihentikan pada tanggal 15 Desember 2023.

Dasbor Wawasan Performa muncul untuk instans DB.

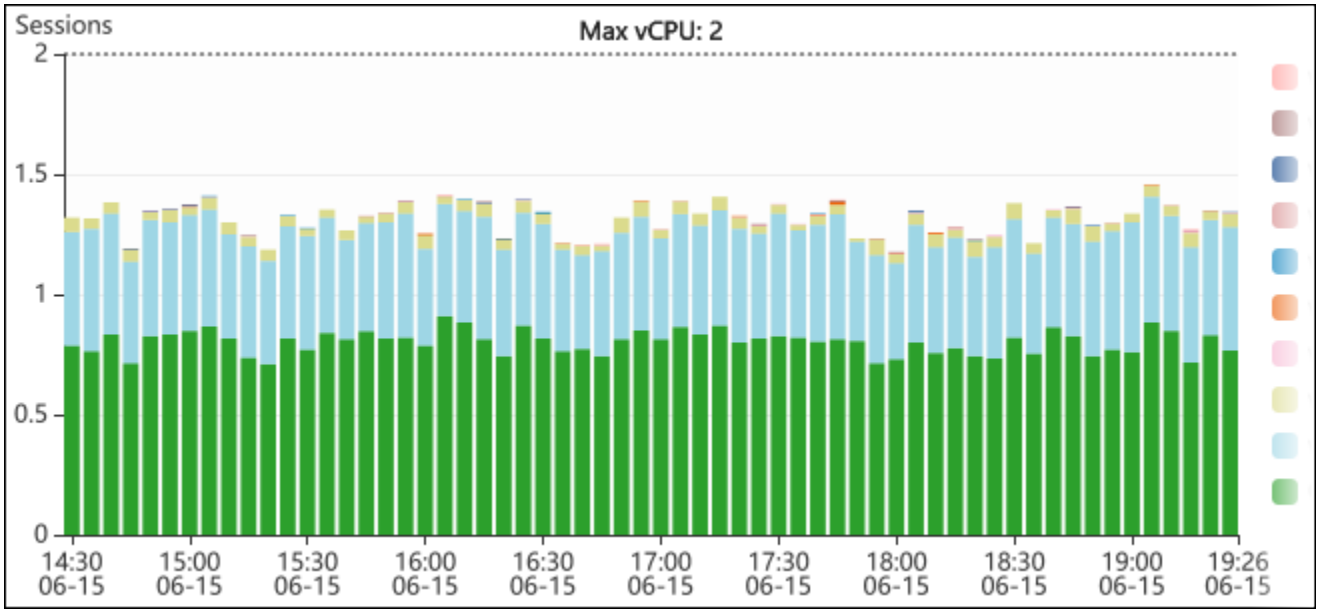
Untuk instans DB dengan Wawasan Performa yang diaktifkan, Anda juga dapat mengakses dasbor dengan memilih item Sesi di daftar instans DB. Di bagian Aktivitas saat ini, item Sesi menunjukkan muatan basis data dalam sesi aktif rata-rata selama lima menit terakhir. Bilah ini secara grafis menunjukkan muatan. Jika bilah kosong, berarti instans DB sedang diam. Saat muatan meningkat, bilah akan terisi dengan warna biru. Saat muatan melewati jumlah CPU virtual (vCPU) pada kelas instans DB, bilah akan berubah menjadi merah, yang menunjukkan adanya potensi kemacetan.



5. (Opsional) Pilih rentang tanggal atau waktu di kanan atas dan tentukan interval waktu relatif atau absolut yang berbeda. Anda kini dapat menentukan periode waktu, dan menghasilkan laporan analisis performa basis data. Laporan ini berisi rekomendasi dan wawasan yang diidentifikasi. Untuk informasi selengkapnya, lihat Membuat laporan analisis performa.

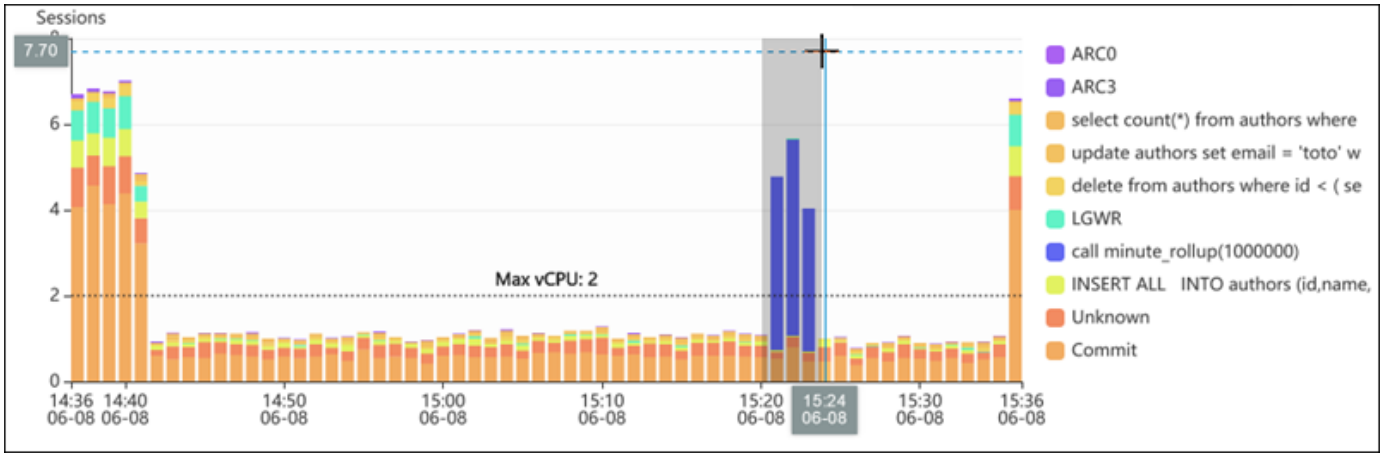


Di tangkapan layar berikut, interval muatan DB adalah 5 jam.

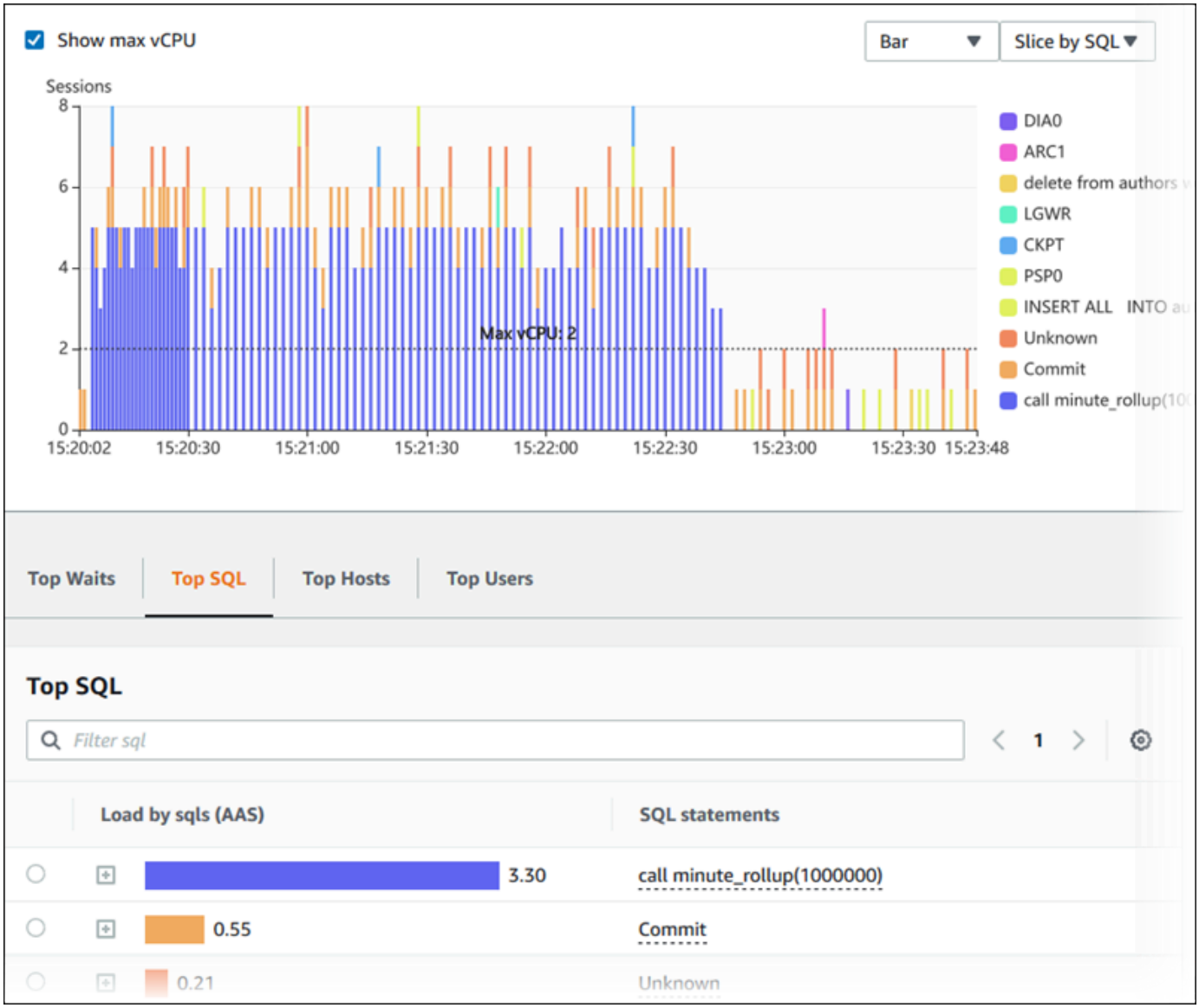


6. (Opsional) Untuk memperbesar sebagian bagan muatan DB, pilih waktu mulai dan seret ke akhir periode waktu yang diinginkan.

   Area yang dipilih disorot dalam bagan muatan DB.



Saat melepaskan mouse, bagan muatan DB akan diperbesar di Wilayah AWS yang dipilih, dan tabel Dimensi teratas dihitung ulang.



7. (Opsional) Untuk menyegarkan data Anda secara otomatis, pilih Segarkan otomatis.

Dasbor Wawasan Performa secara otomatis disegarkan dengan data baru. Laju penyegaran bergantung pada jumlah data yang ditampilkan:

- Penyegaran 5 menit setiap 10 detik.
- Penyegaran 1 jam setiap 5 menit.
- Penyegaran 5 jam setiap 5 menit.
- Penyegaran 24 jam setiap 30 menit.
- Penyegaran 1 minggu setiap hari.

- Penyegaran 1 bulan setiap hari.

Menganalisis muatan DB menurut peristiwa tunggu

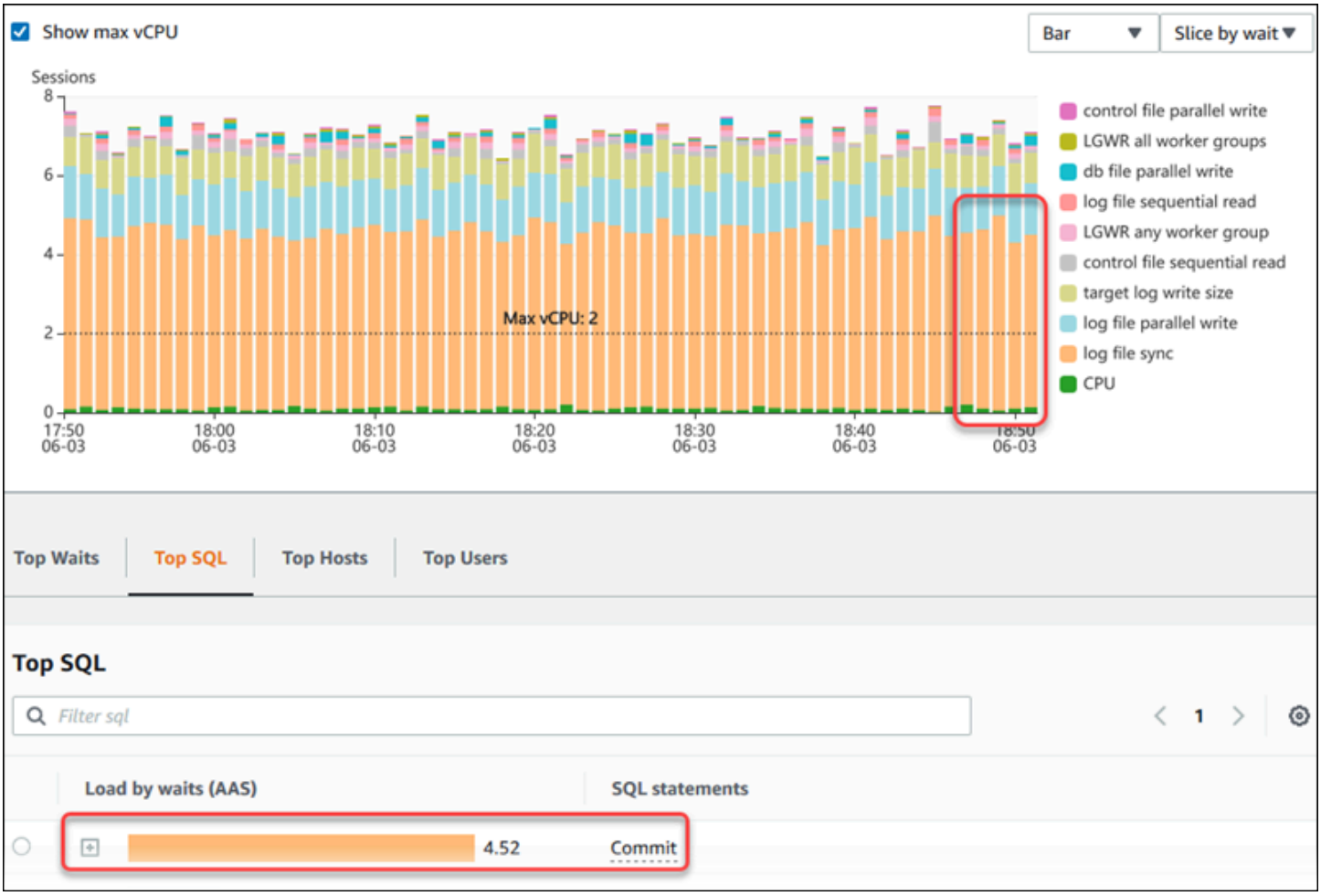
Jika bagan Muatan basis data menunjukkan kemacetan, Anda dapat mengetahui dari mana muatan tersebut berasal. Untuk melakukannya, lihat tabel item muatan teratas di bawah bagan Muatan basis data. Pilih item tertentu, seperti kueri SQL atau pengguna, untuk menelusuri item tersebut dan melihat detailnya.

Muatan DB yang dikelompokkan berdasarkan peristiwa tunggu dan kueri SQL teratas adalah tampilan default dasbor Wawasan Performa. Kombinasi ini biasanya memberikan wawasan paling banyak tentang masalah performa. Muatan DB yang dikelompokkan berdasarkan peristiwa tunggu menunjukkan apakah ada sumber daya atau kemacetan konkurensi dalam basis data. Dalam kasus ini, tab SQL dari tabel item muatan teratas menunjukkan kueri mana yang mendorong muatan tersebut.

Alur kerja tipikal Anda untuk mendiagnosis masalah performa adalah sebagai berikut:

1. Tinjau bagan Muatan basis data dan lihat apakah ada insiden muatan basis data yang melebihi baris CPU Maks.
2. Jika ada, lihat bagan Muatan basis data dan identifikasi satu atau beberapa status tunggu yang paling bertanggung jawab.
3. Identifikasi kueri digest yang menyebabkan muatan dengan melihat kueri mana dari tab SQL di tabel item muatan teratas yang berkontribusi paling besar pada status tunggu tersebut. Anda dapat mengidentifikasinya berdasarkan kolom Muatan DB berdasarkan peristiwa Tunggu.
4. Pilih salah satu kueri digest di tab SQL untuk meluaskannya dan melihat kueri turunan yang menyusunnya.

Misalnya, di dasbor berikut, peristiwa tunggu sinkronisasi file log menyumbang sebagian besar muatan DB. Peristiwa tunggu LGWR semua grup pekerja juga tinggi. Bagan SQL Teratas menunjukkan penyebab peristiwa tunggu sinkronisasi file log: pernyataan `COMMIT` yang sering. Dalam kasus ini, eksekusi yang lebih jarang akan mengurangi muatan DB.



Menganalisis performa basis data selama periode waktu tertentu

Anda dapat membuat laporan analisis performa selama periode waktu tertentu dan mengetahui masalah performa apa pun seperti kemacetan sumber daya atau perubahan dalam kueri di instans DB Anda. Dasbor Wawasan Performa memungkinkan Anda memilih periode waktu dan membuat laporan analisis performa. Anda juga dapat menambahkan satu tag atau lebih ke laporan.

Untuk menggunakan fitur ini, Anda harus menggunakan periode retensi tingkat berbayar. Untuk informasi selengkapnya, lihat Harga dan retensi data untuk Wawasan Performa

Laporan ini dapat dipilih dan dilihat di tab Laporan analisis performa - baru. Laporan ini berisi wawasan, metrik terkait, dan rekomendasi untuk menyelesaikan masalah performa. Laporan ini dapat dilihat selama periode retensi Wawasan Performa.

Laporan dihapus jika waktu mulai periode analisis laporan berada di luar periode retensi. Anda juga dapat menghapus laporan sebelum periode retensi berakhir.

Untuk mendeteksi masalah performa dan menghasilkan laporan analisis untuk instans DB, Anda harus mengaktifkan Wawasan Performa. Untuk informasi selengkapnya tentang mengaktifkan Wawasan Performa, lihat Mengaktifkan dan menonaktifkan Wawasan Performa.

Untuk informasi dukungan wilayah, mesin DB, dan kelas instans untuk fitur ini, lihat Dukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk fitur Wawasan Performa

Membuat laporan analisis performa

Anda dapat membuat laporan analisis performa selama periode tertentu di dasbor Wawasan Performa. Anda dapat memilih periode waktu dan menambahkan satu tag atau lebih ke laporan analisis.

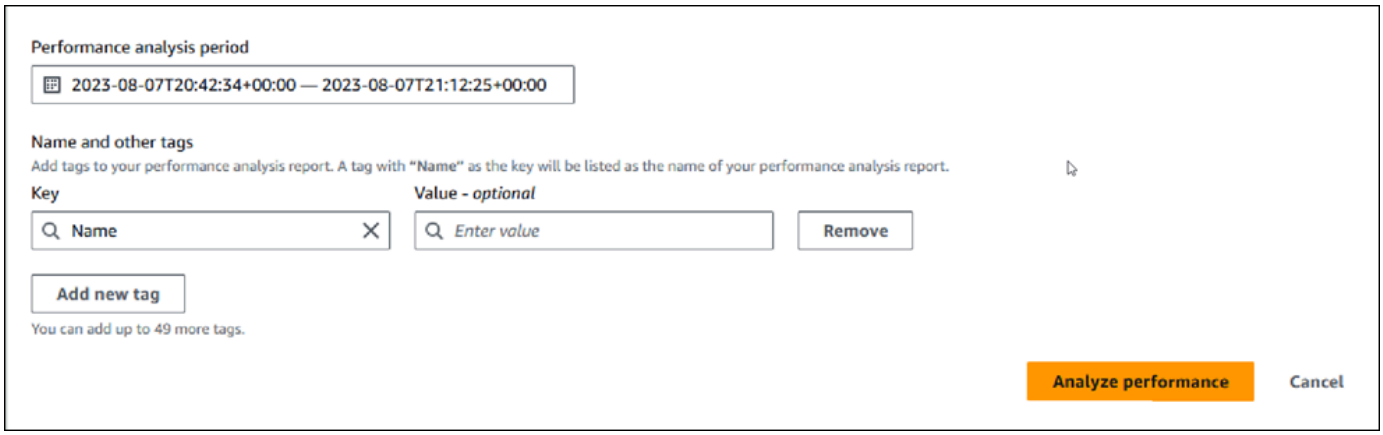
Periode analisis bisa berkisar dari 5 menit hingga 6 hari. Harus ada data performa setidaknya 24 jam sebelum waktu mulai analisis.

Untuk membuat laporan analisis performa selama periode waktu tertentu

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.

   Dasbor Wawasan Performa muncul untuk instans DB.
4. Pilih Analisis performa di bagian Muatan basis data di dasbor.

   Kolom untuk mengatur periode waktu dan menambahkan satu atau beberapa tag ke laporan analisis performa ditampilkan.



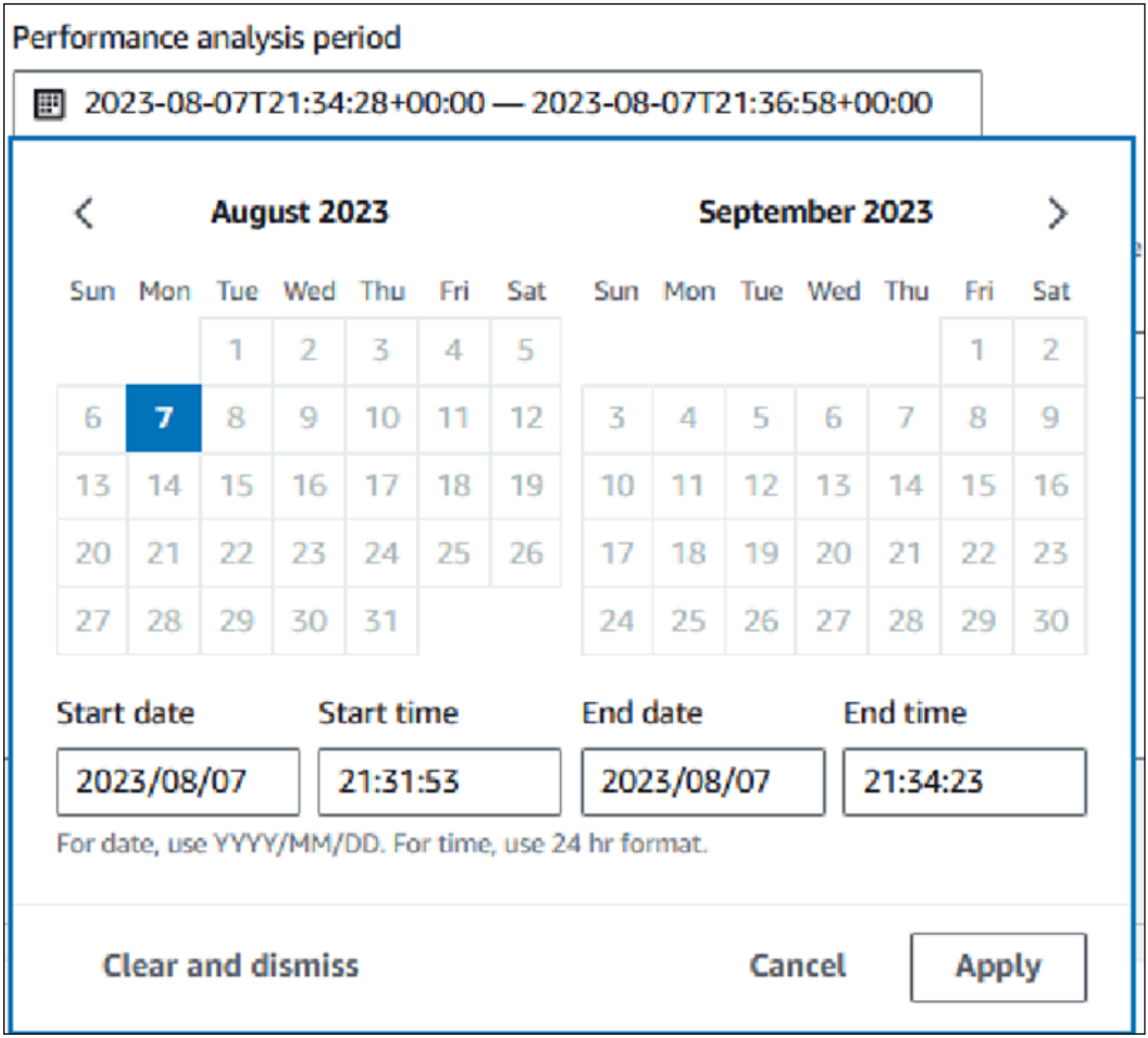
5. Pilih periode waktu. Jika menetapkan periode waktu dalam Rentang relatif atau Rentang absolut di kanan atas, Anda hanya dapat memasukkan atau memilih tanggal dan waktu laporan analisis

dalam periode waktu ini. Jika Anda memilih periode analisis di luar periode waktu ini, pesan kesalahan akan muncul.

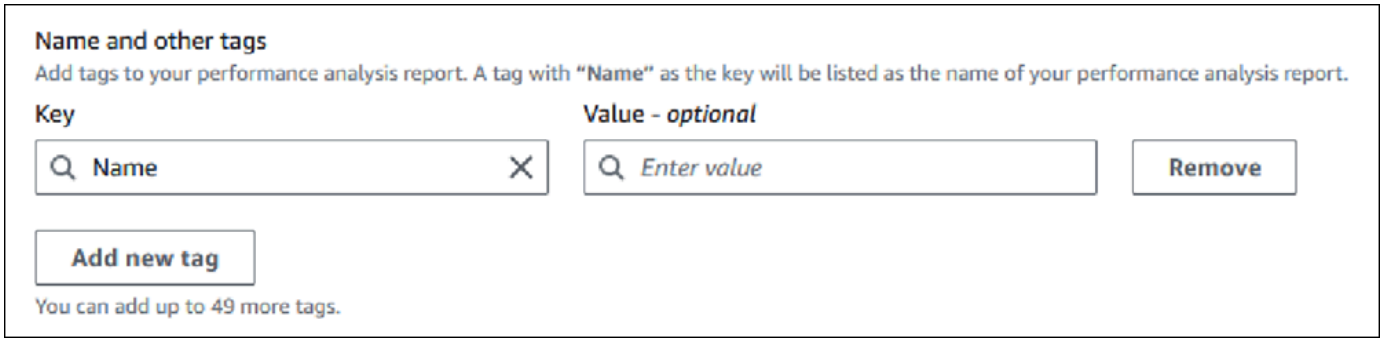
Untuk mengatur periode waktu, Anda dapat mengikuti langkah-langkah berikut:

- Tekan dan seret salah satu slider pada bagan muatan DB.

  Kotak Periode analisis performa menampilkan periode waktu yang dipilih dan bagan muatan DB menyoroti periode waktu yang dipilih.
- Pilih Tanggal mulai, Waktu mulai, Tanggal akhir, dan Waktu akhir di kotak Periode analisis performa.



6. (Opsional) Masukkan Kunci dan Nilai-opsional untuk menambahkan tag untuk laporan.



7. Pilih Analisis performa.

   Spanduk menampilkan pesan apakah pembuatan laporan berhasil atau gagal. Pesan juga menyediakan tautan untuk melihat laporan.

   Contoh berikut menunjukkan spanduk dengan pesan pembuatan laporan berhasil.

   Laporan ini dapat dilihat di tab Laporan analisis performa - baru.

Anda dapat membuat laporan analisis performa menggunakan AWS CLI. Untuk contoh tentang cara membuat laporan menggunakan AWS CLI, lihat Membuat laporan analisis performa selama periode waktu tertentu.

Melihat laporan analisis performa

Tab Laporan analisis performa - baru mencantumkan semua laporan yang dibuat untuk instans DB. Berikut ini ditampilkan untuk setiap laporan:

- ID: Pengidentifikasi unik laporan.
- Nama: Kunci tag yang ditambahkan ke laporan.
- Waktu pembuatan laporan: Waktu Anda membuat laporan.
- Waktu mulai analisis: Waktu mulai analisis dalam laporan.
- Waktu akhir analisis: Waktu akhir analisis dalam laporan.

Untuk melihat laporan analisis performa

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB yang laporan analisisnya ingin dilihat.

   Dasbor Wawasan Performa muncul untuk instans DB.
4. Gulir ke bawah dan pilih tab Laporan analisis performa - baru.

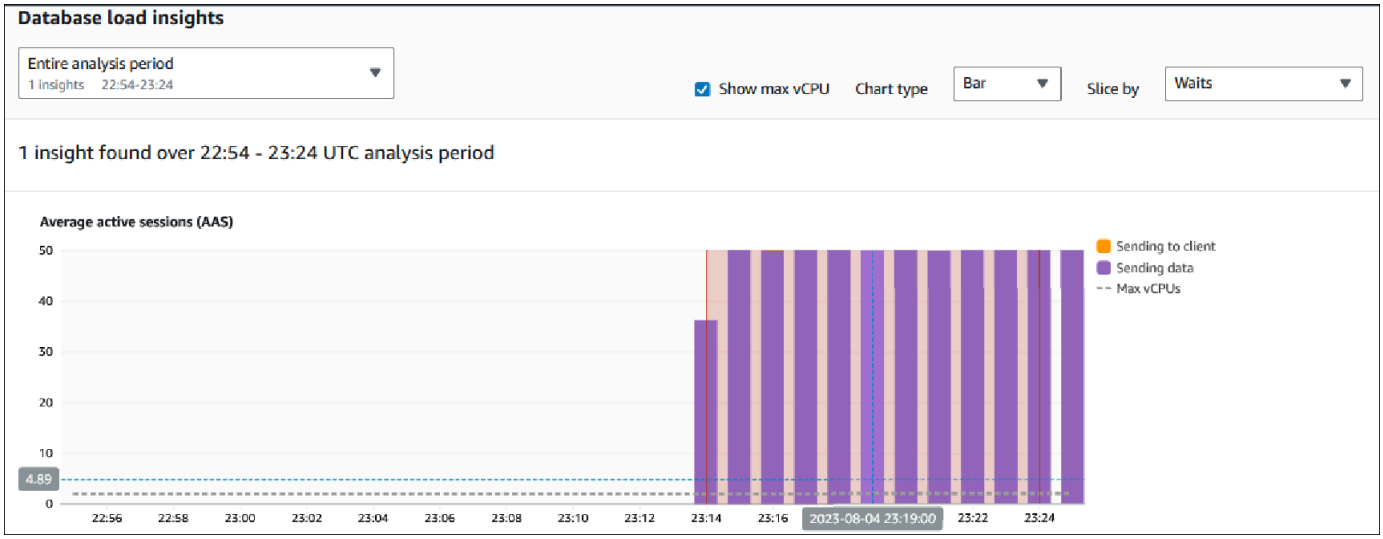
   Semua laporan analisis untuk periode waktu yang berbeda ditampilkan.

5. Pilih ID laporan yang ingin dilihat.

   Bagan muatan DB menampilkan seluruh periode analisis secara default jika ada lebih dari satu wawasan yang diidentifikasi. Jika laporan telah mengidentifikasi satu wawasan, bagan muatan DB akan menampilkan wawasan secara default.

   Dasbor juga mencantumkan tag untuk laporan di bagian Tag.

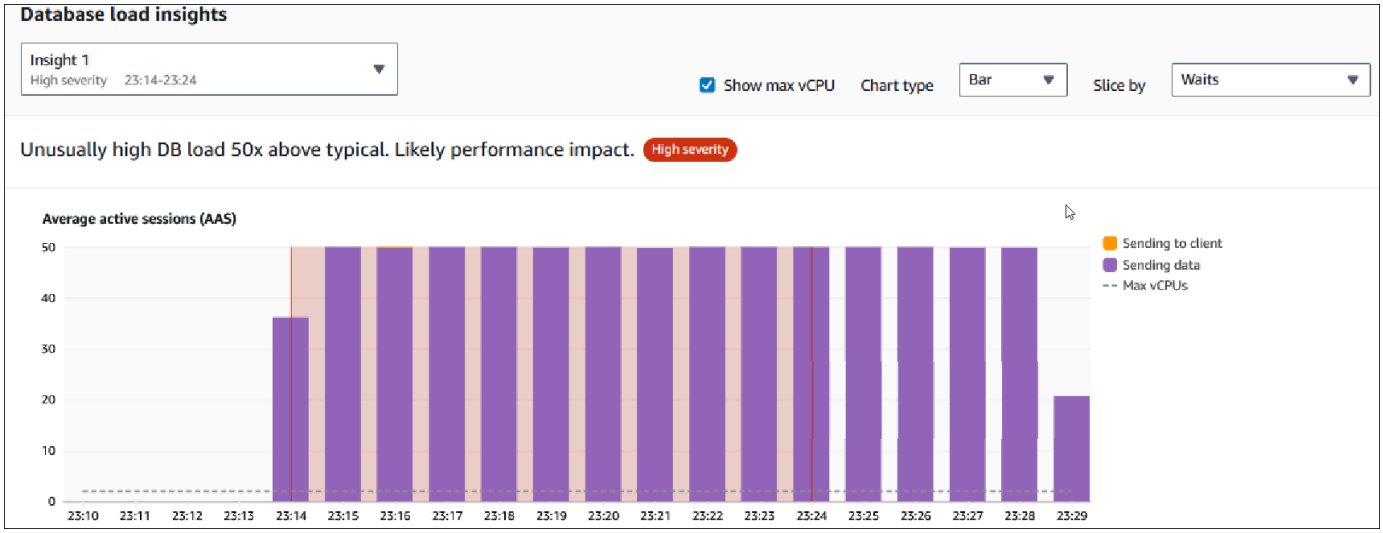
   Contoh berikut menunjukkan seluruh periode analisis untuk laporan.

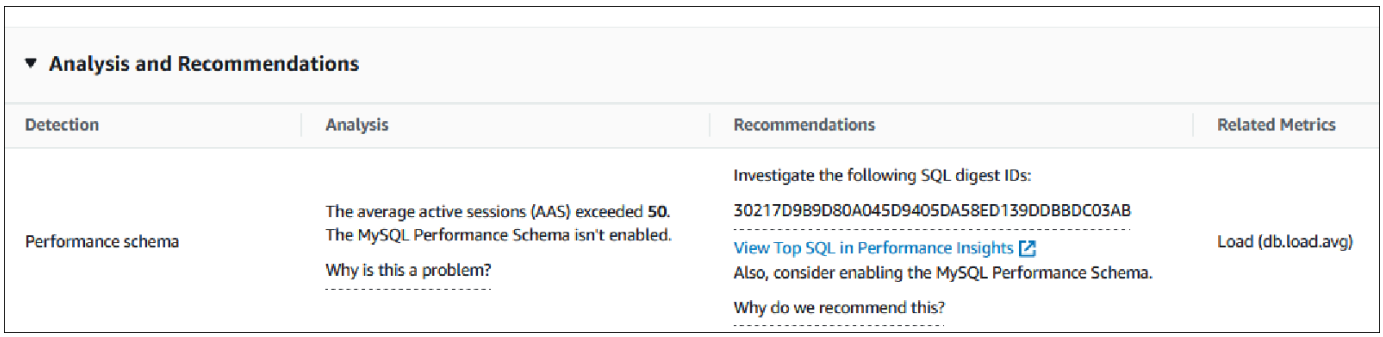


6. Pilih wawasan dalam daftar Wawasan muatan basis data yang ingin dilihat jika ada lebih dari satu wawasan yang diidentifikasi dalam laporan.

   Dasbor menampilkan pesan wawasan, bagan muatan DB yang menyoroti periode waktu wawasan, analisis dan rekomendasi, serta daftar tag laporan.

   Contoh berikut menunjukkan wawasan muatan DB dalam laporan.





Menambahkan tanda ke laporan analisis performa

Anda dapat menambahkan tag saat membuat atau melihat laporan. Anda dapat menambahkan hingga 50 tag untuk sebuah laporan.

Anda memerlukan izin untuk menambahkan tag. Untuk informasi selengkapnya tentang kebijakan akses untuk Wawasan Performa, lihat Mengonfigurasi kebijakan akses untuk Wawasan Performa

Untuk menambahkan satu atau beberapa tag saat membuat laporan, lihat langkah 6 dalam prosedur Membuat laporan analisis performa.

Untuk menambahkan satu tag atau lebih saat melihat laporan

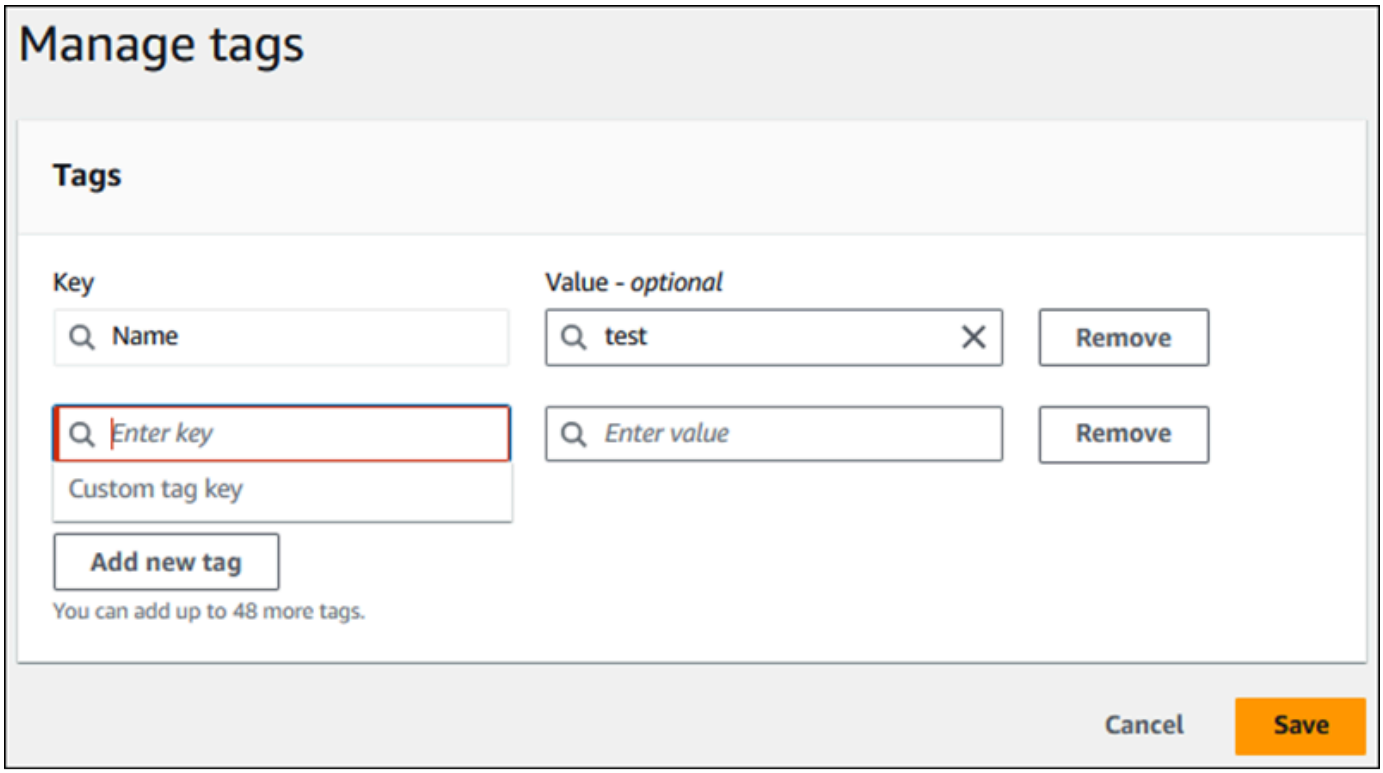
1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.

   Dasbor Wawasan Performa muncul untuk instans DB.

4. Gulir ke bawah dan pilih tab Laporan analisis performa - baru.
5. Pilih laporan yang ingin diberi tag.

   Dasbor menampilkan laporan.
6. Gulir ke bawah ke Tag dan pilih Kelola tag.
7. Pilih Tambahkan tag baru.
8. Masukkan Kunci dan Nilai - opsional, dan pilih Tambahkan tag baru.

   Contoh berikut memberikan opsi untuk menambahkan tag baru untuk laporan yang dipilih.



   Tag baru dibuat untuk laporan.

   Daftar tag untuk laporan ditampilkan di bagian Tag pada dasbor. Jika Anda ingin menghapus tag dari laporan, pilih Hapus di samping tag.

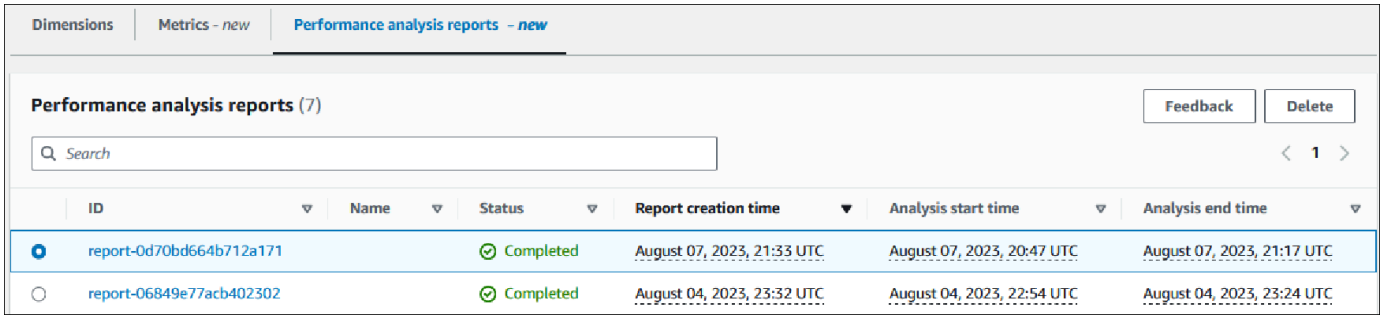
Menghapus laporan analisis performa

Anda dapat menghapus laporan dari daftar laporan yang ditampilkan di tab Laporan analisis performa atau saat melihat laporan.

Untuk menghapus laporan

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Pilih instans DB.

   Dasbor Wawasan Performa muncul untuk instans DB.
4. Gulir ke bawah dan pilih tab Laporan analisis performa - baru.
5. Pilih laporan yang ingin dihapus dan pilih Hapus di kanan atas.



   Jendela konfirmasi ditampilkan. Laporan akan dihapus setelah Anda memilih konfirmasi.
6. (Opsional) Pilih ID laporan yang ingin dihapus.

   Di halaman laporan, pilih Hapus di kanan atas.

   Jendela konfirmasi ditampilkan. Laporan akan dihapus setelah Anda memilih konfirmasi.

Menganalisis kueri di dasbor Wawasan Performa

Di dasbor Wawasan Performa Amazon RDS, Anda dapat menemukan informasi tentang menjalankan kueri terbaru di tab SQL Teratas di tabel Dimensi teratas. Anda dapat menggunakan informasi ini untuk menyetel kueri Anda.

Topik

- Ikhtisar tab SQL Teratas
- Mengakses lebih banyak teks SQL di dasbor Wawasan Performa
- Melihat statistik SQL di dasbor Wawasan Performa

Ikhtisar tab SQL Teratas

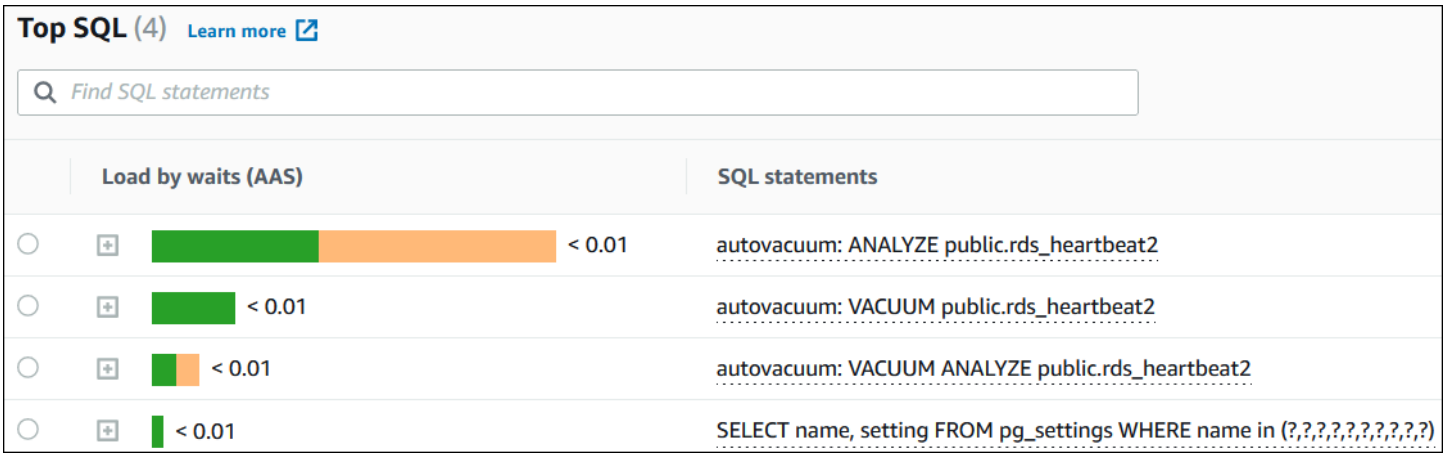
Secara default, tab SQL Teratas menunjukkan 25 kueri yang paling berkontribusi pada muatan DB. Untuk membantu menyetel kueri, Anda dapat menganalisis informasi seperti teks kueri dan statistik SQL. Anda juga dapat memilih statistik yang ingin ditampilkan di tab SQL Teratas.

Topik

- Teks SQL
- Statistik SQL
- Muatan berdasarkan status tunggu (AAS)
- Informasi SQL
- Preferensi

Teks SQL

Secara default, setiap baris dalam tabel SQL Teratas menunjukkan 500 byte teks untuk setiap pernyataan.



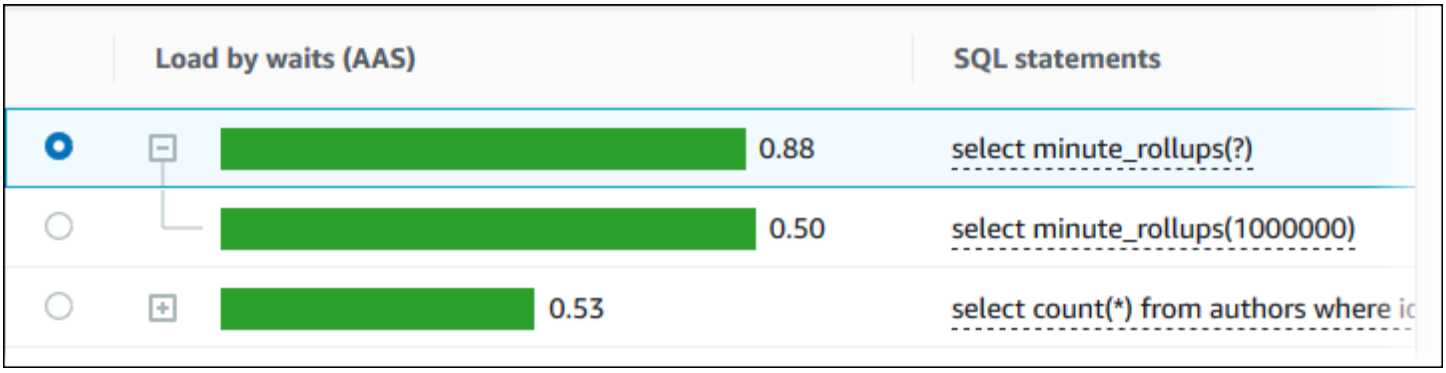
Untuk mempelajari cara melihat lebih dari 500 byte default teks SQL, lihat Mengakses lebih banyak teks SQL di dasbor Wawasan Performa.

Digest SQL adalah gabungan dari beberapa kueri aktual yang secara struktural serupa, tetapi mungkin memiliki nilai literal yang berbeda. Digest menggantikan nilai berkode keras dengan tanda tanya. Misalnya, digest mungkin berupa `SELECT * FROM emp WHERE lname= ?`. Digest ini dapat mencakup kueri turunan berikut:

```
SELECT * FROM emp WHERE lname = 'Sanchez'
```

```
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

Untuk menampilkan pernyataan SQL literal dalam sebuah digest, pilih kueri, lalu pilih simbol plus (+). Dalam contoh berikut, kueri yang dipilih adalah digest.

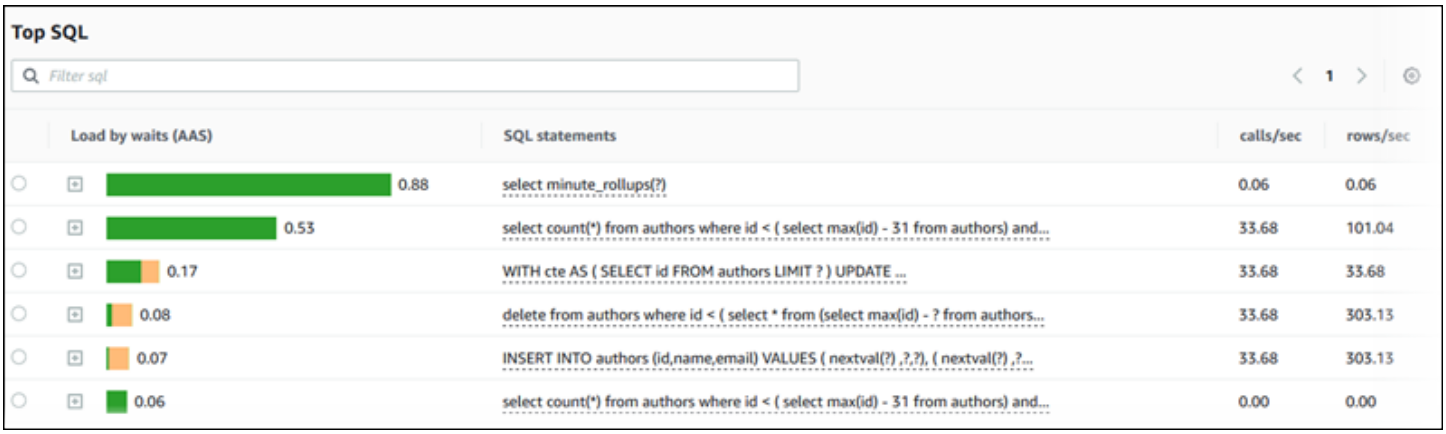


> **Note**
>
> Digest SQL mengelompokkan pernyataan SQL yang serupa, tetapi tidak menyunting informasi sensitif.

Statistik SQL

Statistik SQL adalah metrik terkait performa tentang kueri SQL. Misalnya, Wawasan Performa mungkin menampilkan eksekusi per detik atau baris yang diproses per detik. Wawasan Performa mengumpulkan statistik hanya untuk kueri yang paling umum. Biasanya, ini cocok dengan kueri teratas berdasarkan muatan yang ditampilkan di dasbor Wawasan Performa.

Setiap baris dalam tabel SQL Teratas menunjukkan statistik yang relevan untuk pernyataan SQL atau digest, seperti yang ditunjukkan dalam contoh berikut.



Wawasan Performa dapat melaporkan `0.00` dan `-` (tidak diketahui) untuk statistik SQL. Situasi ini terjadi dalam kondisi berikut:

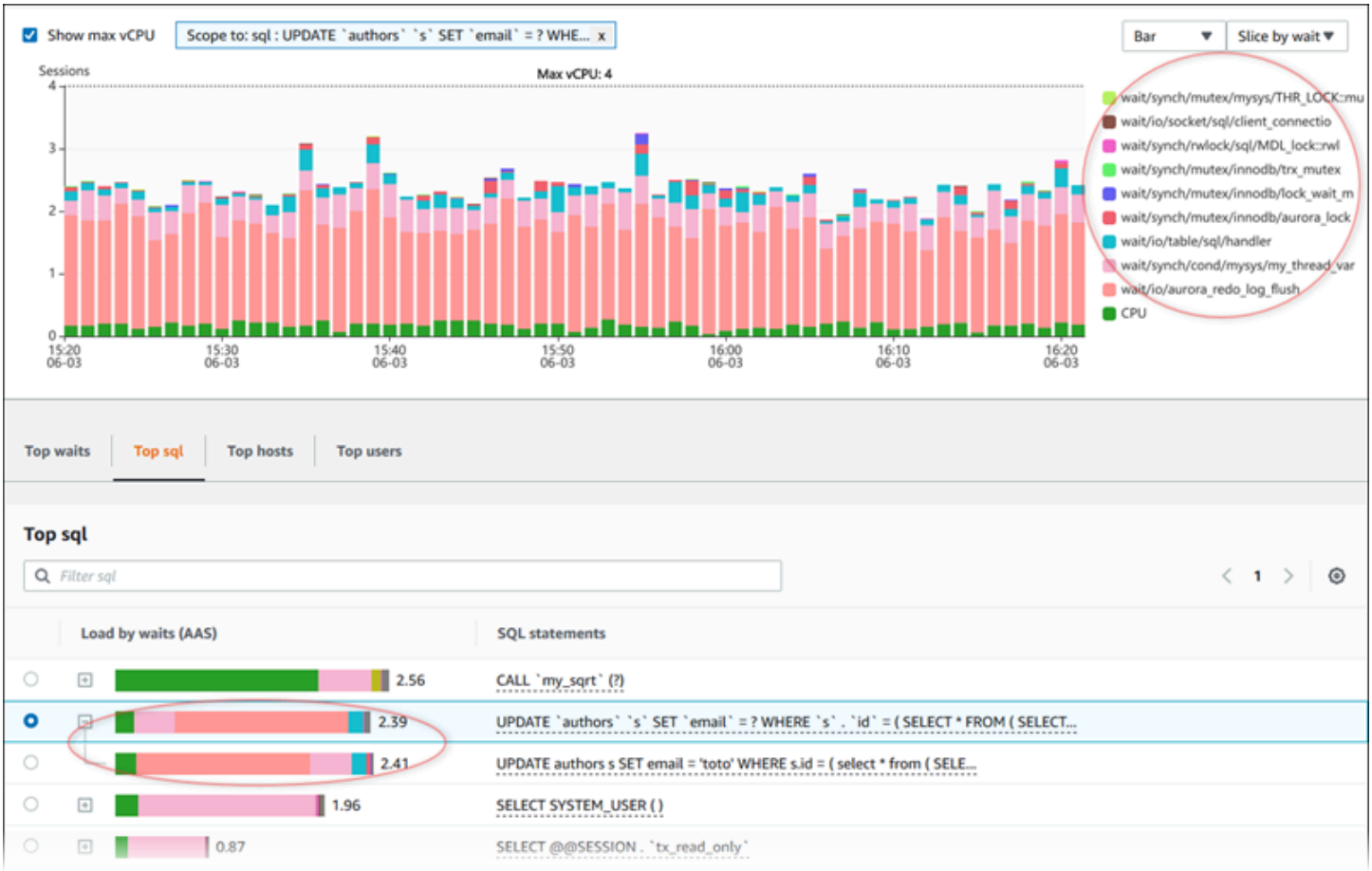
- Hanya ada satu sampel. Misalnya, Wawasan Performa menghitung tingkat perubahan untuk kueri Aurora PostgreSQL berdasarkan beberapa sampel dari tampilan `pg_stats_statements`. Ketika beban kerja berjalan untuk waktu yang singkat, Wawasan Performa mungkin hanya mengumpulkan satu sampel, yang berarti tidak dapat menghitung tingkat perubahan. Nilai yang tidak diketahui ditunjukkan dengan tanda hubung (`-`).
- Dua sampel memiliki nilai yang sama. Wawasan Performa tidak dapat menghitung tingkat perubahan karena tidak ada perubahan yang terjadi, sehingga melaporkan tingkatnya sebagai `0.00`.
- Pernyataan Aurora PostgreSQL tidak memiliki ID yang valid. PostgreSQL membuat ID untuk pernyataan hanya setelah diurai dan dianalisis. Dengan demikian, pernyataan bisa hadir dalam struktur internal dalam memori PostgreSQL tanpa ID. Karena Wawasan Performa mengambil sampel struktur internal dalam memori sekali per detik, kueri latensi rendah mungkin muncul hanya untuk satu sampel. Jika ID kueri tidak tersedia untuk sampel ini, Wawasan Performa tidak dapat mengaitkan pernyataan ini dengan statistiknya. Nilai yang tidak diketahui ditunjukkan dengan tanda hubung (`-`).

Untuk deskripsi statistik SQL untuk mesin Aurora, lihat Statistik SQL untuk Wawasan Performa.

Muatan berdasarkan status tunggu (AAS)

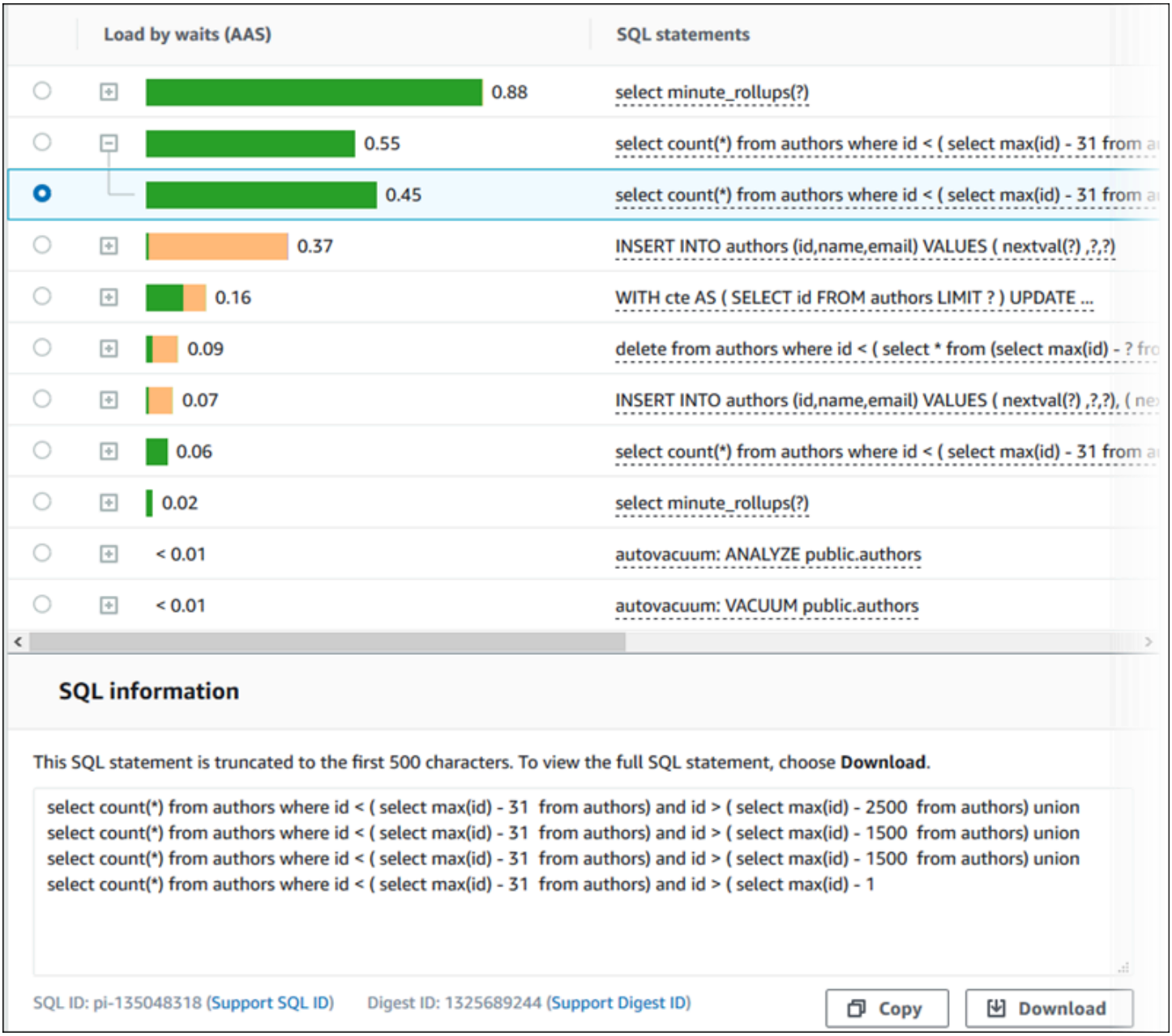
Di SQL Teratas, kolom Muatan berdasarkan status tunggu (AAS) menggambarkan persentase muatan basis data yang terkait dengan setiap item muatan teratas. Kolom ini menunjukkan muatan untuk item tersebut berdasarkan pengelompokan apa pun yang saat ini dipilih di Bagan Muatan DB. Untuk informasi selengkapnya tentang Sesi aktif rata-rata (AAS), lihat Sesi aktif rata-rata.

Misalnya, Anda mungkin mengelompokkan bagan Muatan DB berdasarkan status tunggu. Anda memeriksa kueri SQL di tabel item muatan teratas. Dalam kasus ini, bilah Muatan DB berdasarkan Status Tunggu diberi ukuran, disegmentasi, dan diberi kode warna untuk menunjukkan seberapa banyak status tunggu tertentu yang dikontribusikan oleh kueri. Bilah ini juga menunjukkan status tunggu yang memengaruhi kueri yang dipilih.



Informasi SQL

Di tabel SQL Teratas, Anda dapat membuka pernyataan untuk menampilkan informasinya. Informasi ini muncul di panel bawah.

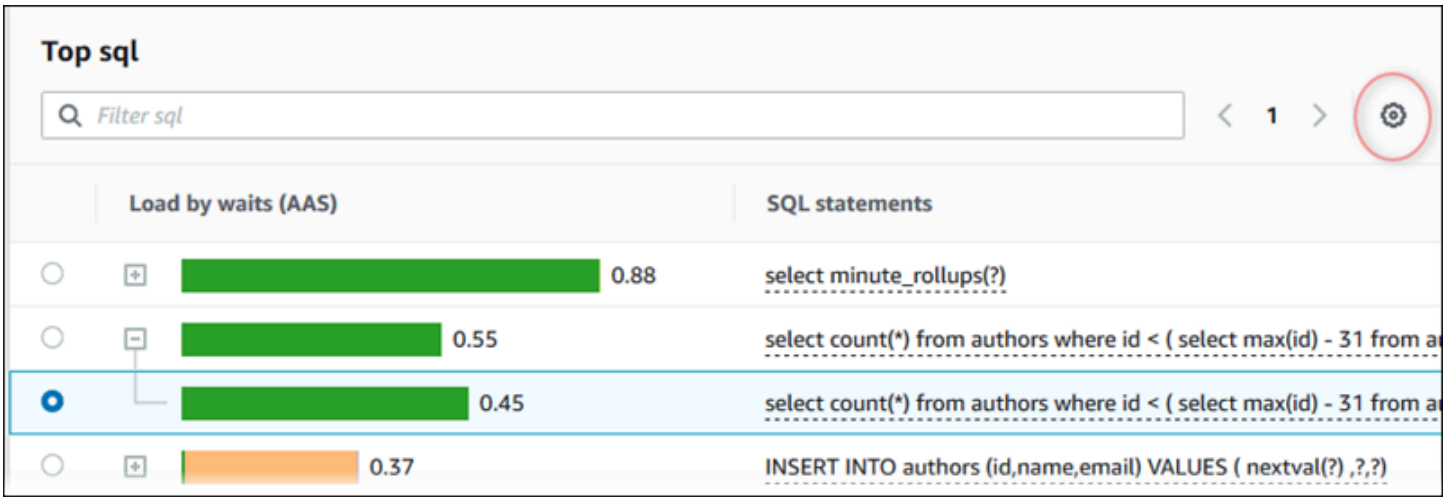


Berikut ini adalah jenis pengidentifikasi (ID) yang terkait dengan pernyataan SQL:

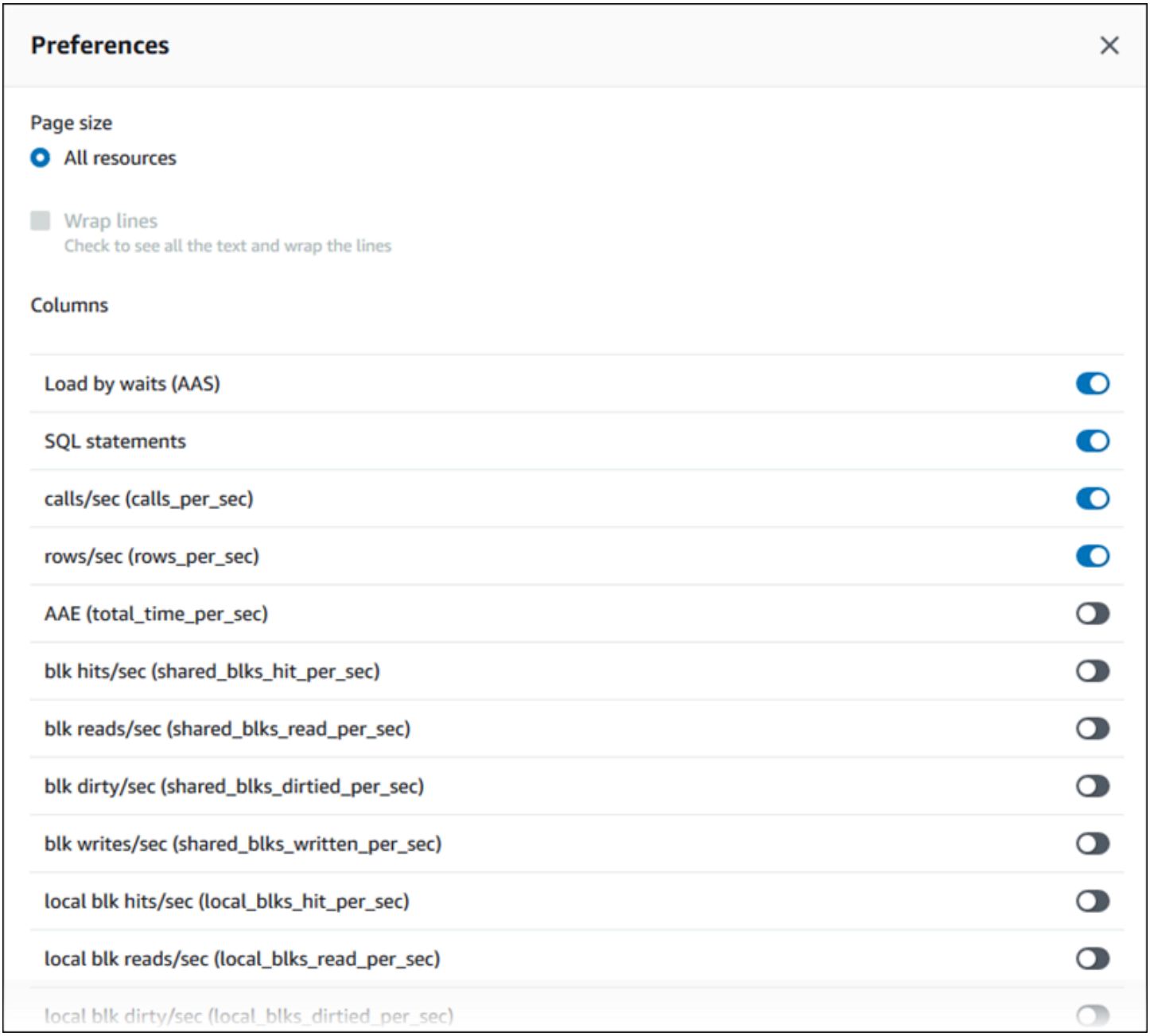
- Support SQL ID – Nilai hash dari ID SQL. Nilai ini hanya untuk mereferensikan ID SQL saat Anda bekerja dengan Dukungan AWS. AWS Dukungan tidak memiliki akses ke ID SQL dan teks SQL Anda yang sebenarnya.
- Support Digest ID – Nilai hash dari ID digest. Nilai ini hanya untuk mereferensikan ID digest saat Anda bekerja dengan Dukungan AWS. AWS Dukungan tidak memiliki akses ke ID digest dan teks SQL Anda yang sebenarnya.

Preferensi

Anda dapat mengontrol statistik yang ditampilkan di tab SQL Teratas dengan memilih ikon Preferensi.



Saat memilih ikon Preferensi, jendela Preferensi akan terbuka. Tangkapan layar berikut adalah contoh jendela Preferensi.



Untuk mengaktifkan statistik yang ingin ditampilkan di tab SQL Teratas, gunakan mouse untuk menggulir ke bagian bawah jendela, lalu pilih Lanjutkan.

Untuk informasi selengkapnya tentang statistik per detik atau per panggilan untuk mesin Aurora, lihat bagian statistik SQL khusus mesin di Statistik SQL untuk Wawasan Performa

Mengakses lebih banyak teks SQL di dasbor Wawasan Performa

Secara default, setiap baris dalam tabel SQL Teratas menunjukkan 500 byte teks SQL untuk setiap pernyataan SQL.

Saat pernyataan SQL melebihi 500 byte, Anda dapat melihat lebih banyak teks di bagian Teks SQL di bawah tabel SQL Teratas. Dalam hal ini, panjang maksimum teks yang ditampilkan dalam Teks SQL adalah 4 KB. Batas ini diperkenalkan oleh konsol dan tunduk pada batas yang ditetapkan oleh mesin basis data. Untuk menyimpan teks yang ditampilkan dalam Teks SQL, pilih Unduh.

Topik

- Batas ukuran teks untuk Aurora MySQL
- Mengatur batas teks SQL untuk instans DB Aurora PostgreSQL
- Melihat dan mengunduh teks SQL di dasbor Wawasan Performa

Batas ukuran teks untuk Aurora MySQL

Saat Anda mengunduh teks SQL, mesin basis data menentukan panjang maksimumnya. Anda dapat mengunduh teks SQL hingga batas per mesin berikut.

Mesin DB	Panjang maksimum teks yang diunduh
Aurora MySQL	4.096 byte

Bagian Teks SQL dari konsol Wawasan Performa menampilkan hingga maksimum yang ditampilkan mesin. Misalnya, jika Aurora MySQL menampilkan paling banyak 1 KB ke Wawasan Performa, Aurora MySQL hanya dapat mengumpulkan dan menampilkan 1 KB, meskipun kueri asalnya lebih besar. Jadi, jika Anda melihat kueri dalam Teks SQL atau mengunduhnya, Wawasan Performa akan menampilkan jumlah byte yang sama.

Jika Anda menggunakan AWS CLI atau API, Wawasan Performa tidak memiliki batas 4 KB yang diberlakukan oleh konsol. `DescribeDimensionKeys` dan `GetResourceMetrics` menampilkan paling banyak 500 byte. `GetDimensionKeyDetails` menampilkan kueri lengkap, tetapi ukurannya tunduk pada batas mesin.

Mengatur batas teks SQL untuk instans DB Aurora PostgreSQL

Aurora PostgreSQL menangani teks secara berbeda. Anda dapat mengatur batas ukuran teks dengan parameter instans DB `track_activity_query_size`. Parameter ini memiliki karakteristik sebagai berikut:

Ukuran teks default

Di Aurora PostgreSQL versi 9.6, pengaturan default untuk parameter `track_activity_query_size` adalah 1.024 byte. Di Aurora PostgreSQL versi 10 atau yang lebih baru, pengaturan default-nya adalah 4.096 byte.

Ukuran teks maksimum

Batas untuk `track_activity_query_size` adalah 102.400 byte untuk Aurora PostgreSQL versi 12 dan versi yang lebih rendah. Ukuran maksimumnya adalah 1 MB untuk versi 13 dan yang lebih baru.

Jika mesin menampilkan 1 MB ke Wawasan Performa, konsol hanya akan menampilkan 4 KB pertama. Jika mengunduh kueri, Anda akan mendapatkan 1 MB penuh. Dalam hal ini, melihat dan mengunduh menampilkan jumlah byte yang berbeda. Untuk informasi selengkapnya tentang parameter instans DB `track_activity_query_size`, lihat Statistik Runtime dalam dokumentasi PostgreSQL.

Untuk meningkatkan ukuran teks SQL, tingkatkan batas `track_activity_query_size`. Untuk memodifikasi parameter, ubah pengaturan parameter di grup parameter yang terkait dengan instans DB Aurora PostgreSQL.

Untuk mengubah pengaturan saat instans menggunakan grup parameter default

1. Buat grup parameter instans DB baru untuk mesin DB dan versi mesin DB yang sesuai.
2. Tetapkan parameter di grup parameter baru.
3. Hubungkan grup parameter baru dengan instans DB.

Untuk informasi tentang cara mengatur parameter instans DB, lihat Memodifikasi parameter dalam grup parameter DB.

Melihat dan mengunduh teks SQL di dasbor Wawasan Performa

Di dasbor Wawasan Performa, Anda dapat melihat atau mengunduh teks SQL.

Untuk melihat lebih banyak teks SQL di dasbor Wawasan Performa

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.

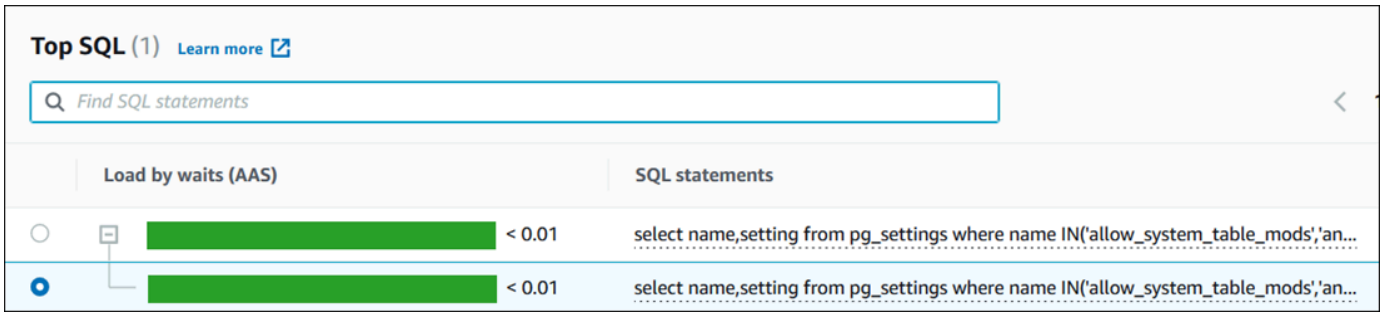
3. Pilih instans DB.

   Dasbor Wawasan Performa ditampilkan untuk instans DB Anda.

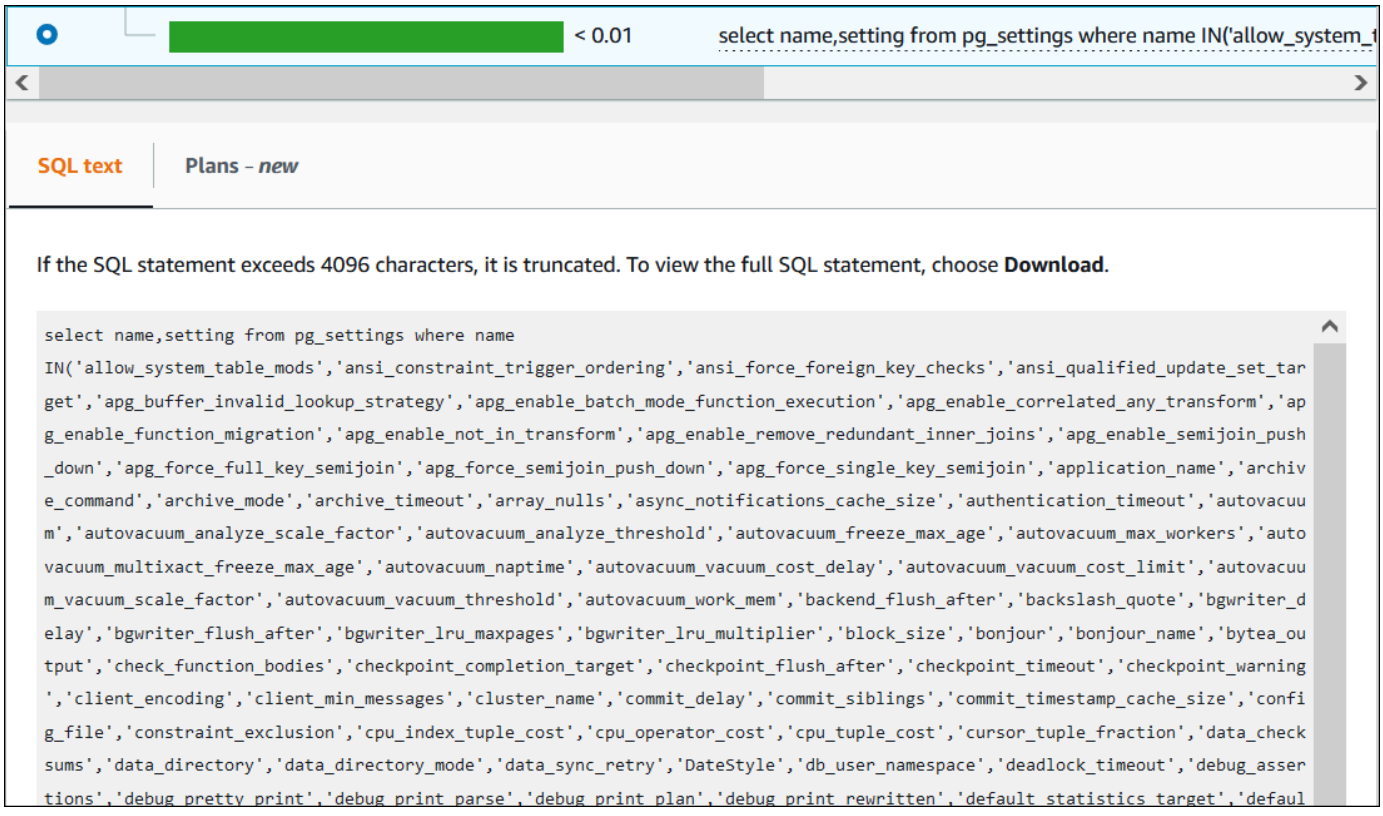
4. Gulir ke bawah ke tab SQL Teratas.

5. Pilih pernyataan SQL.

   Pernyataan SQL dengan teks yang lebih besar dari 500 byte terlihat hampir seperti gambar berikut.



6. Gulir ke bawah ke tab Teks SQL.



   Dasbor Wawasan Performa dapat menampilkan hingga 4.096 byte untuk setiap pernyataan SQL.

7. (Opsional) Pilih Salin untuk menyalin pernyataan SQL yang ditampilkan, atau pilih Unduh untuk mengunduh pernyataan SQL guna menampilkan teks SQL hingga batas mesin DB.

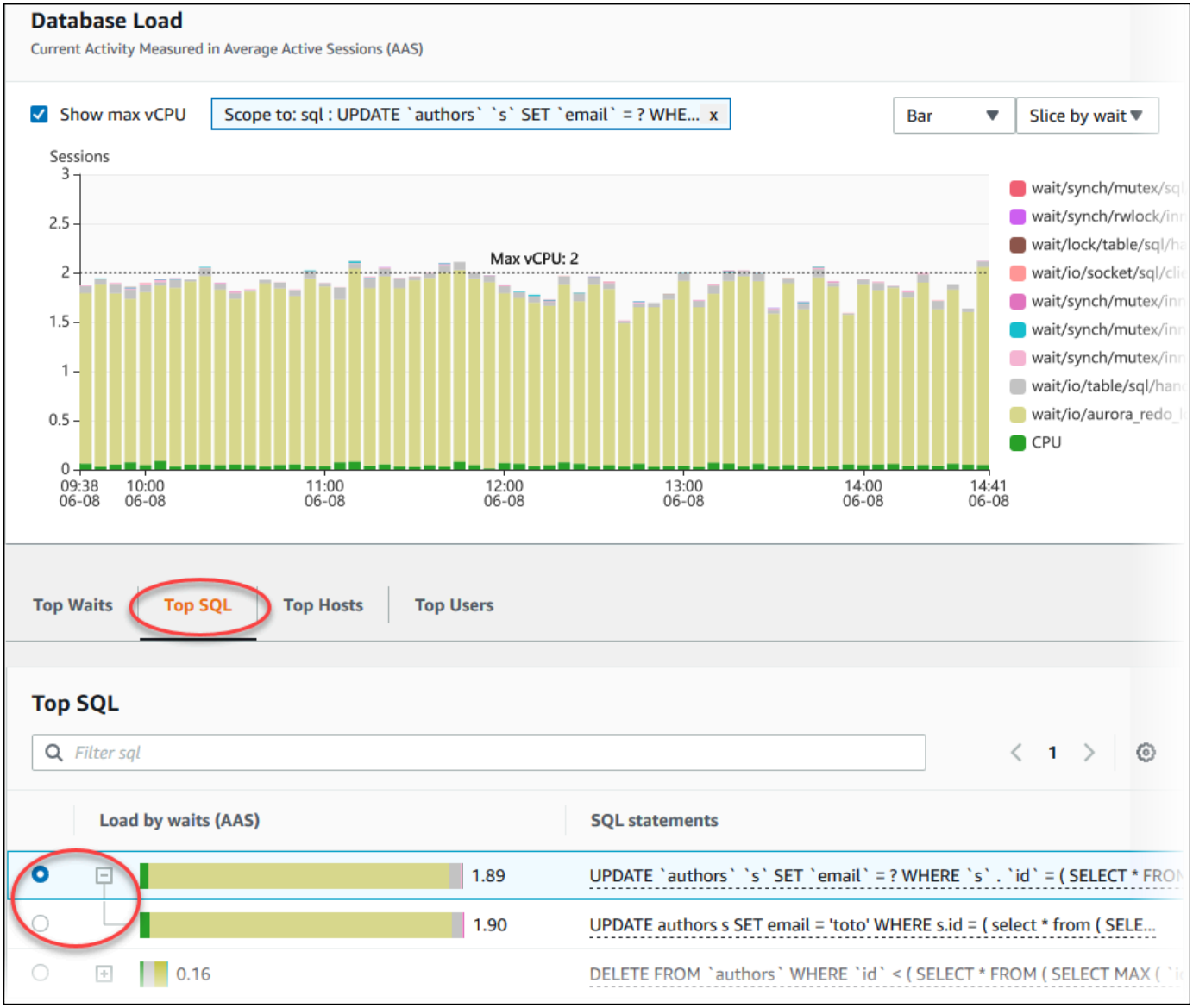
> **Note**
>
> Untuk menyalin atau mengunduh pernyataan SQL, nonaktifkan pemblokir pop-up.

Melihat statistik SQL di dasbor Wawasan Performa

Di dasbor Wawasan Performa, statistik SQL tersedia di tab SQL Teratas pada bagan Muatan basis data.

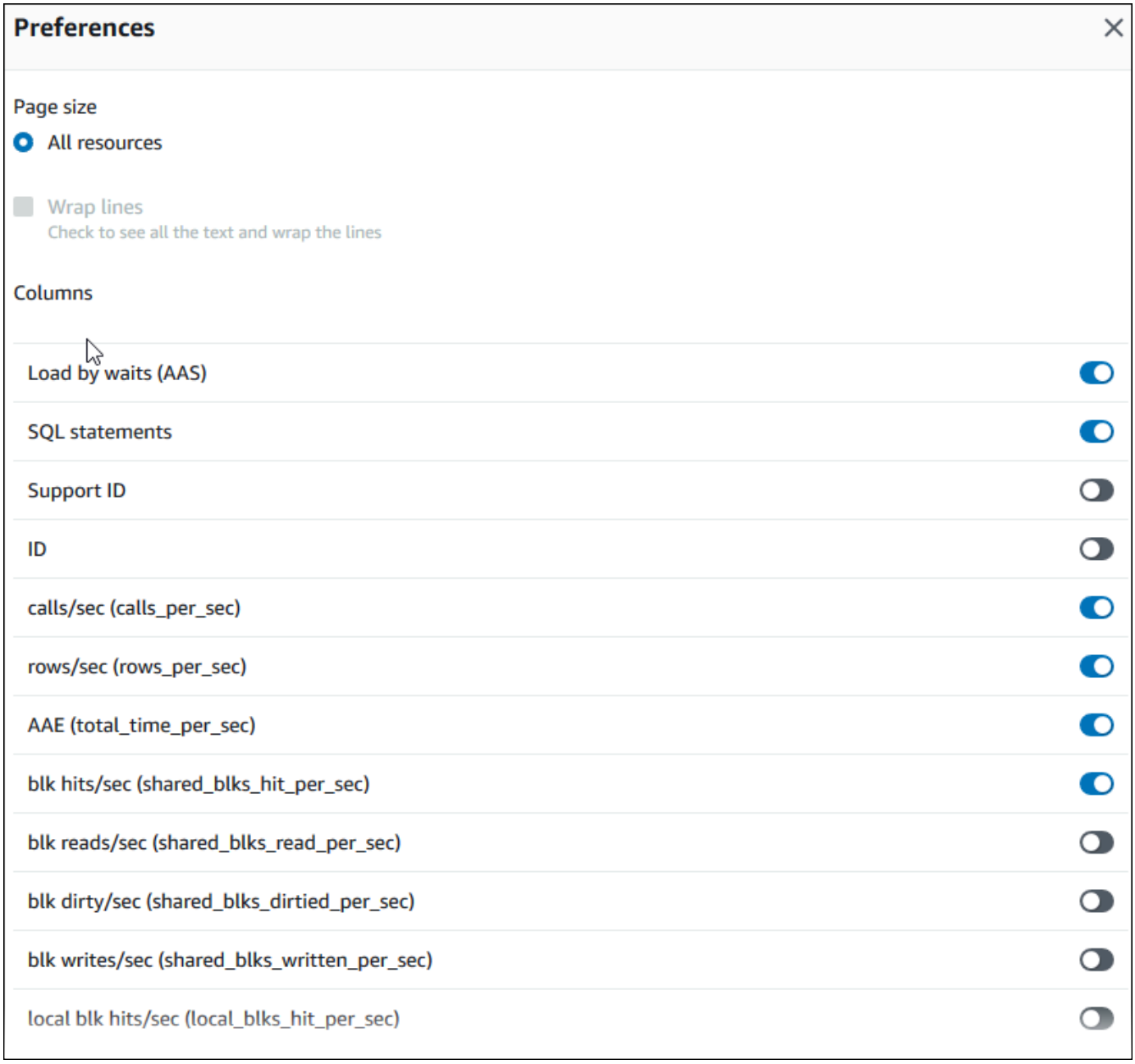
Untuk melihat statistik SQL

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi kiri, pilih Wawasan Performa.
3. Di bagian atas halaman, pilih basis data yang statistik SQL-nya ingin dilihat.
4. Gulir ke bagian bawah halaman dan pilih tab SQL Teratas.
5. Pilih setiap pernyataan (Hanya Aurora MySQL) atau kueri digest.

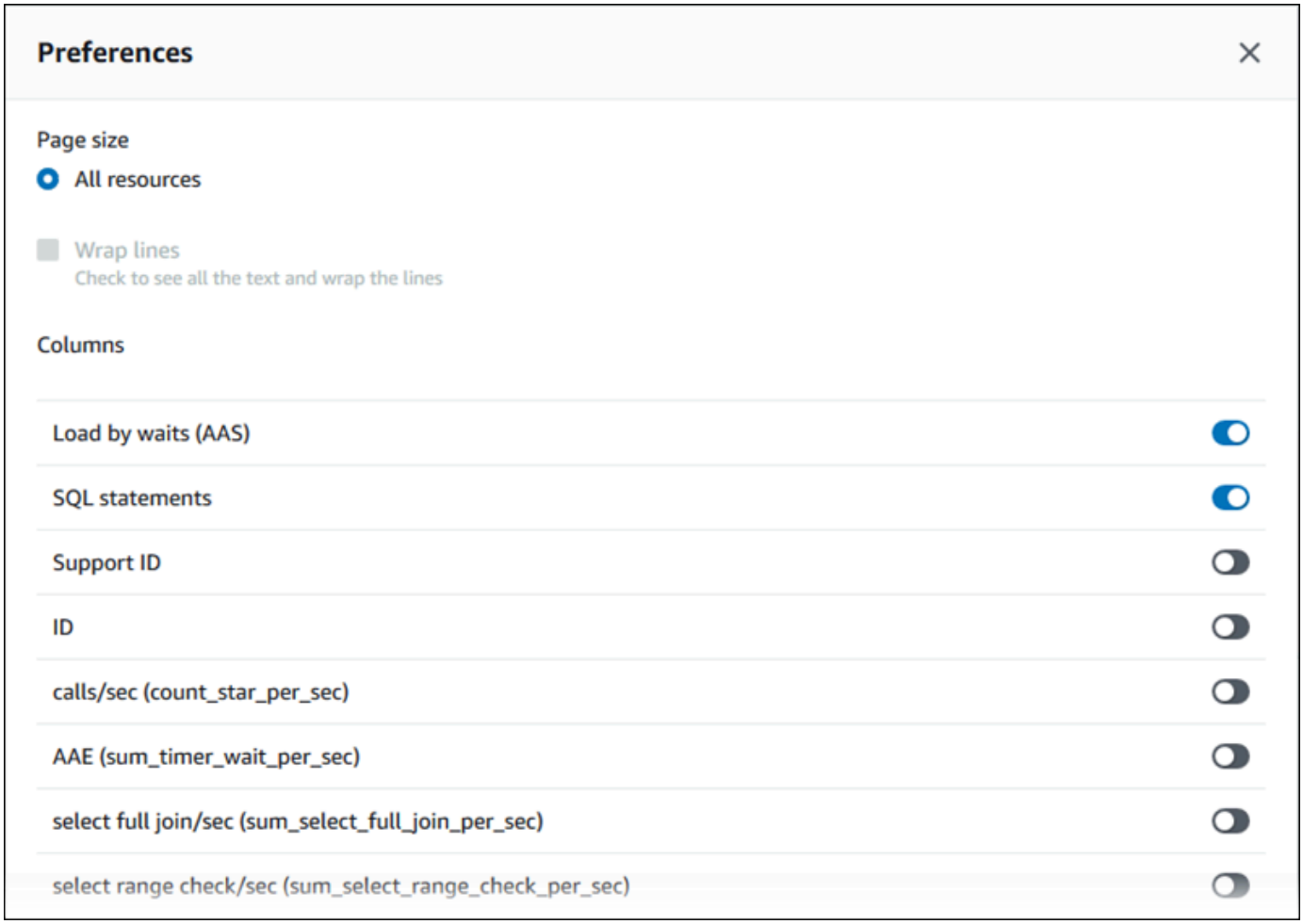


6. Pilih statistik yang akan ditampilkan dengan memilih ikon roda gigi di sudut kanan atas bagan. Untuk deskripsi statistik SQL untuk mesin Amazon RDSAurora, lihat Statistik SQL untuk Wawasan Performa.

   Contoh berikut menunjukkan preferensi untuk Aurora PostgreSQL.



Contoh berikut menunjukkan preferensi untuk instans DB Aurora MySQL.



7. Pilih Simpan untuk menyimpan preferensi Anda.

   Tabel SQL Top dimuat ulang.

Melihat rekomendasi proaktif Performance Insights

Amazon RDS Performance Insights memantau metrik tertentu dan secara otomatis membuat ambang batas dengan menganalisis level apa yang mungkin berpotensi bermasalah untuk sumber daya tertentu. Ketika nilai metrik baru melewati ambang batas yang telah ditentukan selama periode waktu tertentu, Performance Insights menghasilkan rekomendasi proaktif. Rekomendasi ini membantu mencegah dampak kinerja database future. Untuk menerima rekomendasi proaktif ini, Anda harus mengaktifkan Performance Insights dengan periode retensi tingkat berbayar.

Untuk informasi selengkapnya tentang mengaktifkan Wawasan Performa, lihat Mengaktifkan dan menonaktifkan Wawasan Performa. Untuk informasi tentang harga dan retensi data untuk Performance Insights, lihat. Harga dan retensi data untuk Wawasan Performa

Untuk mengetahui wilayah, mesin DB, dan kelas instance yang didukung untuk rekomendasi proaktif, lihatDukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk fitur Wawasan Performa.

Anda dapat melihat analisis terperinci dan investigasi rekomendasi proaktif yang direkomendasikan di halaman detail rekomendasi.

Untuk informasi lebih lanjut tentang rekomendasi, lihatMelihat dan menanggapi rekomendasi Amazon RDS.

Untuk melihat analisis rinci dari rekomendasi proaktif

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, lakukan salah satu hal berikut:

   - Pilih Rekomendasi.

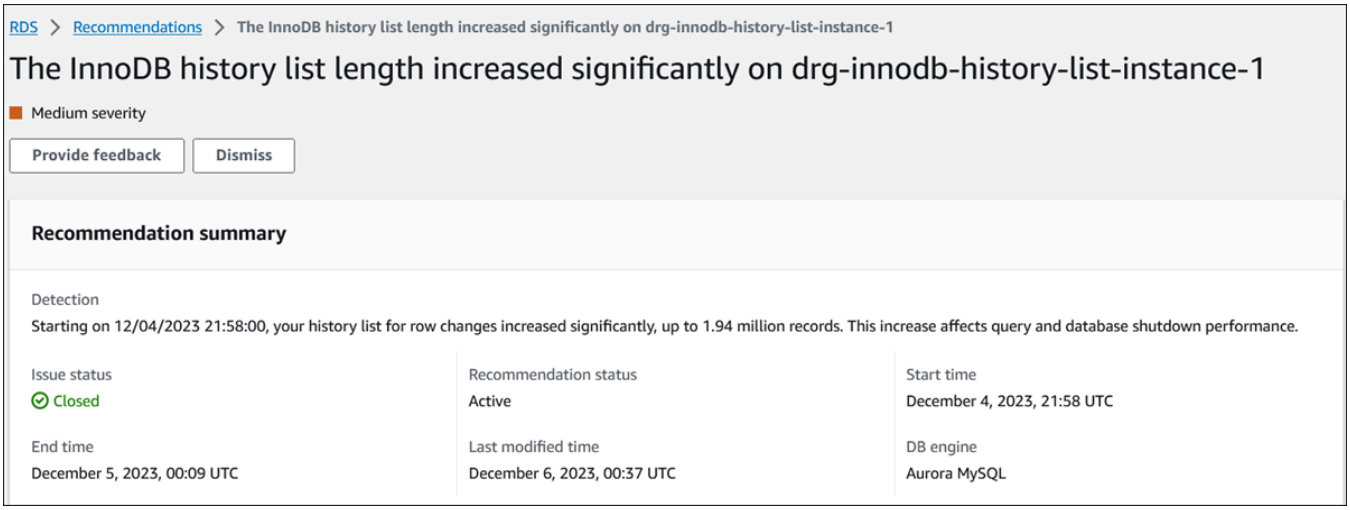
     Halaman Rekomendasi menampilkan daftar rekomendasi yang diurutkan berdasarkan tingkat keparahan semua sumber daya di akun Anda.
   - Pilih Database dan kemudian pilih Rekomendasi untuk sumber daya di halaman database.

     Tab Rekomendasi menampilkan rekomendasi dan detailnya untuk sumber daya yang dipilih.
3. Temukan rekomendasi proaktif dan pilih Lihat detail.

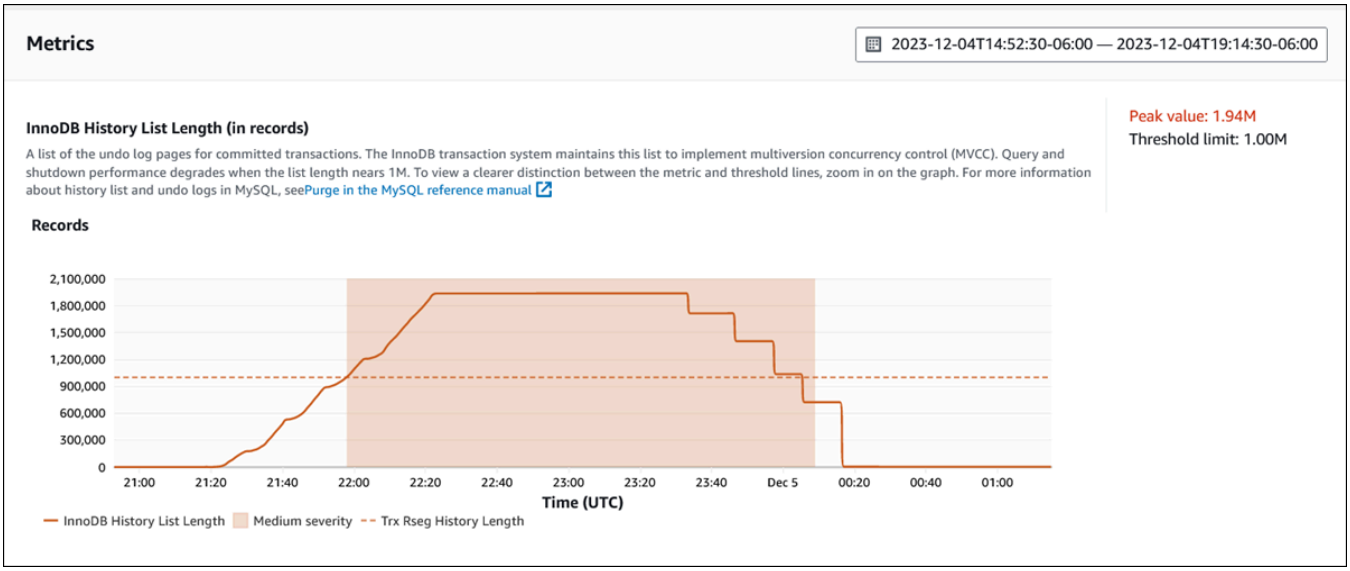
   Halaman detail rekomendasi muncul. Judul memberikan nama sumber daya yang terpengaruh dengan masalah yang terdeteksi dan tingkat keparahannya.

   Berikut ini adalah komponen pada halaman detail rekomendasi:

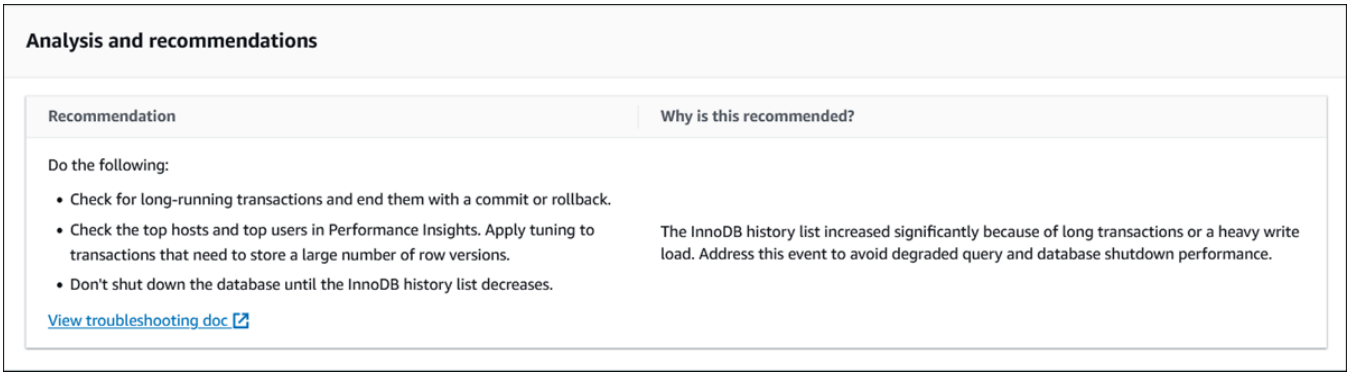
   - Ringkasan rekomendasi — Masalah yang terdeteksi, status rekomendasi dan masalah, waktu mulai dan berakhir masalah, waktu modifikasi rekomendasi, dan jenis mesin.



- Metrik — Grafik masalah yang terdeteksi. Setiap grafik menampilkan ambang batas yang ditentukan oleh perilaku dasar sumber daya dan data metrik yang dilaporkan dari waktu mulai masalah.



- Analisis dan rekomendasi — Rekomendasi dan alasan rekomendasi yang disarankan.



Anda dapat meninjau penyebab masalah dan kemudian melakukan tindakan yang disarankan untuk memperbaiki masalah, atau memilih Singkirkan di kanan atas untuk mengabaikan rekomendasi.

Mengambil metrik dengan API Wawasan Performa

Saat Wawasan Performa diaktifkan, API menyediakan visibilitas tentang performa instans. Amazon CloudWatch Logs menyediakan sumber otoritatif untuk metrik pemantauan terjual untuk layanan. AWS

Wawasan Performa menawarkan tampilan spesifik domain dari muatan basis data yang diukur sebagai sesi aktif rata-rata (AAS). Metrik ini muncul sebagai set data deret waktu dua dimensi bagi konsumen API. Dimensi waktu data menyediakan data muatan DB untuk setiap titik waktu dalam rentang waktu yang dikueri. Setiap titik waktu menguraikan keseluruhan muatan dalam hubungannya dengan dimensi yang diminta, seperti `SQL`, `Wait-event`, `User`, atau `Host`, yang diukur pada titik waktu tersebut.

Wawasan Performa Amazon RDS memantau klaster Amazon Aurora Anda sehingga Anda dapat menganalisis dan memecahkan masalah performa basis data. Salah satu cara untuk menampilkan data Wawasan Performa dapat ditemukan di AWS Management Console. Wawasan Performa juga menyediakan API publik sehingga Anda dapat mengkueri data Anda sendiri. Anda dapat menggunakan API untuk melakukan tindakan berikut:

- Membongkar data ke dalam basis data
- Menambahkan data Wawasan Performa ke dasbor pemantauan yang ada
- Merancang alat pemantauan

Untuk menggunakan API Wawasan Performa, aktifkan Wawasan Performa di salah satu instans DB Amazon RDS Anda. Untuk informasi tentang cara mengaktifkan Wawasan Performa, lihat Mengaktifkan dan menonaktifkan Wawasan Performa. Untuk informasi selengkapnya tentang API Wawasan Performa, lihat Referensi API Wawasan Performa Amazon RDS.

API Wawasan Performa menyediakan operasi berikut.

Tindakan Wawasan Performa	Perintah AWS CLI	Deskripsi
CreatePerformanceAnalysisReport	aws pi create-performance-analysis-report	Membuat laporan analisis performa selama periode waktu tertentu untuk instans DB. Hasilnya adalah `AnalysisReportId` , yakni pengidentifikasi unik laporan.
DeletePerformanceAnalysisReport	aws pi delete-performance-analysis-report	Menghapus laporan analisis performa.
DescribeDimensionKeys	aws pi describe-dimension-keys	Mengambil kunci dimensi N teratas untuk metrik selama periode waktu tertentu.
GetDimensionKeyDetails	aws pi get-dimension-key-details	Mengambil atribut dari grup dimensi tertentu untuk instans DB atau sumber data. Misalnya, jika Anda menentukan ID SQL, dan jika detail dimensi tersedia, `GetDimensionKeyDetails` akan mengambil teks lengkap dimensi `db.sql.statement` yang terkait dengan ID ini. Operasi ini berguna karena `GetResourceMetrics` dan `DescribeDimensionKeys` tidak mendukung pengambilan teks pernyataan SQL besar.

Tindakan Wawasan Performa	Perintah AWS CLI	Deskripsi
GetPerformanceAnalysisReport	aws pi get-performance-analysis-report	Mengambil laporan yang mencakup wawasan untuk laporan. Hasilnya mencakup status laporan, ID laporan, detail waktu laporan, wawasan, dan rekomendasi.
GetResourceMetadata	aws pi get-resource-metadata	Mengambil metadata untuk fitur yang berbeda. Misalnya, metadata mungkin menunjukkan bahwa fitur diaktifkan atau dinonaktifkan pada instans DB tertentu.
GetResourceMetrics	aws pi get-resource-metrics	Mengambil metrik Wawasan Performa untuk sekumpulan sumber data, selama periode waktu tertentu. Anda dapat menyediakan grup dimensi dan dimensi tertentu, serta memberikan kriteria penggabungan dan pemfilteran untuk setiap grup.
ListAvailableResourceDimensions	aws pi list-available-resource-dimensions	Mengambil dimensi yang dapat dikueri untuk setiap jenis metrik yang ditentukan pada instans tertentu.
ListAvailableResourceMetrics	aws pi list-available-resource-metrics	Mengambil semua metrik jenis metrik tertentu yang tersedia yang dapat dikueri untuk instans DB tertentu.

Tindakan Wawasan Performa	Perintah AWS CLI	Deskripsi
ListPerformanceAnalysisReports	aws pi list-performance-analysis-reports	Mengambil semua laporan analisis yang tersedia untuk instans DB. Laporan dicantumkan berdasarkan waktu mulai setiap laporan.
ListTagsForResource	aws pi list-tags-for-resource	Daftar semua tag metadata yang ditambahkan ke sumber daya. Daftar ini mencakup nama dan nilai tag.
TagResource	aws pi tag-resource	Menambahkan tag metadata ke sumber daya Amazon RDS. Tag ini termasuk nama dan nilai.
UntagResource	aws pi untag-resource	Menghapus tag metadata dari sumber daya.

Topik

- AWS CLI untuk Wawasan Performa
- Mengambil metrik deret waktu
- Contoh AWS CLI untuk Wawasan Performa

AWS CLI untuk Wawasan Performa

Anda dapat melihat data Wawasan Performa menggunakan AWS CLI. Anda dapat menampilkan bantuan untuk perintah AWS CLI untuk Wawasan Performa dengan memasukkan berikut ini di baris perintah.

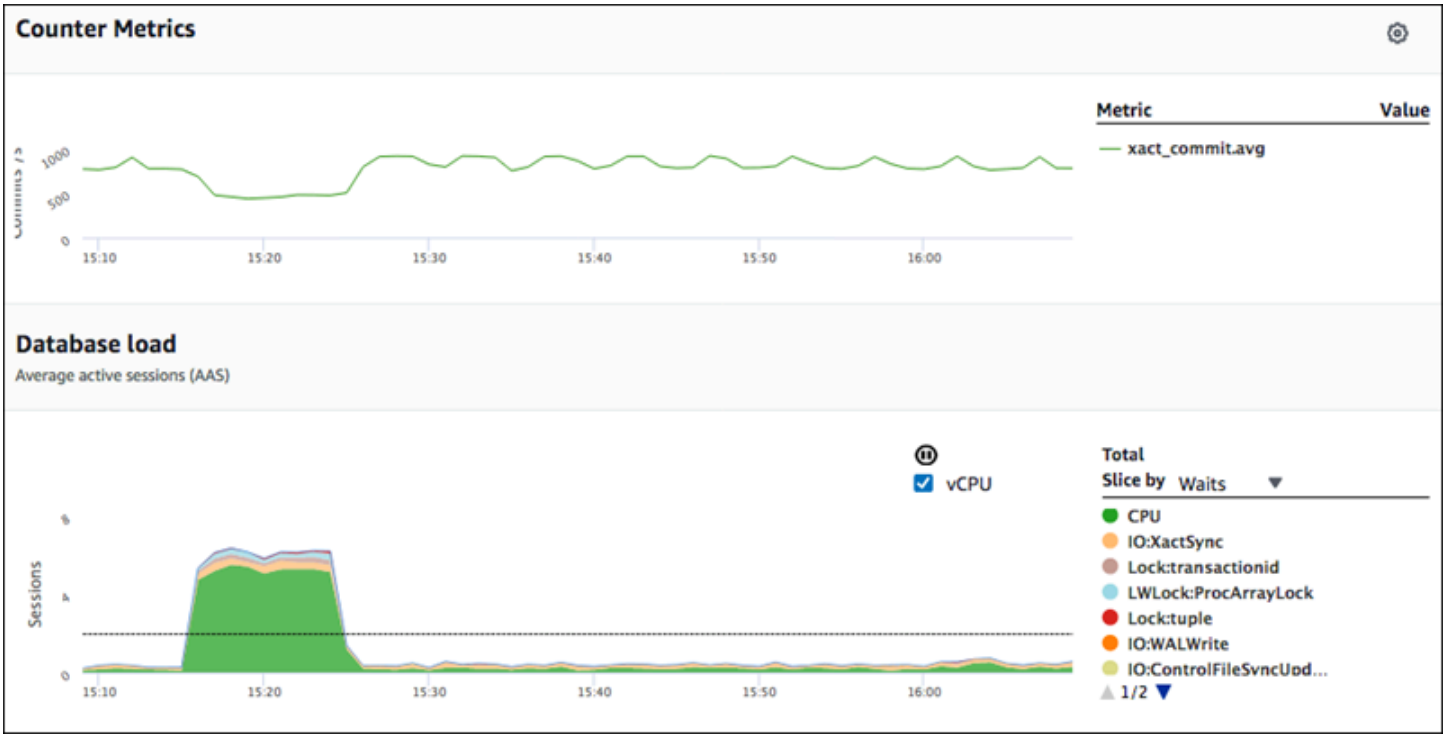
```
aws pi help
```

Jika Anda belum menginstal AWS CLI, lihat Memasang Antarmuka Baris Perintah AWS di Panduan Pengguna AWS CLI untuk informasi tentang cara menginstalnya.

Mengambil metrik deret waktu

Operasi `GetResourceMetrics` mengambil satu atau beberapa metrik deret waktu dari data Wawasan Performa. `GetResourceMetrics` memerlukan metrik dan periode waktu, dan menampilkan respons dengan daftar poin data.

Misalnya, AWS Management Console menggunakan `GetResourceMetrics` untuk mengisi bagan Metrik Penghitung dan bagan Muatan Basis Data seperti yang diperlihatkan pada gambar berikut.



Semua metrik yang ditampilkan oleh `GetResourceMetrics` adalah metrik deret waktu standar, dengan pengecualian `db.load`. Metrik ini ditampilkan dalam diagram Muatan Basis Data. Metrik `db.load` berbeda dengan metrik seri waktu lainnya karena Anda dapat membaginya menjadi beberapa sub-komponen yang disebut dimensi. Di gambar sebelumnya, `db.load` dibagi dan dikelompokkan berdasarkan status tunggu yang membentuk `db.load`.

> **Note**
>
> `GetResourceMetrics` juga dapat menampilkan metrik `db.sampleload`, tetapi metrik `db.load` sesuai di sebagian besar kasus.

Untuk informasi tentang metrik penghitung yang ditampilkan oleh `GetResourceMetrics`, lihat Metrik penghitung Wawasan Performa.

Penghitungan berikut didukung untuk metrik:

- Rata-rata – Nilai rata-rata untuk metrik selama periode waktu tertentu. Tambahkan `.avg` ke nama metrik.
- Minimum – Nilai minimum metrik selama periode waktu tertentu. Tambahkan `.min` ke nama metrik.
- Maksimum – Nilai maksimum metrik selama periode waktu tertentu. Tambahkan `.max` ke nama metrik.
- Jumlah – Jumlah nilai metrik selama periode waktu tertentu. Tambahkan `.sum` ke nama metrik.
- Jumlah sampel – Frekuensi pengumpulan metrik selama periode waktu tertentu. Tambahkan `.sample_count` ke nama metrik.

Sebagai contoh, misalkan sebuah metrik dikumpulkan selama 300 detik (5 menit), dan metrik tersebut dikumpulkan satu kali setiap menit. Nilai untuk setiap menit adalah 1, 2, 3, 4, dan 5. Dalam kasus ini, penghitungan berikut ditampilkan:

- Rata-rata – 3
- Minimum – 1
- Maksimum – 5
- Jumlah – 15
- Jumlah sampel – 5

Untuk informasi tentang cara menggunakan perintah `get-resource-metrics` AWS CLI, lihat `get-resource-metrics`.

Untuk opsi `--metric-queries`, tentukan satu atau beberapa kueri yang diinginkan untuk mendapatkan hasil. Setiap kueri terdiri dari `Metric` wajib dan `GroupBy` opsional serta parameter `Filter`. Berikut ini adalah contoh spesifikasi opsi `--metric-queries`.

```
{
   "Metric": "string",
   "GroupBy": {
     "Group": "string",
     "Dimensions": ["string", ...],
     "Limit": integer
   },
   "Filter": {"string": "string"
```

```
    ...}
```

Contoh AWS CLI untuk Wawasan Performa

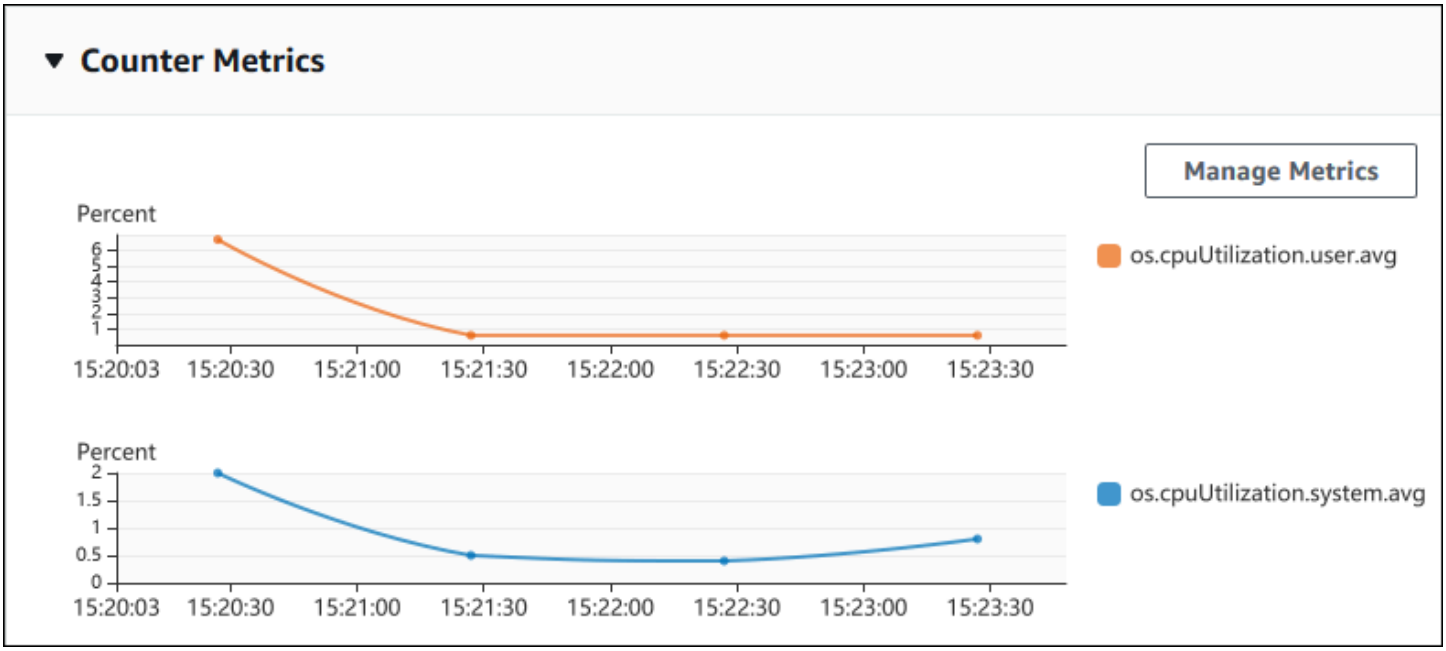
Contoh berikut menunjukkan cara menggunakan AWS CLI untuk Wawasan Performa.

Topik

- Mengambil metrik penghitung
- Mengambil rata-rata muatan DB untuk peristiwa tunggu teratas
- Mengambil rata-rata muatan DB untuk SQL teratas
- Mengambil Rata-Rata Muatan DB yang difilter berdasarkan SQL
- Mengambil teks lengkap pernyataan SQL
- Membuat laporan analisis performa selama periode waktu tertentu
- Mengambil laporan analisis performa
- Daftar semua laporan analisis performa untuk instans DB
- Menghapus laporan analisis performa
- Menambahkan tag ke laporan analisis performa
- Mencantumkan semua tag untuk laporan analisis performa
- Menghapus tag dari laporan analisis performa

Mengambil metrik penghitung

Tangkapan layar berikut menunjukkan dua bagan metrik penghitung dalam AWS Management Console.



Contoh berikut menunjukkan cara mengumpulkan data yang sama yang digunakan oleh AWS Management Console untuk menghasilkan dua bagan metrik penghitung.

Untuk Linux, macOS, atau Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

Untuk Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries '[{"Metric": "os.cpuUtilization.user.avg"  },
                      {"Metric": "os.cpuUtilization.idle.avg"}]'
```

Anda juga dapat membaca perintah dengan lebih mudah dengan menentukan file untuk opsi `--metrics-query`. Contoh berikut menggunakan file yang disebut query.json untuk opsi tersebut. File memiliki konten berikut.

```
[
    {
        "Metric": "os.cpuUtilization.user.avg"
    },
    {
        "Metric": "os.cpuUtilization.idle.avg"
    }
]
```

Jalankan perintah berikut untuk menggunakan file.

Untuk Linux, macOS, atau Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Untuk Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

Contoh sebelumnya menentukan nilai-nilai berikut untuk opsi:

- `--service-type` – RDS untuk Amazon RDS
- `--identifier` – ID sumber daya untuk instans DB
- `--start-time` dan `--end-time` – Nilai `DateTime` ISO 8601 untuk periode kueri, dengan berbagai format yang didukung

Ini akan dikueri selama rentang waktu satu jam:

- `--period-in-seconds` – `60` untuk kueri per menit
- `--metric-queries` – Rangkaian dua kueri, masing-masing hanya untuk satu metrik.

  Nama metrik menggunakan titik untuk mengklasifikasikan metrik dalam kategori yang berguna, dengan elemen terakhir sebagai fungsi. Dalam contoh, fungsinya adalah `avg` untuk setiap kueri. Seperti halnya Amazon CloudWatch, fungsi yang didukung adalah `min`,`max`,`total`, dan`avg`.

Responsnya terlihat seperti berikut.

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                "Metric": "os.cpuUtilization.user.avg" //Metric1
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 4.0
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 10.0
                }
                //... 60 datapoints for the os.cpuUtilization.user.avg metric
            ]
        },
        {
            "Key": {
                "Metric": "os.cpuUtilization.idle.avg" //Metric2
            },
```

```
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 12.0
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 13.5
                },
                //... 60 datapoints for the os.cpuUtilization.idle.avg metric
            ]
        }
    ] //end of MetricList
} //end of response
```

Respons ini memiliki `Identifier`, `AlignedStartTime`, dan `AlignedEndTime`. B nilai `--period-in-seconds` adalah `60`, waktu mulai dan akhir telah disesuaikan dengan menit. Jika `--period-in-seconds` adalah `3600`, waktu mulai dan akhir akan disesuaikan dengan jam.

`MetricList` dalam respons memiliki sejumlah entri, masing-masing dengan entri `Key` dan `DataPoints`. Masing-masing `DataPoint` memiliki `Timestamp` dan `Value`. Masing-masing daftar `Datapoints` memiliki 60 poin data karena kueri tersebut adalah untuk data per menit selama satu jam, dengan `Timestamp1/Minute1`, `Timestamp2/Minute2`, dan seterusnya, hingga `Timestamp60/Minute60`.

Karena kueri tersebut adalah untuk dua metrik penghitung yang berbeda, ada dua elemen dalam respons `MetricList`.

Mengambil rata-rata muatan DB untuk peristiwa tunggu teratas

Contoh berikut adalah kueri yang sama yang digunakan oleh AWS Management Console untuk menghasilkan grafik baris area bertumpuk. Contoh ini mengambil `db.load.avg` selama satu jam terakhir dengan muatan yang dibagi berdasarkan tujuh peristiwa tunggu teratas. Perintah ini sama dengan perintah dalam Mengambil metrik penghitung. Namun, file query.json berisi konten berikut.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 7 }
    }
]
```

Jalankan perintah berikut.

Untuk Linux, macOS, atau Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Untuk Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

Contoh ini menentukan metrik `db.load.avg` dan `GroupBy` dari tujuh peristiwa tunggu teratas. Untuk detail tentang nilai yang valid untuk contoh ini, lihat DimensionGroup dalam Referensi API Wawasan Kinerja.

Responsnya terlihat seperti berikut.

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1540857600.0,
    "AlignedEndTime": 1540861200.0,
    "MetricList": [
        { //A list of key/datapoints
            "Key": {
                //A Metric with no dimensions. This is the total db.load.avg
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                //Each list of datapoints has the same timestamps and same number of
 items
                {
```

```
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.5166666666666667
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.38333333333333336
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 0.26666666666666666
                }
                //... 60 datapoints for the total db.load.avg key
            ]
        },
        {
            "Key": {
                //Another key. This is db.load.avg broken down by CPU
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.name": "CPU",
                    "db.wait_event.type": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1540857660.0, //Minute1
                    "Value": 0.35
                },
                {
                    "Timestamp": 1540857720.0, //Minute2
                    "Value": 0.15
                },
                //... 60 datapoints for the CPU key
            ]
        },
        //... In total we have 8 key/datapoints entries, 1) total, 2-8) Top Wait Events
    ] //end of MetricList
} //end of response
```

Dalam respon ini, ada delapan entri dalam `MetricList`. Ada satu entri untuk total `db.load.avg`, dan tujuh entri masing-masing untuk `db.load.avg` yang dibagi berdasarkan salah satu dari tujuh peristiwa tunggu teratas. Tidak seperti di contoh pertama, karena ada dimensi pengelompokan, pasti

ada satu kunci untuk setiap pengelompokan metrik. Tidak boleh hanya ada satu kunci untuk setiap metrik, seperti dalam kasus penggunaan metrik penghitung dasar.

Mengambil rata-rata muatan DB untuk SQL teratas

Contoh berikut mengelompokkan `db.wait_events` berdasarkan 10 pernyataan SQL teratas. Ada dua grup berbeda untuk pernyataan SQL:

- `db.sql` – Pernyataan SQL lengkap, seperti `select * from customers where customer_id = 123`
- `db.sql_tokenized` – Pernyataan SQL token, seperti `select * from customers where customer_id = ?`

Saat menganalisis performa basis data, sebaiknya pertimbangkan pernyataan SQL yang hanya berbeda dari segi parameternya sebagai satu item logika. Jadi, Anda dapat menggunakan `db.sql_tokenized` saat melakukan kueri. Namun, terutama ketika Anda tertarik untuk menjelaskan rencana, terkadang lebih berguna untuk memeriksa pernyataan SQL lengkap beserta parameter, dan pengelompokan kueri berdasarkan `db.sql`. Ada hubungan induk-turunan antara SQL token dan lengkap, dengan beberapa SQL lengkap (turunan) yang dikelompokkan dalam SQL token (induk) yang sama.

Perintah dalam contoh ini terlihat seperti perintah dalam Mengambil rata-rata muatan DB untuk peristiwa tunggu teratas. Namun, file query.json berisi konten berikut.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.sql_tokenized", "Limit": 10 }
    }
]
```

Contoh berikut menggunakan `db.sql_tokenized`.

Untuk Linux, macOS, atau Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-29T00:00:00Z \
   --end-time   2018-10-30T00:00:00Z \
```

```
   --period-in-seconds 3600 \
   --metric-queries file://query.json
```

Untuk Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-29T00:00:00Z ^
   --end-time   2018-10-30T00:00:00Z  ^
   --period-in-seconds 3600 ^
   --metric-queries file://query.json
```

Contoh ini menanyakan lebih dari 24 jam, dengan satu jam period-in-seconds.

Contoh ini menentukan metrik `db.load.avg` dan `GroupBy` dari tujuh peristiwa tunggu teratas. Untuk detail tentang nilai yang valid untuk contoh ini, lihat DimensionGroup dalam Referensi API Wawasan Kinerja.

Responsnya terlihat seperti berikut.

```
{
    "AlignedStartTime": 1540771200.0,
    "AlignedEndTime": 1540857600.0,
    "Identifier": "db-XXX",

    "MetricList": [ //11 entries in the MetricList
        {
            "Key": { //First key is total
                "Metric": "db.load.avg"
            }
            "DataPoints": [ //Each DataPoints list has 24 per-hour Timestamps and a
 value
                {
                    "Value": 1.6964980544747081,
                    "Timestamp": 1540774800.0
                },
                //... 24 datapoints
            ]
        },
        {
            "Key": { //Next key is the top tokenized SQL
```

```
                "Dimensions": {
                    "db.sql_tokenized.statement": "INSERT INTO authors (id,name,email)
 VALUES\n( nextval(?)  ,?,?)",
                    "db.sql_tokenized.db_id": "pi-2372568224",
                    "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE"
                },
                "Metric": "db.load.avg"
            },
            "DataPoints": [ //... 24 datapoints
            ]
        },
        // In total 11 entries, 10 Keys of top tokenized SQL, 1 total key
    ] //End of MetricList
} //End of response
```

Respons ini memiliki 11 entri di `MetricList` (1 total, 10 SQL token teratas), dengan setiap entri memiliki 24 `DataPoints` per jam.

Untuk SQL token, ada tiga entri di setiap daftar dimensi:

- `db.sql_tokenized.statement` – Pernyataan SQL token.
- `db.sql_tokenized.db_id` – Baik ID basis data native yang digunakan untuk merujuk ke SQL, maupun ID sintetis yang dihasilkan oleh Wawasan Performa untuk Anda jika ID basis data native tidak tersedia. Contoh ini menampilkan ID sintetis `pi-2372568224`.
- `db.sql_tokenized.id` – ID kueri di dalam Wawasan Performa.

  Di AWS Management Console, ID ini disebut sebagai ID Dukungan. Disebut demikian karena ID ini adalah data yang dapat diperiksa oleh Dukungan AWS untuk membantu memecahkan masalah dalam basis data Anda. AWS menangani keamanan dan privasi data Anda dengan sangat serius, dan hampir semua data disimpan terenkripsi dengan kunci utama pelanggan (CMK) AWS KMS. Oleh karena itu, tidak ada orang di dalam AWS yang dapat melihat data ini. Di contoh sebelumnya, baik `tokenized.statement` maupun `tokenized.db_id` disimpan dengan enkripsi. Jika Anda mengalami masalah terkait basis data, Dukungan AWS dapat membantu Anda dengan merujuk ID Dukungan.

Ketika melakukan kueri, mungkin lebih mudah untuk menentukan `Group` dalam `GroupBy`. Namun, untuk kontrol lebih mendetail atas data yang ditampilkan, tentukan daftar dimensi. Misalnya, jika yang dibutuhkan hanya `db.sql_tokenized.statement`, atribut `Dimensions` dapat ditambahkan ke file query.json.

```
[
    {
        "Metric": "db.load.avg",
        "GroupBy": {
            "Group": "db.sql_tokenized",
            "Dimensions":["db.sql_tokenized.statement"],
            "Limit": 10
        }
    }
]
```

Mengambil Rata-Rata Muatan DB yang difilter berdasarkan SQL

Gambar sebelumnya menunjukkan bahwa kueri tertentu dipilih, dan grafik baris area bertumpuk sesi aktif rata-rata teratas dicakup ke kueri tersebut. Meskipun kueri masih diperuntukkan bagi tujuh peristiwa tunggu teratas secara keseluruhan, nilai responsnya akan difilter. Filter menyebabkannya hanya memperhitungkan sesi yang cocok untuk filter tertentu.

Kueri API terkait dalam contoh ini sama seperti perintah di Mengambil rata-rata muatan DB untuk SQL teratas. Namun, file query.json berisi konten berikut.

```
[
```

```
 {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 5  },
        "Filter": { "db.sql_tokenized.id": "AKIAIOSFODNN7EXAMPLE" }
    }
]
```

Untuk Linux, macOS, atau Unix:

```
aws pi get-resource-metrics \
   --service-type RDS \
   --identifier db-ID \
   --start-time 2018-10-30T00:00:00Z \
   --end-time   2018-10-30T01:00:00Z \
   --period-in-seconds 60 \
   --metric-queries file://query.json
```

Untuk Windows:

```
aws pi get-resource-metrics ^
   --service-type RDS ^
   --identifier db-ID ^
   --start-time 2018-10-30T00:00:00Z ^
   --end-time   2018-10-30T01:00:00Z ^
   --period-in-seconds 60 ^
   --metric-queries file://query.json
```

Responsnya terlihat seperti berikut.

```
{
    "Identifier": "db-XXX",
    "AlignedStartTime": 1556215200.0,
    "MetricList": [
        {
            "Key": {
                "Metric": "db.load.avg"
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.4878117913832196
                },
```

```
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.192823803967328
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/aurora_redo_log_flush"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 1.1360544217687074
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 1.058051341890315
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "io",
                    "db.wait_event.name": "wait/io/table/sql/handler"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.16241496598639457
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05163360560093349
                }
            ]
        },
```

```
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
                    "db.wait_event.name": "wait/synch/mutex/innodb/
aurora_lock_thread_slot_futex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.11479591836734694
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.013127187864644107
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "CPU",
                    "db.wait_event.name": "CPU"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.05215419501133787
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.05805134189031505
                }
            ]
        },
        {
            "Key": {
                "Metric": "db.load.avg",
                "Dimensions": {
                    "db.wait_event.type": "synch",
```

```
                    "db.wait_event.name": "wait/synch/mutex/innodb/lock_wait_mutex"
                }
            },
            "DataPoints": [
                {
                    "Timestamp": 1556218800.0,
                    "Value": 0.017573696145124718
                },
                {
                    "Timestamp": 1556222400.0,
                    "Value": 0.002333722287047841
                }
            ]
        }
    ],
    "AlignedEndTime": 1556222400.0
} //end of response
```

Dalam respons ini, semua nilai difilter sesuai dengan kontribusi AKIAIOSFODNN7EXAMPLE SQL token yang ditentukan dalam file query.json. Kunci mungkin juga mengikuti urutan yang berbeda dari kueri tanpa filter, karena lima peristiwa tunggu teratas tersebutlah yang memengaruhi SQL yang difilter.

Mengambil teks lengkap pernyataan SQL

Contoh berikut mengambil teks lengkap pernyataan SQL untuk instans DB `db-10BCD2EFGHIJ3KL4M5NO6PQRS5`. `--group` adalah `db.sql`, dan `--group-identifier` adalah `db.sql.id`. Dalam contoh ini, *`my-sql-id`*merupakan ID SQL diambil dengan memanggil `pi get-resource-metrics` atau. `pi describe-dimension-keys`

Jalankan perintah berikut.

Untuk Linux, macOS, atau Unix:

```
aws pi get-dimension-key-details \
   --service-type RDS \
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 \
   --group db.sql \
   --group-identifier my-sql-id \
   --requested-dimensions statement
```

Untuk Windows:

```
aws pi get-dimension-key-details ^
   --service-type RDS ^
   --identifier db-10BCD2EFGHIJ3KL4M5NO6PQRS5 ^
   --group db.sql ^
   --group-identifier my-sql-id ^
   --requested-dimensions statement
```

Dalam contoh ini, detail dimensinya tersedia. Dengan demikian, Wawasan Performa mengambil teks lengkap pernyataan SQL, tanpa memotongnya.

```
{
    "Dimensions":[
    {
        "Value": "SELECT e.last_name, d.department_name FROM employees e, departments d
 WHERE e.department_id=d.department_id",
        "Dimension": "db.sql.statement",
        "Status": "AVAILABLE"
    },
    ...
    ]
}
```

Membuat laporan analisis performa selama periode waktu tertentu

Contoh berikut membuat laporan analisis performa dengan waktu mulai `1682969503` dan waktu akhir `1682979503` untuk basis data `db-loadtest-0`.

```
        aws pi-test create-performance-analysis-report \
        --service-type RDS \
        --identifier db-loadtest-0 \
        --start-time 1682969503 \
        --end-time 1682979503 \
        --endpoint-url https://api.titan.pi.a2z.com \
        --region us-west-2
```

Responsnya adalah pengidentifikasi unik `report-0234d3ed98e28fb17` untuk laporan tersebut.

```
      {
         "AnalysisReportId": "report-0234d3ed98e28fb17"
      }
```

Mengambil laporan analisis performa

Contoh berikut mengambil detail laporan analisis untuk laporan `report-0d99cc91c4422ee61`.

```
        aws pi-test get-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--endpoint-url https://api.titan.pi.a2z.com \
--region us-west-2
```

Respons-nya menampilkan status laporan, ID, detail waktu, dan wawasan.

```
        {
    "AnalysisReport": {
        "Status": "Succeeded",
        "ServiceType": "RDS",
        "Identifier": "db-loadtest-0",
        "StartTime": 1680583486.584,
        "AnalysisReportId": "report-0d99cc91c4422ee61",
        "EndTime": 1680587086.584,
        "CreateTime": 1680587087.139,
        "Insights": [
           ... (Condensed for space)
        ]
    }
}
```

Daftar semua laporan analisis performa untuk instans DB

Contoh berikut mencantumkan semua laporan analisis performa yang tersedia untuk basis data `db-loadtest-0`.

```
            aws pi-test list-performance-analysis-reports \
--service-type RDS \
--identifier db-loadtest-0 \
--endpoint-url https://api.titan.pi.a2z.com \
--region us-west-2
```

Respons ini mencantumkan semua laporan dengan ID laporan, status, dan detail periode waktu.

```
            {
    "AnalysisReports": [
        {
            "Status": "Succeeded",
            "EndTime": 1680587086.584,
            "CreationTime": 1680587087.139,
            "StartTime": 1680583486.584,
            "AnalysisReportId": "report-0d99cc91c4422ee61"
        },
        {
            "Status": "Succeeded",
            "EndTime": 1681491137.914,
            "CreationTime": 1681491145.973,
            "StartTime": 1681487537.914,
            "AnalysisReportId": "report-002633115cc002233"
        },
        {
            "Status": "Succeeded",
            "EndTime": 1681493499.849,
            "CreationTime": 1681493507.762,
            "StartTime": 1681489899.849,
            "AnalysisReportId": "report-043b1e006b47246f9"
        },
        {
            "Status": "InProgress",
            "EndTime": 1682979503.0,
            "CreationTime": 1682979618.994,
            "StartTime": 1682969503.0,
            "AnalysisReportId": "report-01ad15f9b88bcbd56"
        }
    ]
}
```

Menghapus laporan analisis performa

Contoh berikut menghapus laporan analisis untuk basis data `db-loadtest-0`.

```
       aws pi-test delete-performance-analysis-report \
--service-type RDS \
--identifier db-loadtest-0 \
--analysis-report-id report-0d99cc91c4422ee61 \
--endpoint-url https://api.titan.pi.a2z.com \
--region us-west-2
```

Menambahkan tag ke laporan analisis performa

Contoh berikut menambahkan tag dengan kunci `name` dan nilai `test-tag` ke laporan `report-01ad15f9b88bcbd56`.

```
           aws pi-test tag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tags Key=name,Value=test-tag \
--endpoint-url https://api.titan.pi.a2z.com \
--region us-west-2
```

Mencantumkan semua tag untuk laporan analisis performa

Contoh berikut mencantumkan semua tag untuk laporan `report-01ad15f9b88bcbd56`.

```
       aws pi-test list-tags-for-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--endpoint-url https://api.titan.pi.a2z.com \
--region us-west-2
```

Respons ini mencantumkan nilai dan kunci untuk semua tag yang ditambahkan ke laporan:

```
        {
    "Tags": [
        {
            "Value": "test-tag",
            "Key": "name"
        }
    ]
}
```

Menghapus tag dari laporan analisis performa

Contoh berikut menghapus tag `name` dari laporan `report-01ad15f9b88bcbd56`.

```
        aws pi-test untag-resource \
--service-type RDS \
--resource-arn arn:aws:pi:us-west-2:356798100956:perf-reports/RDS/db-loadtest-0/
report-01ad15f9b88bcbd56 \
--tag-keys name \
--endpoint-url https://api.titan.pi.a2z.com \
--region us-west-2
```

Setelah tag dihapus, pemanggilan API `list-tags-for-resource` tidak akan mencantumkan tag ini.

Mencatat panggilan Wawasan Performa menggunakan AWS CloudTrail

Wawasan Performa berjalan dengan AWS CloudTrail, layanan yang memberikan data tindakan yang dilakukan oleh pengguna, peran, atau layanan AWS di Wawasan Performa. CloudTrail mengambil semua panggilan API untuk Wawasan Performa sebagai peristiwa. Pengambilan ini mencakup panggilan dari konsol Amazon RDS dan dari panggilan kode ke operasi API Wawasan Performa.

Jika membuat jejak, Anda dapat mengaktifkan pengiriman peristiwa CloudTrail berkelanjutan ke bucket Amazon S3, termasuk peristiwa untuk Wawasan Performa. Jika tidak mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru di konsol CloudTrail dalam Riwayat peristiwa. Dengan data yang dikumpulkan oleh CloudTrail, Anda dapat menentukan informasi tertentu. Informasi ini mencakup permintaan yang dibuat untuk Wawasan Performa, alamat IP asal permintaan, siapa yang

membuat permintaan, dan kapan permintaan tersebut dibuat. Informasi ini juga mencakup detail tambahan.

Untuk mempelajari selengkapnya tentang CloudTrail, lihat Panduan Pengguna AWS CloudTrail.

Menggunakan informasi Wawasan Performa di CloudTrail

CloudTrail diaktifkan pada akun AWS saat Anda membuat akun tersebut. Saat aktivitas terjadi di Wawasan Performa, aktivitas tersebut dicatat dalam peristiwa CloudTrail beserta peristiwa layanan AWS lain di konsol CloudTrail dalam Riwayat peristiwa. Anda dapat melihat, mencari, dan mengunduh peristiwa terbaru di akun AWS. Untuk informasi selengkapnya, lihat Melihat Peristiwa dengan Riwayat Peristiwa CloudTrail di Panduan Pengguna AWS CloudTrail.

Untuk data peristiwa yang sedang berlangsung di akun AWS Anda, termasuk peristiwa untuk Wawasan Performa, buatlah jejak. Jejak memungkinkan CloudTrail mengirimkan file log ke bucket Amazon S3. Secara default, saat Anda membuat jejak di konsol tersebut, jejak diterapkan ke semua Wilayah AWS. Jejak mencatat peristiwa dari semua Wilayah AWS di sebagian AWS dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi layanan AWS lainnya untuk dianalisis lebih lanjut dan bertindak berdasarkan data peristiwa yang dikumpulkan di log CloudTrail. Untuk informasi selengkapnya, lihat topik berikut di Panduan Pengguna AWS CloudTrail:

- Ikhtisar Pembuatan Jejak
- Layanan dan Integrasi yang Didukung CloudTrail
- Mengonfigurasi Pemberitahuan Amazon SNS untuk CloudTrail
- Menerima File Log CloudTrail dari Beberapa Wilayah dan Menerima File log CloudTrail dari Beberapa Akun

Semua operasi Wawasan Performa dicatat oleh CloudTrail dan didokumentasikan dalam Referensi API Wawasan Performa. Misalnya, panggilan ke operasi `DescribeDimensionKeys` dan `GetResourceMetrics` menghasilkan entri dalam file log CloudTrail.

Setiap peristiwa atau entri log berisi informasi tentang siapa yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan berikut:

- Apakah permintaan tersebut dibuat dengan kredensial root atau pengguna IAM.
- Apakah permintaan dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna gabungan.

- Apakah permintaan tersebut dibuat oleh layanan AWS lain.

Untuk informasi selengkapnya, lihat Elemen userIdentity CloudTrail.

Entri file log Wawasan Performa

Jejak adalah konfigurasi yang memungkinkan pengiriman peristiwa sebagai file log ke bucket Amazon S3 yang Anda tentukan. File log CloudTrail berisi satu atau beberapa entri log. Peristiwa menunjukkan satu permintaan dari sumber mana pun. Setiap peristiwa mencakup informasi tentang operasi yang diminta, tanggal dan waktu operasi, parameter permintaan, dan sebagainya. File log CloudTrail bukanlah jejak tumpukan yang berurutan dari panggilan API publik, sehingga file tersebut tidak muncul dalam urutan tertentu.

Contoh berikut menunjukkan entri log CloudTrail yang menunjukkan operasi `GetResourceMetrics`.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
         "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T19:28:46Z",
    "eventSource": "pi.amazonaws.com",
    "eventName": "GetResourceMetrics",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "72.21.198.67",
    "userAgent": "aws-cli/1.16.240 Python/3.7.4 Darwin/18.7.0 botocore/1.12.230",
    "requestParameters": {
        "identifier": "db-YTDU5J5V66X7CXSCVDFD2V3SZM",
        "metricQueries": [
            {
                "metric": "os.cpuUtilization.user.avg"
            },
            {
                "metric": "os.cpuUtilization.idle.avg"
            }
        ],
        "startTime": "Dec 18, 2019 5:28:46 PM",
```

```
        "periodInSeconds": 60,
        "endTime": "Dec 18, 2019 7:28:46 PM",
        "serviceType": "RDS"
    },
    "responseElements": null,
    "requestID": "9ffbe15c-96b5-4fe6-bed9-9fccff1a0525",
    "eventID": "08908de0-2431-4e2e-ba7b-f5424f908433",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

Menganalisis anomali kinerja dengan Amazon DevOps Guru untuk Amazon RDS

Amazon DevOps Guru adalah layanan operasi yang dikelola sepenuhnya yang membantu pengembang dan operator meningkatkan kinerja dan ketersediaan aplikasi mereka. DevOpsGuru menurunkan tugas yang terkait dengan mengidentifikasi masalah operasional sehingga Anda dapat dengan cepat menerapkan rekomendasi untuk meningkatkan aplikasi Anda. Untuk informasi selengkapnya, lihat Apa itu Amazon DevOps Guru? di Panduan Pengguna Amazon DevOps Guru.

DevOpsGuru mendeteksi, menganalisis, dan membuat rekomendasi untuk masalah operasional yang ada untuk semua mesin Amazon RDS DB. DevOps Fitur pemantauan ini memungkinkan DevOps Guru for RDS mendeteksi dan mendiagnosis kemacetan kinerja dan merekomendasikan tindakan korektif tertentu. DevOpsGuru untuk RDS juga dapat mendeteksi kondisi bermasalah di database Aurora Anda sebelum terjadi.

Anda sekarang dapat melihat rekomendasi ini di konsol RDS. Untuk informasi selengkapnya, lihat Melihat dan menanggapi rekomendasi Amazon RDS.

Video berikut adalah ikhtisar DevOps Guru untuk RDS.

Untuk menyelam lebih dalam tentang hal ini, lihat Amazon DevOps Guru untuk RDS di bawah tenda.

Topik

- Manfaat DevOps Guru untuk RDS
- Bagaimana DevOps Guru untuk RDS bekerja
- Menyiapkan DevOps Guru untuk RDS

Manfaat DevOps Guru untuk RDS

Jika Anda bertanggung jawab atas sebuah basis data Amazon Aurora, Anda mungkin tidak tahu bahwa sedang terjadi suatu peristiwa atau regresi yang memengaruhi basis data itu. Ketika mengetahui masalah ini, Anda mungkin tidak tahu alasannya terjadi atau apa yang harus dilakukan terhadapnya. Daripada beralih ke administrator database (DBA) untuk bantuan atau mengandalkan alat pihak ketiga, Anda dapat mengikuti rekomendasi dari DevOps Guru untuk RDS.

Anda mendapatkan keuntungan berikut dari analisis rinci DevOps Guru untuk RDS:

Diagnosis cepat

DevOpsGuru untuk RDS terus memantau dan menganalisis telemetri database. DevOpsGuru untuk RDS menggunakan teknik statistik dan pembelajaran mesin untuk menambang data ini dan mendeteksi anomali. Untuk mempelajari lebih lanjut data telemetri, lihat Memantau beban basis data dengan Wawasan Performa di Amazon Aurora dan Memantau metrik-metrik OS dengan Pemantauan Disempurnakan dalam Panduan Pengguna Amazon Aurora.

Resolusi cepat

Setiap anomali mengidentifikasi masalah kinerja dan menyarankan alur investigasi atau tindakan korektif. Misalnya, DevOps Guru untuk RDS mungkin menyarankan Anda menyelidiki peristiwa menunggu tertentu. Atau mungkin menyarankan agar Anda menyetel pengaturan kumpulan aplikasi Anda untuk membatasi jumlah koneksi basis data. Berdasarkan rekomendasi ini, Anda dapat menyelesaikan masalah kinerja lebih cepat daripada dengan memecahkan masalah secara manual.

Wawasan proaktif

DevOpsGuru untuk RDS menggunakan metrik dari sumber daya Anda untuk mendeteksi perilaku yang berpotensi bermasalah sebelum menjadi masalah yang lebih besar. Misalnya, fitur ini dapat mendeteksi ketika basis data Anda menggunakan makin banyak tabel sementara pada disk, yang dapat mulai mempengaruhi kinerja. DevOpsGuru kemudian memberikan rekomendasi untuk membantu Anda mengatasi masalah sebelum menjadi masalah yang lebih besar.

Pengetahuan mendalam insinyur Amazon dan pembelajaran mesin

Untuk mendeteksi masalah kinerja dan membantu Anda mengatasi kemacetan, DevOps Guru for RDS mengandalkan pembelajaran mesin (ML) dan rumus matematika tingkat lanjut. Insinyur basis data Amazon berkontribusi pada pengembangan temuan DevOps Guru untuk RDS, yang merangkum bertahun-tahun mengelola ratusan ribu database. Dengan memanfaatkan pengetahuan kolektif ini, DevOps Guru untuk RDS dapat mengajari Anda praktik terbaik.

Bagaimana DevOps Guru untuk RDS bekerja

DevOpsGuru for RDS mengumpulkan data tentang database Aurora RDS Anda dari Amazon RDS Performance Insights. Metrik yang paling penting adalah`DBLoad`. DevOpsGuru for RDS menggunakan metrik Performance Insights, menganalisisnya dengan pembelajaran mesin, dan menerbitkan wawasan ke dasbor.

Wawasan adalah kumpulan anomali terkait yang terdeteksi oleh DevOps Guru.

Wawasan proaktif

Wawasan proaktif memberi tahu Anda tentang perilaku bermasalah sebelum menimbulkan masalah. Wawasan berisi anomali dengan rekomendasi dan metrik terkait untuk membantu Anda mengatasi masalah di basis data Amazon Aurora sebelum menjadi masalah yang lebih besar. Wawasan ini dipublikasikan di dasbor DevOps Guru.

Misalnya, DevOps Guru mungkin mendeteksi bahwa database Aurora PostgreSQL Anda membuat banyak tabel sementara di disk. Jika tidak ditangani, tren ini dapat menyebabkan masalah kinerja. Setiap wawasan proaktif mencakup rekomendasi untuk perilaku korektif dan penaut ke topik yang relevan di Menyesuaikan Aurora MySQL dengan wawasan proaktif Amazon DevOps Guru atau Menyesuaikan Aurora PostgreSQL dengan wawasan proaktif Amazon DevOps Guru. Untuk informasi selengkapnya, lihat Bekerja dengan wawasan di DevOps Guru di Panduan Pengguna Amazon DevOps Guru.

Wawasan reaktif

Wawasan reaktif mengidentifikasi perilaku anomali saat terjadi. Jika DevOps Guru for RDS menemukan masalah kinerja di Amazon instans PostgreSQL DB, Guru akan menerbitkan wawasan reaktif di dasbor Guru. DevOps Untuk informasi selengkapnya, lihat Bekerja dengan wawasan di DevOps Guru di Panduan Pengguna Amazon DevOps Guru.

Anomali kausal

Anomali kausal adalah anomali tingkat puncak dalam wawasan reaktif. Beban basis data (beban DB) adalah anomali kausal untuk DevOps Guru untuk RDS.

Anomali mengukur dampak kinerja dengan menetapkan tingkat keparahan Tinggi, Sedang, atau Rendah. Untuk mempelajari lebih lanjut, lihat Konsep kunci untuk DevOps Guru for RDS di Panduan Pengguna Amazon DevOps Guru.

Jika DevOps Guru mendeteksi anomali saat ini pada instans DB Anda, Anda akan diberi tahu di halaman Database konsol RDS. Konsol juga memperingatkan Anda tentang anomali yang terjadi dalam 24 jam terakhir. Untuk menuju halaman anomali dari konsol RDS, pilih penaut dalam pesan peringatan. Konsol RDS juga memperingatkan Anda di halaman itu untuk klaster basis data Amazon Aurora.

Anomali kontekstual

Anomali kontekstual adalah temuan dalam Beban basis data (Beban DB) yang terkait dengan wawasan reaktif. Setiap anomali kontekstual menjelaskan masalah kinerja Amazon Aurora tertentu yang memerlukan penyelidikan. Misalnya, DevOps Guru untuk RDS mungkin menyarankan Anda mempertimbangkan untuk meningkatkan kapasitas CPU atau menyelidiki peristiwa tunggu yang berkontribusi pada pemuatan DB.

> **Important**
>
> Sebaiknya uji setiap perubahan pada instans uji sebelum diterapkan pada instans produksi. Dengan cara ini, Anda memahami dampak perubahan.

Untuk mempelajari lebih lanjut, lihat Menganalisis anomali di Amazon RDS di Panduan Pengguna Amazon DevOps Guru.

Menyiapkan DevOps Guru untuk RDS

Untuk mengizinkan DevOps Guru for Amazon RDS mempublikasikan wawasan untuk Amazon Aurora, RDS untuk database , selesaikan tugas-tugas berikut.

Topik

- Mengkonfigurasi kebijakan akses IAM untuk DevOps Guru untuk RDS
- Mengaktifkan Wawasan Performa untuk instans basis data Aurora Anda
- Mengaktifkan DevOps Guru dan menentukan cakupan sumber daya

Mengkonfigurasi kebijakan akses IAM untuk DevOps Guru untuk RDS

Untuk melihat peringatan dari DevOps Guru di konsol RDS, pengguna atau peran AWS Identity and Access Management (IAM) Anda harus memiliki salah satu dari kebijakan berikut:

- Kebijakan terkelola AWS `AmazonDevOpsGuruConsoleFullAccess`
- Kebijakan terkelola AWS `AmazonDevOpsGuruConsoleReadOnlyAccess` dan salah satu kebijakan berikut:
  - Kebijakan terkelola AWS `AmazonRDSFullAccess`
  - Kebijakan terkelola pelanggan yang mencakup `pi:GetResourceMetrics` dan `pi:DescribeDimensionKeys`

Untuk informasi selengkapnya, lihat Mengonfigurasi kebijakan akses untuk Wawasan Performa.

Mengaktifkan Wawasan Performa untuk instans basis data Aurora Anda

DevOpsGuru untuk RDS mengandalkan Performance Insights untuk datanya. Tanpa Performance Insights, DevOps Guru menerbitkan anomali, tetapi tidak menyertakan analisis dan rekomendasi terperinci.

Saat membuat klaster basis data Aurora atau mengubah instans klaster, Anda dapat mengaktifkan Wawasan Performa. Untuk informasi selengkapnya, lihat Mengaktifkan dan menonaktifkan Wawasan Performa.

Mengaktifkan DevOps Guru dan menentukan cakupan sumber daya

Anda dapat mengaktifkan DevOps Guru agar memonitor Amazon Aurora RDS .

Topik

- Menghidupkan DevOps Guru di konsol RDS
- Menambahkan Aurora RDS di konsol Guru DevOps
- Menambahkan sumber daya Aurora dengan menggunakan AWS CloudFormation

Menghidupkan DevOps Guru di konsol RDS

Anda dapat mengambil beberapa jalur di konsol Amazon RDS untuk mengaktifkan DevOps Guru.

Topik

- Mengaktifkan DevOps Guru saat Anda membuat Aurora database PostgreSQL
- Menghidupkan DevOps Guru dari spanduk notifikasi
- Menanggapi kesalahan izin saat Anda mengaktifkan Guru DevOps

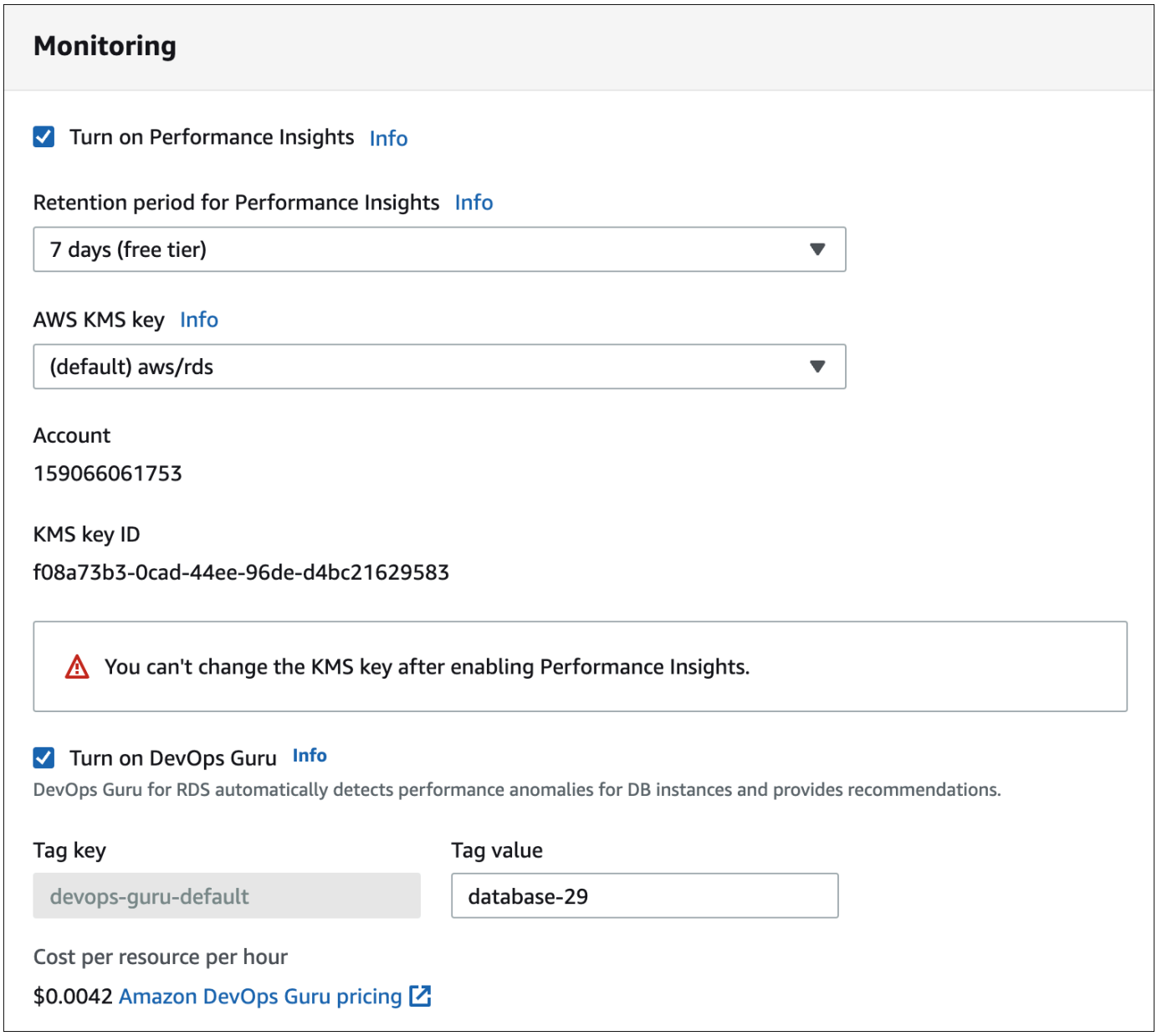
Mengaktifkan DevOps Guru saat Anda membuat Aurora database PostgreSQL

Alur kerja pembuatan mencakup pengaturan yang mengaktifkan cakupan DevOps Guru untuk database Anda. Pengaturan ini diaktifkan secara bawaan saat Anda memilih templat Produksi.

Untuk mengaktifkan DevOps Guru saat Anda membuat Aurora database PostgreSQL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

2. Ikuti langkah-langkah di Membuat klaster DB, sampai tetapi tidak meliputi langkah ketika Anda memilih setelan pemantauan.
3. Di Pemantauan, pilih Aktifkan Wawasan Performa. Agar DevOps Guru for RDS dapat memberikan analisis terperinci tentang anomali kinerja, Performance Insights harus diaktifkan.
4. Pilih Aktifkan DevOps Guru.



5. Buat tag untuk database Anda sehingga DevOps Guru dapat memantaunya. Lakukan hal-hal berikut:
   - Di bidang teks untuk Kunci tag, masukkan nama yang dimulai dengan **`Devops-Guru-`**.

- Di bidang teks untuk Nilai tag, masukkan nilai apa pun. Misalnya, jika Anda memasukkan **`rds-database-1`** untuk nama basis data Aurora, Anda juga dapat memasukkan **`rds-database-1`** sebagai nilai tag.

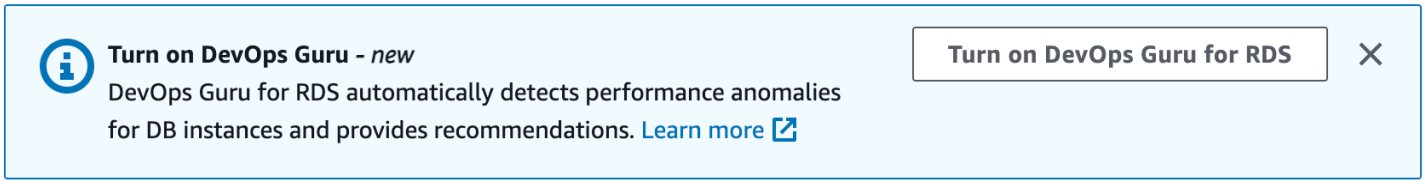
Untuk informasi selengkapnya tentang tag, lihat "Menggunakan tag untuk mengidentifikasi sumber daya dalam aplikasi DevOps Guru Anda" di Panduan Pengguna Amazon DevOps Guru.

6. Selesaikan langkah-langkah selebihnya di Membuat klaster DB.

Menghidupkan DevOps Guru dari spanduk notifikasi

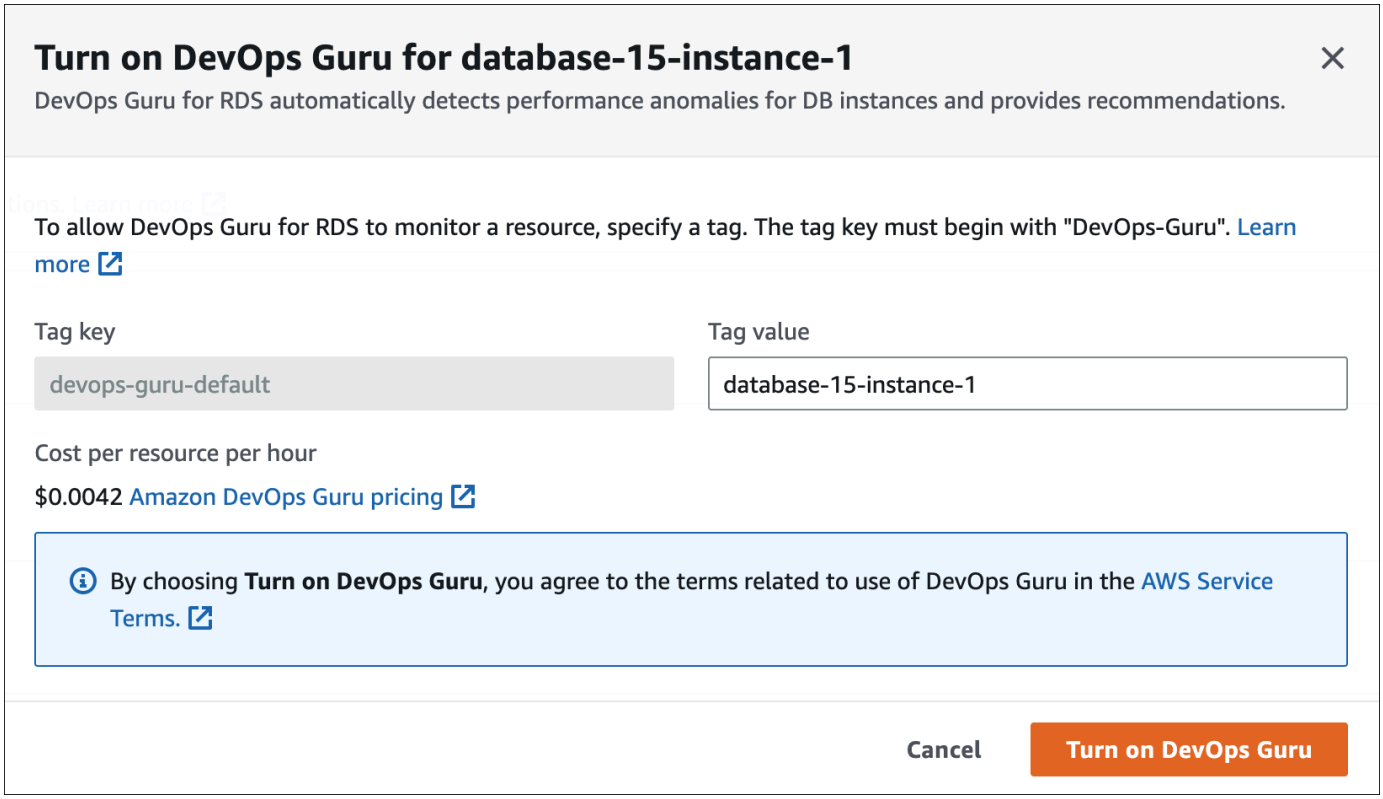
Jika sumber daya Anda tidak dicakup oleh DevOps Guru, Amazon RDS akan memberi tahu Anda dengan spanduk di lokasi berikut:

- Tab Pemantauan instans klaster basis data
- Dasbor Wawasan Performa



Untuk mengaktifkan DevOps Guru untuk Aurora Anda untuk database PostgreSQL

1. Di spanduk, pilih Aktifkan DevOps Guru untuk RDS.
2. Masukkan nama kunci dan nilai tag. Untuk informasi selengkapnya tentang tag, lihat "Menggunakan tag untuk mengidentifikasi sumber daya dalam aplikasi DevOps Guru Anda" di Panduan Pengguna Amazon DevOps Guru.



3. Pilih Aktifkan DevOps Guru.

Menanggapi kesalahan izin saat Anda mengaktifkan Guru DevOps

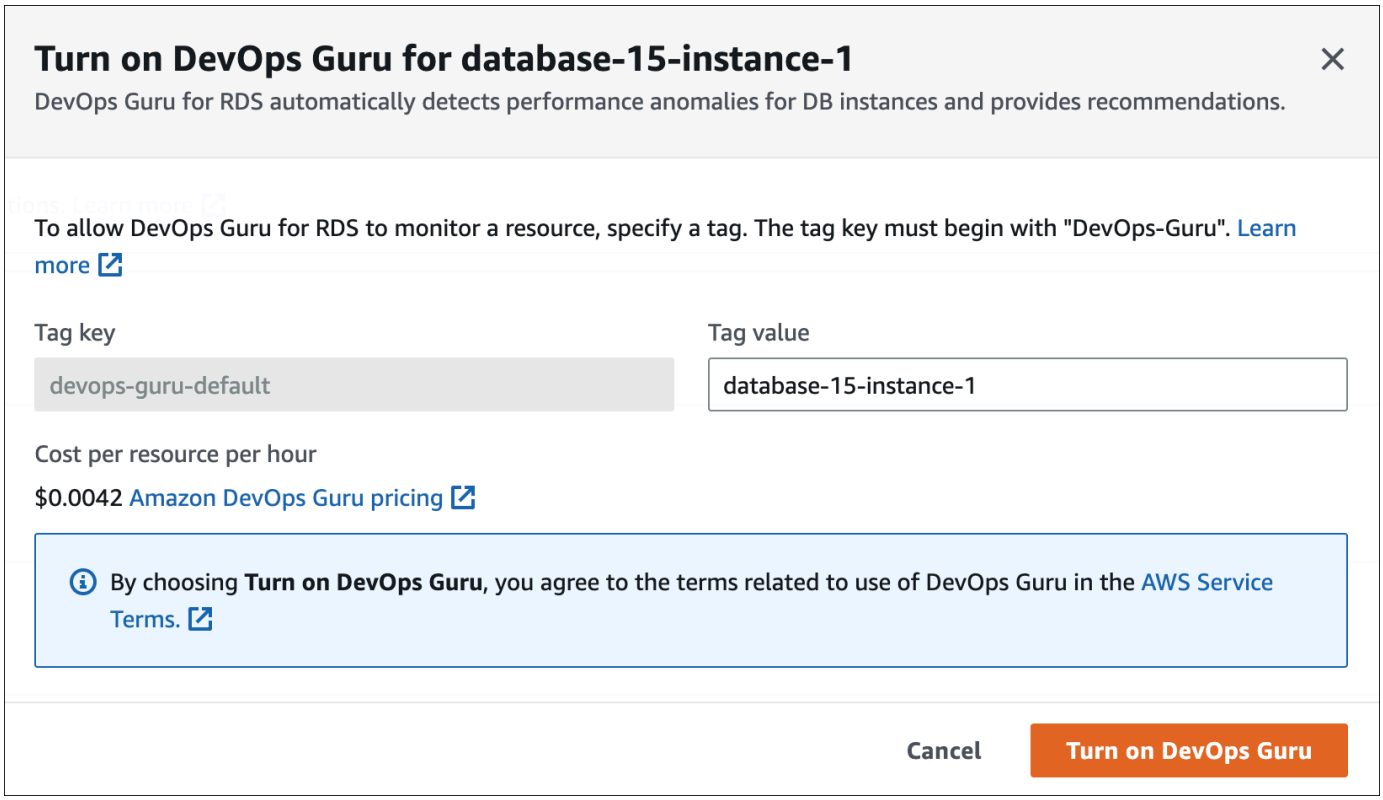
Jika Anda mengaktifkan DevOps Guru dari konsol RDS saat membuat database, RDS mungkin menampilkan spanduk berikut tentang izin yang hilang.

Untuk menanggapi kesalahan izin

1. Beri peran atau pengguna IAM Anda peran terkelola pengguna `AmazonDevOpsGuruConsoleFullAccess`. Untuk informasi selengkapnya, lihat Mengkonfigurasi kebijakan akses IAM untuk DevOps Guru untuk RDS.
2. Buka konsol .
3. Di panel navigasi, pilih Wawasan Performa.
4. Pilih instans basis data di klaster yang baru saja Anda buat.
5. Nyalakan DevOps Guru untuk RDS.

DevOps Guru for RDS

6. Pilih nilai tag. Untuk informasi selengkapnya, lihat "Menggunakan tag untuk mengidentifikasi sumber daya di aplikasi DevOps Guru Anda" di Panduan Pengguna Amazon DevOps Guru.



7. Pilih Aktifkan DevOps Guru.

Menambahkan Aurora RDS di konsol Guru DevOps

Anda dapat menentukan cakupan sumber daya DevOps Guru Anda di konsol DevOps Guru. Ikuti langkah yang dijelaskan dalam Tentukan cakupan sumber daya DevOps Guru Anda di Panduan Pengguna Amazon DevOps Guru. Saat Anda mengedit sumber daya yang dianalisis, pilih salah satu opsi berikut:

- Pilih Semua sumber daya akun untuk menganalisis semua sumber daya yang didukung, yang meliputi basis data Aurora, di Akun AWS dan Kawasan Anda.
- Pilih CloudFormation tumpukan untuk menganalisis Aurora database PostgreSQL yang ada di tumpukan yang Anda pilih. Untuk informasi selengkapnya, lihat Menggunakan AWS CloudFormation tumpukan untuk mengidentifikasi sumber daya dalam aplikasi DevOps Guru Anda di Panduan Pengguna Amazon DevOps Guru.

- Pilih Tag untuk menganalisis basis data Aurora yang telah Anda beri tag. Untuk informasi selengkapnya, lihat Menggunakan tag untuk mengidentifikasi sumber daya dalam aplikasi DevOps Guru Anda di Panduan Pengguna Amazon DevOps Guru.

Untuk informasi selengkapnya, lihat Aktifkan DevOps DevOps Guru di Panduan Pengguna Amazon Guru.

Menambahkan sumber daya Aurora dengan menggunakan AWS CloudFormation

Anda dapat menggunakan tag untuk menambahkan cakupan untuk Aurora ke template Anda. CloudFormation Prosedur berikut mengasumsikan bahwa Anda memiliki CloudFormation template baik untuk Aurora dan tumpukan Guru. DevOps

Untuk menentukan Aurora DB instance menggunakan tag CloudFormation

1. Dalam CloudFormation template untuk instans DB Anda, tentukan tag menggunakan pasangan kunci/nilai.

   Contoh berikut menetapkan nilai `my-aurora-db-instance1` untuk `Devops-guru-cfn-default` bagi instans basis data Aurora.

   ```
   MyAuroraDBInstance1:
     Type: "AWS::RDS::DBInstance"
     Properties:
       DBClusterIdentifier: my-aurora-db-cluster
       DBInstanceIdentifier: my-aurora-db-instance1
       Tags:
         - Key: Devops-guru-cfn-default
           Value: devopsguru-my-aurora-db-instance1
   ```

2. Dalam CloudFormation template untuk tumpukan DevOps Guru Anda, tentukan tag yang sama di filter pengumpulan sumber daya Anda.

   Contoh berikut mengkonfigurasi DevOps Guru untuk menyediakan cakupan sumber daya dengan nilai `my-aurora-db-instance1` tag.

   ```
   DevOpsGuruResourceCollection:
     Type: AWS::DevOpsGuru::ResourceCollection
     Properties:
       ResourceCollectionFilter:
         Tags:
   ```

```
        - AppBoundaryKey: "Devops-guru-cfn-default"
          TagValues:
          - "devopsguru-my-aurora-db-instance1"
```

Contoh berikut menyediakan cakupan untuk semua sumber daya dalam batas aplikasi `Devops-guru-cfn-default`.

```
DevOpsGuruResourceCollection:
  Type: AWS::DevOpsGuru::ResourceCollection
  Properties:
    ResourceCollectionFilter:
      Tags:
        - AppBoundaryKey: "Devops-guru-cfn-default"
          TagValues:
          - "*"
```

Untuk informasi selengkapnya, lihat AWS::DevOpsGuru::ResourceCollectionAWS: :RDS: :DBInstance di Panduan Pengguna. AWS CloudFormation

Memantau ancaman dengan Amazon GuardDuty RDS Protection

Amazon GuardDuty adalah layanan pemantauan keamanan berkelanjutan yang menganalisis dan memproses berbagai sumber data, termasuk aktivitas login RDS. Ini menggunakan umpan intelijen ancaman dan pembelajaran mesin untuk mengidentifikasi aktivitas yang tidak terduga, berpotensi tidak sah, dan berbahaya di lingkungan Anda AWS .

GuardDuty RDS Protection menganalisis dan memprofilkan peristiwa login untuk potensi ancaman akses ke database Amazon Aurora Anda. Saat Anda mengaktifkan Perlindungan RDS, GuardDuty mengkonsumsi peristiwa login RDS dari database Aurora Anda. Perlindungan RDS memantau peristiwa ini dan memprofilkannya untuk potensi ancaman orang dalam atau aktor eksternal.

Untuk informasi selengkapnya tentang mengaktifkan Perlindungan GuardDuty RDS, lihat Perlindungan GuardDuty RDS di Panduan Pengguna Amazon GuardDuty .

Ketika RDS Protection mendeteksi potensi ancaman, seperti pola yang tidak biasa dalam upaya login yang berhasil atau gagal, GuardDuty menghasilkan temuan baru dengan rincian tentang database yang berpotensi dikompromikan. Anda dapat melihat detail temuan di bagian ringkasan pencarian di GuardDuty konsol Amazon. Detail temuan bervariasi berdasarkan jenis temuan. Detail utama, jenis sumber daya, dan peran sumber daya, menentukan jenis informasi yang tersedia untuk temuan apa pun. Untuk informasi selengkapnya tentang detail umum yang tersedia untuk temuan dan jenis temuan, lihat Detail Menemukan dan jenis temuan Perlindungan GuardDuty RDS masing-masing di Panduan GuardDuty Pengguna Amazon.

Anda dapat mengaktifkan atau menonaktifkan fitur Perlindungan RDS untuk fitur mana pun Akun AWS di Wilayah AWS mana pun fitur ini tersedia. Ketika Perlindungan RDS tidak diaktifkan, GuardDuty tidak mendeteksi database Aurora yang berpotensi dikompromikan atau memberikan rincian kompromi.

GuardDuty Akun yang ada dapat mengaktifkan Perlindungan RDS dengan masa uji coba 30 hari. Untuk GuardDuty akun baru, Perlindungan RDS sudah diaktifkan dan termasuk dalam periode uji coba gratis 30 hari. Untuk informasi selengkapnya, lihat Memperkirakan GuardDuty biaya di Panduan GuardDuty Pengguna Amazon.

Untuk informasi tentang Wilayah AWS s yang GuardDuty belum mendukung Perlindungan RDS, lihat Ketersediaan fitur khusus wilayah di Panduan Pengguna Amazon GuardDuty .

Tabel berikut menyediakan versi database Aurora yang didukung GuardDuty RDS Protection:

Mesin DB Amazon Aurora	Versi mesin yang didukung
Aurora MySQL	• 2.10.2 atau yang lebih baru • 3.02.1 atau yang lebih baru
Aurora PostgreSQL	• 10.17 atau yang lebih baru • 11.12 atau yang lebih baru • 12.7 atau yang lebih baru • 13.3 atau yang lebih baru • 14.3 atau yang lebih baru • 15.2 atau yang lebih baru • 16.1 atau yang lebih baru

Memantau metrik OS dengan Pemantauan yang Disempurnakan

Dengan Pemantauan yang Disempurnakan, Anda dapat memantau sistem operasi instans DB Anda secara real-time. Metrik Pemantauan yang Disempurnakan berguna saat Anda ingin melihat bagaimana proses atau thread yang berbeda menggunakan CPU.

Topik

- Ikhtisar Pemantauan yang Disempurnakan
- Menyiapkan dan mengaktifkan Pemantauan yang Ditingkatkan
- Melihat metrik OS di konsol RDS
- Melihat metrik OS menggunakan Log CloudWatch

Ikhtisar Pemantauan yang Disempurnakan

Amazon RDS menyediakan metrik secara real-time untuk sistem operasi (OS) tempat instans DB Anda dijalankan. Anda dapat melihat semua metrik sistem dan memproses informasi instans DB RDS Anda di konsol. Anda dapat mengelola metrik yang ingin dipantau untuk setiap instans dan menyesuaikan dasbor sesuai kebutuhan Anda. Untuk deskripsi metrik Pemantauan yang Disempurnakan, lihat Metrik OS dalam Pemantauan yang Disempurnakan.

RDS mengirimkan metrik dari Enhanced Monitoring ke akun Amazon CloudWatch Logs Anda. Anda dapat membuat filter metrik CloudWatch dari CloudWatch Log dan menampilkan grafik di dasbor. CloudWatch Anda dapat menggunakan output Enhanced Monitoring JSON dari CloudWatch Log dalam sistem pemantauan pilihan Anda. Untuk informasi selengkapnya, lihat Pemantauan yang Disempurnakan di Tanya Jawab Umum tentang Amazon RDS.

Topik

- Perbedaan antara CloudWatch dan metrik Pemantauan yang Ditingkatkan
- Retensi metrik Pemantauan yang Disempurnakan
- Biaya Pemantauan yang Disempurnakan

Perbedaan antara CloudWatch dan metrik Pemantauan yang Ditingkatkan

Hipervisor menciptakan dan menjalankan mesin virtual (VM). Menggunakan hypervisor, sebuah instance dapat mendukung beberapa VM tamu dengan berbagi memori dan CPU secara virtual.

CloudWatch mengumpulkan metrik tentang pemanfaatan CPU dari hypervisor untuk instance DB. Sebaliknya, Pemantauan yang Disempurnakan mengumpulkan metrik dari agen di instans DB.

Anda mungkin menemukan perbedaan antara pengukuran CloudWatch dan Enhanced Monitoring, karena lapisan hypervisor melakukan sedikit pekerjaan. Perbedaan ini bisa lebih menonjol jika instans DB Anda menggunakan kelas instans yang lebih kecil. Dalam skenario ini, ada lebih banyak mesin virtual (VM) yang mungkin dikelola oleh lapisan hipervisor pada satu instans fisik.

Untuk deskripsi metrik Pemantauan yang Disempurnakan, lihat Metrik OS dalam Pemantauan yang Disempurnakan. Untuk informasi selengkapnya tentang CloudWatch metrik, lihat Panduan CloudWatch Pengguna Amazon.

Retensi metrik Pemantauan yang Disempurnakan

Secara default, metrik Pemantauan yang Ditingkatkan disimpan selama 30 hari di CloudWatch Log. Periode retensi ini berbeda dari CloudWatch metrik biasa.

Untuk mengubah jumlah waktu metrik disimpan di CloudWatch Log, ubah retensi untuk grup `RDSOSMetrics` log di CloudWatch konsol. Untuk informasi selengkapnya, lihat Mengubah penyimpanan data CloudWatch log di log di Panduan Pengguna CloudWatch Log Amazon.

Biaya Pemantauan yang Disempurnakan

Metrik Pemantauan yang Ditingkatkan disimpan di CloudWatch Log, bukan dalam CloudWatch metrik. Biaya Pemantauan yang Disempurnakan ditentukan oleh faktor-faktor berikut:

- Anda dikenakan biaya untuk Enhanced Monitoring hanya jika Anda melebihi tingkat gratis yang disediakan oleh Amazon CloudWatch Logs. Biaya didasarkan pada transfer data CloudWatch Log dan tingkat penyimpanan.
- Jumlah informasi yang ditransfer untuk instans RDS berbanding lurus dengan perincian yang ditentukan untuk fitur Pemantauan yang Disempurnakan. Interval pemantauan yang lebih kecil menghasilkan pelaporan metrik OS yang lebih sering dan meningkatkan biaya pemantauan. Untuk mengelola biaya, atur perincian yang berbeda untuk instans yang berbeda di akun Anda.
- Biaya penggunaan Pemantauan yang Disempurnakan diterapkan untuk setiap instans DB yang fitur Pemantauan yang Disempurnakannya diaktifkan. Pemantauan instans DB dalam jumlah akan lebih mahal dibandingkan dengan pemantauan dalam jumlah sedikit.
- Instans DB yang mendukung beban kerja komputasi yang lebih berat memiliki lebih banyak aktivitas proses OS yang perlu dan biaya untuk Pemantauan yang Disempurnakan lebih tinggi.

Untuk informasi selengkapnya tentang harga, lihat CloudWatch harga Amazon.

Menyiapkan dan mengaktifkan Pemantauan yang Ditingkatkan

Untuk menggunakan Pemantauan yang Ditingkatkan, Anda harus membuat peran IAM, lalu mengaktifkan Pemantauan yang Ditingkatkan.

Topik

- Membuat peran IAM untuk Pemantauan yang Ditingkatkan
- Mengaktifkan dan menonaktifkan Pemantauan yang Ditingkatkan
- Melindungi dari masalah confused deputy

Membuat peran IAM untuk Pemantauan yang Ditingkatkan

Pemantauan yang Ditingkatkan memerlukan izin untuk bertindak atas nama Anda untuk mengirim informasi metrik OS ke CloudWatch Log. Anda memberikan izin Pemantauan yang Ditingkatkan menggunakan peran AWS Identity and Access Management (IAM). Anda dapat membuat peran ini saat mengaktifkan Pemantauan yang Ditingkatkan atau membuatnya terlebih dahulu.

Topik

- Membuat peran IAM saat Anda mengaktifkan Pemantauan yang Ditingkatkan
- Membuat peran IAM saat Anda mengaktifkan Pemantauan yang Ditingkatkan

Membuat peran IAM saat Anda mengaktifkan Pemantauan yang Ditingkatkan

Jika Anda mengaktifkan Pemantauan yang Ditingkatkan di konsol RDS, Amazon RDS dapat membuat peran IAM yang diperlukan untuk Anda. Peran ini bernama `rds-monitoring-role`. RDS menggunakan peran ini untuk instans DB, replika baca, atau klaster DB Multi-AZ tertentu.

Untuk membuat peran IAM saat Anda mengaktifkan Pemantauan yang Ditingkatkan

1. Ikuti langkah-langkah di Mengaktifkan dan menonaktifkan Pemantauan yang Ditingkatkan.
2. Atur Peran Pemantauan ke Default pada langkah tempat Anda memilih peran.

Membuat peran IAM saat Anda mengaktifkan Pemantauan yang Ditingkatkan

Anda dapat membuat peran yang diperlukan sebelum mengaktifkan Pemantauan yang Ditingkatkan. Jika Anda mengaktifkan Pemantauan yang Ditingkatkan, tentukan nama peran baru Anda. Anda harus membuat peran yang diperlukan ini jika Anda mengaktifkan Pemantauan yang Ditingkatkan menggunakan AWS CLI atau API RDS.

Pengguna yang mengaktifkan Pemantauan yang Ditingkatkan harus diberi izin `PassRole`. Untuk informasi selengkapnya, lihat Contoh 2 dalam Memberikan izin pengguna untuk meneruskan peran ke AWS layanan di Panduan Pengguna IAM.

Untuk membuat peran IAM untuk pemantauan yang ditingkatkan Amazon RDS

1. Buka Konsol IAM di https://console.aws.amazon.com.
2. Di panel navigasi, pilih Peran.
3. Pilih Buat peran.
4. Pilih tab Layanan AWS , lalu pilih RDS dari daftar layanan.
5. Pilih RDS - Pemantauan yang Ditingkatkan, lalu pilih Berikutnya.
6. Pastikan kebijakan Izin menampilkan AmazonRDS EnhancedMonitoringRole, lalu pilih Berikutnya.
7. Untuk Nama peran, masukkan nama peran Anda. Misalnya, masukkan **`emaccess`**.

   Entitas tepercaya untuk peran Anda adalah AWS layanan monitoring.rds.amazonaws.com.
8. Pilih Buat peran.

Mengaktifkan dan menonaktifkan Pemantauan yang Ditingkatkan

Anda dapat mengaktifkan dan menonaktifkan Enhanced Monitoring menggunakan AWS Management Console, AWS CLI, atau RDS API. Anda memilih instans DB RDS yang Pemantauan yang Ditingkatkannya ingin diaktifkan. Anda dapat mengatur granularitas yang berbeda untuk pengumpulan metrik pada setiap instans DB.

Konsol

Anda dapat mengaktifkan Pemantauan yang Ditingkatkan saat membuat klaster atau replika baca, atau saat Anda memodifikasi instans. Jika Anda memodifikasi instans DB untuk mengaktifkan Pemantauan yang Ditingkatkan, Anda tidak perlu mem-boot ulang instans DB Anda agar perubahan diterapkan.

Anda dapat mengaktifkan Pemantauan yang Ditingkatkan di konsol RDS saat Anda melakukan salah satu tindakan berikut di halaman Basis data:

- Buat klaster – Pilih Buat basis data.
- Buat replika baca – Pilih Tindakan, lalu Buat replika baca.
- Modifikasi instans DB – Pilih Ubah.

Untuk mengaktifkan atau menonaktifkan Pemantauan yang Ditingkatkan di konsol RDS

1. Gulir ke bagian Konfigurasi tambahan.
2. Di Pemantauan, pilih Aktifkan Pemantauan yang Ditingkatkan untuk instans DB atau replika baca. Untuk menonaktifkan Pemantauan yang Ditingkatkan, pilih Nonaktifkan Pemantauan yang Ditingkatkan.
3. Setel properti Peran Pemantauan ke peran IAM yang Anda buat untuk mengizinkan Amazon RDS berkomunikasi dengan CloudWatch Log Amazon untuk Anda, atau pilih Default agar RDS membuat peran untuk Anda namai. `rds-monitoring-role`
4. Atur properti Granularitas ke interval tersebut, dalam detik, antara titik-titik saat metrik dikumpulkan untuk instans DB atau replika baca. Properti Granularitas dapat diatur ke salah satu nilai berikut: `1`, `5`, `10`, `15`, `30`, atau `60`.

   Waktu yang paling cepat di mana konsol RDS disegarkan adalah setiap 5 detik. Jika Anda mengatur granularitas ke 1 detik di konsol RDS, Anda masih dapat melihat metrik yang diperbarui hanya setiap 5 detik. Anda dapat mengambil pembaruan metrik 1 detik dengan menggunakan CloudWatch Log.

AWS CLI

Untuk mengaktifkan Enhanced Monitoring menggunakan AWS CLI, dalam perintah berikut, atur `--monitoring-interval` opsi ke nilai selain `0` dan atur `--monitoring-role-arn` opsi ke peran yang Anda buatMembuat peran IAM untuk Pemantauan yang Ditingkatkan.

- create-db-instance
- create-db-instance-read-replika
- modify-db-instance

Opsi `--monitoring-interval` menentukan interval, dalam detik, antara titik-titik saat metrik Pemantauan yang Ditingkatkan dikumpulkan. Nilai yang valid untuk opsi ini adalah `0`, `1`, `5`, `10`, `15`, `30`, dan `60`.

Untuk mematikan Enhanced Monitoring menggunakan AWS CLI, atur `--monitoring-interval` opsi ke `0` dalam perintah ini.

Example

Contoh berikut mengaktifkan Pemantauan yang Ditingkatkan untuk instans DB:

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Untuk Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier mydbinstance ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Example

Contoh berikut mengaktifkan Pemantauan yang Ditingkatkan untuk klaster DB Multi-AZ:

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --monitoring-interval 30 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --monitoring-interval 30 ^
    --monitoring-role-arn arn:aws:iam::123456789012:role/emaccess
```

API RDS

Untuk mengaktifkan Pemantauan yang Ditingkatkan menggunakan RDS API, atur parameter `MonitoringInterval` ke nilai selain `0` dan atur parameter `MonitoringRoleArn` ke peran yang Anda buat di Membuat peran IAM untuk Pemantauan yang Ditingkatkan. Tetapkan parameter ini dalam tindakan berikut:

- CreateDBInstance
- dibuatB InstanceReadReplica
- ModifyDBInstance

Parameter `MonitoringInterval` menentukan interval, dalam detik, antara titik-titik saat metrik Pemantauan yang Ditingkatkan dikumpulkan. Nilai yang valid adalah `0`, `1`, `5`, `10`, `15`, `30`, dan `60`.

Untuk menonaktifkan Pemantauan yang Ditingkatkan menggunakan API RDS, atur `MonitoringInterval` ke `0`.

Melindungi dari masalah confused deputy

Masalah deputi yang bingung adalah masalah keamanan di mana entitas yang tidak memiliki izin untuk melakukan tindakan dapat memaksa entitas yang lebih istimewa untuk melakukan tindakan. Pada tahun AWS, peniruan lintas layanan dapat mengakibatkan masalah wakil yang membingungkan. Peniruan identitas lintas layanan dapat terjadi ketika satu layanan (layanan yang dipanggil) memanggil layanan lain (layanan yang dipanggil). Layanan pemanggilan dapat dimanipulasi menggunakan izinnya untuk bertindak pada sumber daya pelanggan lain dengan cara yang seharusnya tidak dilakukannya kecuali bila memiliki izin untuk mengakses. Untuk mencegah hal ini, AWS menyediakan alat yang membantu Anda melindungi data untuk semua layanan dengan pengguna utama layanan yang telah diberi akses ke sumber daya di akun Anda. Untuk informasi selengkapnya, lihat Masalah confused deputy.

Untuk membatasi izin ke sumber daya yang dapat diberikan Amazon RDS kepada layanan lain, sebaiknya gunakan kunci konteks kondisi global `aws:SourceArn` dan `aws:SourceAccount` dalam kebijakan kepercayaan untuk peran Pemantauan yang Ditingkatkan. Jika Anda menggunakan kedua kunci konteks kondisi global, keduanya harus menggunakan ID akun yang sama.

Cara paling efektif untuk melindungi dari masalah confused deputy adalah dengan menggunakan kunci konteks kondisi global `aws:SourceArn` dengan ARN lengkap sumber daya. Untuk Amazon RDS, atur `aws:SourceArn` ke `arn:aws:rds:`*`Region`*`:`*`my-account-id`*`:db:`*`dbname`*.

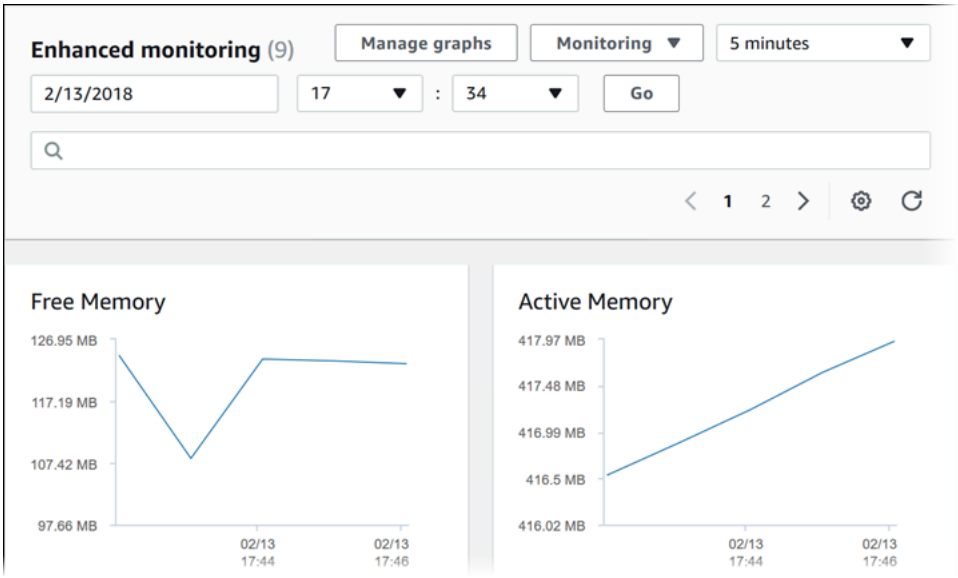
Contoh berikut menggunakan kunci konteks kondisi global `aws:SourceArn` dan `aws:SourceAccount` dalam kebijakan kepercayaan untuk mencegah masalah deputi yang bingung.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "monitoring.rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringLike": {
          "aws:SourceArn": "arn:aws:rds:Region:my-account-id:db:dbname"
        },
        "StringEquals": {
          "aws:SourceAccount": "my-account-id"
        }
      }
    }
  ]
}
```

Melihat metrik OS di konsol RDS

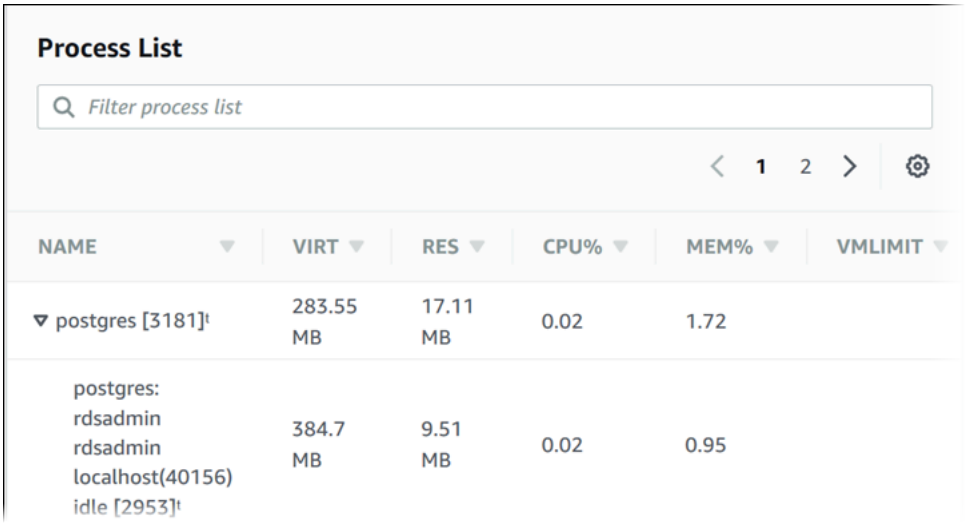
Anda dapat melihat metrik OS yang dilaporkan oleh Pemantauan yang Disempurnakan di konsol RDS dengan memilih Pemantauan yang disempurnakan untuk Pemantauan.

Contoh berikut menunjukkan halaman Pemantauan yang Disempurnakan. Untuk deskripsi metrik Pemantauan yang Disempurnakan, lihat Metrik OS dalam Pemantauan yang Disempurnakan.



Jika Anda ingin melihat detail proses yang berjalan pada instans DB Anda, pilih Daftar proses OS untuk Pemantauan.

Tampilan Daftar Proses ditunjukkan sebagai berikut.



Metrik Pemantauan yang Disempurnakan yang ditunjukkan dalam tampilan Daftar proses disusun sebagai berikut:

- Proses turunan RDS – Menampilkan ringkasan proses RDS yang mendukung instans DB, misalnya `aurora` untuk klaster DB Amazon Aurora. Rangkaian proses muncul bersarang di bawah proses induk. Rangkaian proses hanya menampilkan penggunaan CPU karena metrik lain sama untuk semua rangkaian proses. Konsol menampilkan maksimal 100 proses dan rangkaian. Hasilnya adalah gabungan dari proses dan rangkaian yang menggunakan CPU dan memori. Jika ada lebih

dari 50 proses dan lebih dari 50 rangkaian, konsol akan menampilkan 50 pengonsumsi teratas di setiap kategori. Tampilan ini membantu Anda mengidentifikasi proses mana yang memiliki dampak paling besar pada performa.

- Proses RDS – Menampilkan ringkasan sumber daya yang digunakan oleh agen manajemen RDS, proses pemantauan diagnostik, dan proses AWS lain yang diperlukan untuk mendukung instans DB RDS.
- Proses OS – Menampilkan ringkasan proses sistem dan kernel, yang umumnya berdampak minimal pada performa.

Item yang tercantum untuk setiap proses meliputi:

- VIRT – Menampilkan ukuran virtual proses.
- RES – Menampilkan memori fisik aktual yang sedang digunakan oleh proses.
- CPU% – Menampilkan persentase total bandwidth CPU yang sedang digunakan oleh proses.
- CPU% – Menampilkan persentase total memori yang sedang digunakan oleh proses.

Data pemantauan yang ditampilkan di konsol RDS diambil dari Log Amazon CloudWatch. Anda juga dapat mengambil metrik untuk instans DB sebagai log stream dari Log CloudWatch. Untuk informasi selengkapnya, lihat Melihat metrik OS menggunakan Log CloudWatch.

Metrik Pemantauan yang Disempurnakan tidak ditampilkan selama:

- Failover instans DB.
- Mengubah kelas instans dari instans DB (komputasi skala).

Metrik Pemantauan yang Disempurnakan ditampilkan selama proses reboot instans DB karena hanya mesin basis data yang di-reboot. Metrik untuk sistem operasi tetap dilaporkan.

Melihat metrik OS menggunakan Log CloudWatch

Setelah mengaktifkan Pemantauan yang Disempurnakan untuk klaster, Anda dapat melihat metrik menggunakan Log CloudWatch, dengan setiap log stream yang mewakili satu instans DB atau klaster DB yang dipantau. Pengidentifikasi log stream adalah pengidentifikasi sumber daya (`DbiResourceId`) untuk instans DB atau klaster DB.

Untuk melihat data log Pemantauan yang Disempurnakan

1. Buka konsol CloudWatch di https://console.aws.amazon.com/cloudwatch/.
2. Jika perlu, pilih Wilayah AWS tempat klaster Anda berada. Untuk informasi selengkapnya, lihat Wilayah dan titik akhir dalam Referensi Umum Amazon Web Services.
3. Pilih Log di panel navigasi.
4. Pilih RDSOSMetrics dari daftar grup log.
5. Pilih log stream yang ingin dilihat dari daftar log stream.

Referensi metrik untuk Amazon Aurora

Dalam referensi ini, Anda dapat menemukan deskripsi metrik Amazon Aurora untuk Amazon CloudWatch, Wawasan Performa, dan Peningkatan Pemantauan.

Topik

- CloudWatch Metrik Amazon untuk Amazon Aurora
- Dimensi-dimensi Amazon CloudWatch untuk Aurora
- Ketersediaan metrik Aurora di konsol Amazon RDS
- CloudWatch Metrik Amazon untuk Performance Insights
- Metrik penghitung Wawasan Performa
- Statistik SQL untuk Wawasan Performa
- Metrik OS dalam Pemantauan yang Disempurnakan

CloudWatch Metrik Amazon untuk Amazon Aurora

Ruang nama `AWS/RDS` mencakup metrik berikut yang berlaku untuk entitas basis data yang berjalan di Amazon Aurora. Beberapa metrik berlaku untuk Aurora MySQL, Aurora PostgreSQL, atau keduanya. Selain itu, beberapa metrik bersifat spesifik untuk klaster DB, instans DB replika, atau semua instans DB.

Untuk metrik basis data global Aurora, lihat Metrik Amazon CloudWatch untuk penerusan tulis di Aurora MySQL dan CloudWatch Metrik Amazon untuk penerusan tulis di Aurora PostgreSQL. Untuk metrik kueri paralel Aurora, lihat Memantau kueri paralel.

Topik

- Metrik tingkat klaster untuk Amazon Aurora
- Metrik tingkat instans untuk Amazon Aurora
- Metrik CloudWatch penggunaan Amazon untuk

Metrik tingkat klaster untuk Amazon Aurora

Tabel berikut menjelaskan metrik yang spesifik untuk klaster Aurora.

Metrik tingkat klaster Amazon Aurora

Metrik	Deskripsi	Berlaku untuk	Unit
`AuroraGlobalDBDataTransferBytes`	Dalam Database Global Aurora, jumlah data redo log yang ditransfer dari AWS Wilayah master ke Wilayah sekunder. AWS **Note** Metrik ini hanya tersedia di Wilayah AWS sekunder.	Aurora MySQL dan Aurora PostgreSQL	Byte
`AuroraGlobalDBProgressLag`	Dalam Basis Data Global Aurora, ukuran seberapa jauh klaster sekunder tertinggal dari klaster primer untuk transaksi pengguna dan transaksi sistem. **Note** Metrik ini hanya tersedia di Wilayah AWS sekunder.	Aurora MySQL dan Aurora PostgreSQL	Milidetik
`AuroraGlobalDBReplicatedWriteIO`	Dalam Database Global Aurora, jumlah operasi I/O tulis direplikasi dari AWS Region primer ke volume cluster di Region sekunder. AWS Perhitungan penagihan untuk AWS	Aurora MySQL dan Aurora PostgreSQL	Hitungan

Metrik	Deskripsi	Berlaku untuk	Unit
	Wilayah sekunder dalam database global digunakan `VolumeWriteIOPs` untuk memperhitungkan penulisan yang dilakukan dalam cluster. Perhitungan penagihan untuk AWS Wilayah utama dalam database global digunakan `VolumeWriteIOPs` untuk memperhitungkan aktivitas penulisan dalam klaster tersebut, dan `AuroraGlobalDBReplicatedWriteIO` untuk memperhitungkan replikasi Lintas wilayah dalam database global. **Note** Metrik ini hanya tersedia di Wilayah AWS sekunder.		

Metrik	Deskripsi	Berlaku untuk	Unit
`AuroraGlobalDBReplicationLag`	Untuk Basis Data Global Aurora, jumlah lag saat mereplikasi pembaruan dari Wilayah AWS primer. **Note** Metrik ini hanya tersedia di Wilayah AWS sekunder.	Aurora MySQL dan Aurora PostgreSQL	Milidetik
`AuroraGlobalDBRPOLag`	Dalam Basis Data Aurora Global, waktu lag sasaran titik pemulihan (RPO). Metrik ini mengukur seberapa jauh klaster sekunder tertinggal dari klaster primer untuk transaksi pengguna. **Note** Metrik ini hanya tersedia di Wilayah AWS sekunder.	Aurora MySQL dan Aurora PostgreSQL	Milidetik

Metrik	Deskripsi	Berlaku untuk	Unit
`AuroraVolumeBytesLeftTotal`	Sisa ruang yang tersedia untuk volume klaster. Saat volume klaster bertambah , nilai ini menurun. Jika mencapai nol, cluster melaporkan out-of-space kesalahan. Jika Anda ingin mendeteksi apakah klaster Aurora MySQL Anda mendekati batas ukuran 128 tebibytes (TiB), nilai ini lebih sederhana dan lebih dapat diandalkan untuk dipantau dibandingkan `VolumeBytesUsed` . `AuroraVolumeBytesLeftTotal` memperhitungkan penyimpanan yang digunakan untuk housekeeping internal dan alokasi lain yang tidak memengaruhi tagihan penyimpanan Anda.	Aurora MySQL	Byte
`BacktrackChangeRecordsCreationRate`	Jumlah catatan perubahan pelacakan mundur yang dibuat selama 5 menit untuk klaster DB Anda.	Aurora MySQL	Hitungan per 5 menit
`BacktrackChangeRecordsStored`	Jumlah catatan perubahan pelacakan mundur yang digunakan oleh klaster DB Anda.	Aurora MySQL	Hitungan

Metrik	Deskripsi	Berlaku untuk	Unit
`BackupRetentionPeriodStorageUsed`	Jumlah total penyimpanan cadangan yang digunakan untuk mendukung fitur point-in-time pemulihan dalam jendela retensi cadangan cluster Aurora DB. Jumlah ini termasuk dalam total yang dilaporkan oleh metrik `TotalBackupStorageBilled` . Metrik ini dihitung secara terpisah untuk setiap klaster Aurora. Untuk petunjuk, lihat Memahami penggunaan penyimpanan cadangan Amazon Aurora.	Aurora MySQL dan Aurora PostgreSQL	Byte
`ServerlessDatabaseCapacity`	Kapasitas klaster DB Aurora Serverless saat ini.	Aurora MySQL dan Aurora PostgreSQL	Hitungan

Metrik	Deskripsi	Berlaku untuk	Unit
SnapshotStorageUsed	Jumlah total penyimpanan cadangan yang digunakan oleh semua snapshot Aurora untuk klaster DB Aurora di luar periode penyimpanan cadangan. Jumlah ini termasuk dalam total yang dilaporkan oleh metrik TotalBackupStorageBilled . Metrik ini dihitung secara terpisah untuk setiap klaster Aurora. Untuk petunjuk, lihat Memahami penggunaan penyimpanan cadangan Amazon Aurora.	Aurora MySQL dan Aurora PostgreSQL	Byte
TotalBackupStorageBilled	Jumlah total penyimpanan cadangan dalam byte yang akan ditagihkan untuk klaster DB Aurora tertentu. Metrik ini mencakup penyimpanan cadangan yang diukur oleh metrik BackupRetentionPeriodStorageUsed dan SnapshotStorageUsed . Metrik ini dihitung secara terpisah untuk setiap klaster Aurora. Untuk petunjuk, lihat Memahami penggunaan penyimpanan cadangan Amazon Aurora.	Aurora MySQL dan Aurora PostgreSQL	Byte

Metrik	Deskripsi	Berlaku untuk	Unit
`VolumeBytesUsed`	Jumlah penyimpanan yang digunakan oleh instans Aurora DB Anda. Nilai ini memengaruhi biaya klaster DB Aurora (untuk informasi harga, lihat halaman harga Amazon RDS). Nilai ini tidak mencermin kan beberapa alokasi penyimpanan internal yang tidak memengaru hi tagihan penyimpan an. Untuk Aurora MySQL Anda dapat mengantis ipasi out-of-space masalah lebih akurat dengan menguji `AuroraVol umeBytesLeftTotal` apakah mendekati nol daripada membandingkan `VolumeBytesUsed` dengan batas penyimpanan 128 TiB.	Aurora MySQL dan Aurora PostgreSQL	Byte

Metrik	Deskripsi	Berlaku untuk	Unit
`VolumeReadIOPs`	Jumlah operasi I/O baca yang ditagih dari volume klaster dalam interval 5 menit. Operasi baca yang ditagih dihitung pada tingkat volume klaster, dikumpulkan dari semua instans di klaster DB Aurora, lalu dilaporkan pada interval 5 menit. Nilai ini dihitung dengan mengambil nilai dari metrik Operasi baca selama periode 5 menit. Anda dapat menentukan jumlah operasi baca yang ditagih per detik dengan mengambil nilai metrik Operasi baca yang ditagih dan membaginya dengan 300 detik. Misalnya, jika Operasi baca yang ditagih menghasilkan 13.686, maka operasi baca yang ditagih per detik adalah 45 (13.686 / 300 = 45,62). Anda mengakumulasi operasi baca yang ditagih untuk kueri yang meminta halaman basis data yang tidak ada dalam cache buffer dan harus dimuat	Aurora MySQL dan Aurora PostgreSQL	Hitungan per 5 menit

Metrik	Deskripsi	Berlaku untuk	Unit
	dari penyimpanan. Anda mungkin melihat lonjakan dalam operasi baca yang ditagih karena hasil kueri dibaca dari penyimpanan, lalu dimuat ke dalam cache buffer. **Tip** Jika klaster Aurora MySQL Anda menggunakan kueri paralel, Anda mungkin melihat peningkatan nilai `VolumeReadIOPS` . Kueri paralel tidak menggunakan kumpulan buffer. Jadi, meskipun kuerinya cepat, pemrosesan yang dioptimalkan ini dapat menghasilkan peningkatan operasi baca dan biaya terkait.		

Metrik	Deskripsi	Berlaku untuk	Unit
`VolumeWriteIOPs`	Jumlah operasi I/O disk tulis untuk volume klaster, yang dilaporkan dalam interval 5 menit. Untuk penjelasan mendetail tentang cara menghitung operasi tulis yang ditagih, lihat `VolumeReadIOPs` .	Aurora MySQL dan Aurora PostgreSQL	Hitungan per 5 menit

Metrik tingkat instans untuk Amazon Aurora

CloudWatch Metrik khusus instance berikut berlaku untuk semua instance Aurora MySQL dan Aurora PostgreSQL kecuali dinyatakan lain.

Metrik tingkat instans Amazon Aurora

Metrik	Deskripsi	Berlaku untuk	Unit
`AbortedClients`	Jumlah koneksi klien yang belum ditutup dengan benar.	Aurora MySQL	Hitungan
`ActiveTransactions`	Rata-rata jumlah transaksi saat ini yang dijalankan pada instans basis data Aurora per detik. Secara default, Aurora tidak mengaktifkan metrik ini. Untuk mulai mengukur nilai ini, tetapkan `innodb_monitor_enable='all'` dalam grup parameter DB untuk instans DB spesifik.	Aurora MySQL	Hitungan per detik

Metrik	Deskripsi	Berlaku untuk	Unit
`ACUUtilization`	Nilai metrik `ServerlessDatabaseCapacity` dibagi dengan nilai ACU maksimum klaster DB. Metrik ini berlaku untuk Aurora Nirserver v1 dan Aurora Nirserver v2.	Aurora MySQL dan Aurora PostgreSQL	Persentase

Metrik	Deskripsi	Berlaku untuk	Unit
`AuroraBinlogReplicaLag`	Jumlah waktu klaster DB replika log biner yang berjalan pada Aurora Edisi Kompatibel MySQL mengalami lag dari sumber replikasi log biner. Lag berarti bahwa sumber menghasilkan catatan lebih cepat daripada replika dapat menerapkannya. Metrik ini melaporkan nilai yang berbeda tergantung pada versi mesin: Aurora MySQL versi 2 Bidang `Seconds_Behind_Master` dari `SHOW SLAVE STATUS` MySQL Aurora MySQL versi 3 `SHOW REPLICA STATUS` Anda dapat menggunakan metrik ini untuk memantau kesalahan dan lag replika dalam klaster yang bertindak sebagai replika log biner. Nilai metrik menunjukkan hal berikut:	Primer untuk Aurora MySQL	Detik

Metrik	Deskripsi	Berlaku untuk	Unit

	Nilai tinggi
	
	Replika mengalami lag dari sumber replikasi.
	
	`0` atau nilai yang mendekati `0`
	
	Proses replika aktif dan terkini.
	
	`-1`
	
	Aurora tidak dapat menentukan lag, yang dapat terjadi selama pengaturan replika atau ketika replika mengalami kesalahan.
	
	Karena replikasi log biner hanya terjadi pada instans penulis klaster, sebaiknya gunakan versi metrik ini yang terkait dengan peran WRITER.
	
	Untuk informasi selengkapnya tentang melakukan replikasi, lihat Mereplikasi klaster DB Amazon Aurora MySQL di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang pemecahan masalah, lihat Masalah replikasi Amazon Aurora MySQL.		

Metrik	Deskripsi	Berlaku untuk	Unit
`AuroraEstimatedSharedMemoryBytes`	Perkiraan jumlah memori buffer bersama atau pool buffer yang digunakan secara aktif selama interval polling yang terakhir dikonfigurasi.		Byte
`AuroraOptimizedReadsCacheHitRatio`	Persentase permintaan yang dilayani oleh cache Optimized Reads. Nilai dihitung menggunakan rumus berikut: `orcache_blks_hit/ (orcache_blks_hit + storage_blks_read)` Kapan `AuroraOptimizedReadsCacheHitRatio` 100%, itu berarti tidak ada halaman yang dibaca dari cache Bacaan yang Dioptimalkan dan nilainya akan menjadi`0`.	Primer untuk Aurora PostgreSQL	Persentase
`AuroraReplicaLag`	Untuk replika Aurora, jumlah lag saat mereplikasi pembaruan dari instans primer.	Replika untuk Aurora MySQL dan Aurora PostgreSQL	Milidetik

Metrik	Deskripsi	Berlaku untuk	Unit
AuroraReplicaLagMaximum	Jumlah maksimum lag antara instans primer dan instans DB Aurora dalam klaster DB.	Primer untuk Aurora MySQL dan Aurora PostgreSQL	Milidetik
AuroraReplicaLagMinimum	Jumlah minimum lag antara instans primer dan instans DB Aurora dalam klaster DB.	Primer untuk Aurora MySQL dan Aurora PostgreSQL	Milidetik
AuroraSlowConnectionHandleCount	Jumlah koneksi yang telah menunggu dua detik atau lebih lama untuk memulai handshake. Metrik ini hanya berlaku untuk Aurora MySQL versi 3.	Aurora MySQL	Hitungan
AuroraSlowHandshakeCount	Jumlah koneksi yang membutuhkan waktu 50 milidetik atau lebih lama untuk menyelesaikan handshake. Metrik ini hanya berlaku untuk Aurora MySQL versi 3.	Aurora MySQL	Hitungan
BacktrackWindowActual	Perbedaan antara periode pelacakan mundur target dan periode pelacakan mundur sebenarnya.	Primer untuk Aurora MySQL	Menit

Metrik	Deskripsi	Berlaku untuk	Unit
`BacktrackWindowAlert`	Jumlah berapa kali periode pelacakan mundur sebenarnya lebih kecil daripada periode pelacakan mundur target untuk periode waktu tertentu.	Primer untuk Aurora MySQL	Hitungan
`BlockedTransactions`	Jumlah rata-rata transaksi dalam basis data yang diblok detik.	Aurora MySQL	Hitungan per detik
`BufferCacheHitRatio`	Persentase permintaan yang dilayani oleh cache buffer.	Aurora MySQL dan Aurora PostgreSQL	Persentase
`CommitLatency`	Durasi rata-rata yang diperlukan oleh mesin dan penyimpanan untuk menyelesaikan operasi commit.	Aurora MySQL dan Aurora PostgreSQL	Milidetik
`CommitThroughput`	Rata-rata jumlah operasi commit per detik.	Aurora MySQL dan Aurora PostgreSQL	Hitungan per detik
`ConnectionAttempts`	Jumlah percobaan untuk terhubung ke sebuah instans, baik berhasil maupun tidak.	Aurora MySQL	Hitungan

Metrik	Deskripsi	Berlaku untuk	Unit
`CPUCreditBalance`	Jumlah kredit CPU yang diakumulasikan oleh satu instans, yang dilaporkan pada interval 5 menit. Anda dapat menggunakan metrik ini untuk menentukan berapa lama instans DB dapat melakukan burst melampaui tingkat performa dasarnya pada tingkat tertentu. Metrik ini hanya berlaku untuk kelas instance ini: • Aurora MySQL: `db.t2.small` , `db.t2.medium` , `db.t3`, dan `db.t4g` • Aurora PostgreSQL: `db.t3` dan `db.t4g` **Note** Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans	Aurora MySQL dan Aurora PostgreSQL	Hitungan

Metrik	Deskripsi	Berlaku untuk	Unit
	T, lihat Jenis kelas instans DB. Cara kerja kredit peluncuran di Amazon RDS dan di Amazon EC2 sama. Untuk informasi selengkapnya, lihat Kredit peluncuran dalam Panduan Pengguna Amazon Elastic Compute Cloud untuk Instans Linux.		

Metrik	Deskripsi	Berlaku untuk	Unit
`CPUCreditUsage`	Jumlah CPU yang digunakan selama periode tertentu, yang dilaporkan pada interval 5 menit. Metrik ini mengukur jumlah waktu selama CPU fisik digunakan untuk memproses petunjuk dari CPU virtual yang dialokasikan ke instans DB. Metrik ini hanya berlaku untuk kelas instance ini: • Aurora MySQL: `db.t2.small` , `db.t2.medium` , `db.t3`, dan `db.t4g` • Aurora PostgreSQL: `db.t3` dan `db.t4g` **Note** Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans	Aurora MySQL dan Aurora PostgreSQL	Hitungan

Metrik	Deskripsi	Berlaku untuk	Unit
	T, lihat Jenis kelas instans DB.		
`CPUSurplusCreditBalance`	Jumlah kredit surplus yang telah digunakan oleh sebuah instans tak terbatas ketika nilai `CPUCreditBalance` -nya nol. Nilai `CPUSurplusCreditBalance` ditutupi dengan kredit CPU yang diperoleh. Jika jumlah kredit surplus melebihi jumlah kredit maksimum yang dapat diperoleh instans dalam jangka waktu 24 jam, kredit surplus yang digunakan di atas jumlah maksimum akan dikenai biaya tambahan. Metrik kredit CPU hanya tersedia dalam frekuensi 5 menit.	Aurora MySQL dan Aurora PostgreSQL	Kredit (vCPU-menit)

Metrik	Deskripsi	Berlaku untuk	Unit
`CPUSurplusCreditsCharged`	Jumlah kredit surplus yang digunakan yang tidak ditutupi oleh kredit CPU yang diperoleh, sehingga menimbulkan biaya tambahan. Kredit surplus yang digunakan akan dikenai biaya jika salah satu dari hal berikut terjadi: • Kredit surplus yang digunakan melampaui jumlah kredit maksimum yang bisa didapatkan oleh instans dalam periode 24 jam. Kredit surplus yang digunakan di atas jumlah maksimum akan dikenai biaya pada akhir jam. • Instans dihentikan atau diakhiri. • instans dialihkan dari `unlimited` ke `standard`. Metrik kredit CPU hanya tersedia dalam frekuensi 5 menit.	Aurora MySQL dan Aurora PostgreSQL	Kredit (vCPU-menit)

Metrik	Deskripsi	Berlaku untuk	Unit
`CPUUtilization`	Persentase CPU yang digunakan oleh instans DB Aurora.	Aurora MySQL dan Aurora PostgreSQL	Persentase
`DatabaseConnections`	Jumlah koneksi jaringan klien ke instans basis data. Jumlah sesi basis data bisa lebih tinggi dari nilai metrik karena nilai metrik tidak termasuk yang berikut: • Sesi yang tidak lagi memiliki koneksi jaringan, tetapi belum dibersihkan oleh basis data • Sesi yang dibuat oleh mesin basis data untuk tujuannya sendiri • Sesi yang dibuat oleh kemampuan eksekusi paralel mesin basis data • Sesi yang dibuat oleh penjadwal pekerjaan mesin basis data • Koneksi Amazon Aurora	Aurora MySQL dan Aurora PostgreSQL	Hitungan
`DDLLatency`	Durasi rata-rata permintaan seperti contoh, membuat, mengubah, dan membatalkan permintaan.	Aurora MySQL	Milidetik

Metrik	Deskripsi	Berlaku untuk	Unit
DDLThroughput	Jumlah rata-rata permintaan DDL per detik.	Aurora MySQL	Hitungan per detik
Deadlocks	Jumlah rata-rata deadlock di basis data per detik.	Aurora MySQL dan Aurora PostgreSQL	Hitungan per detik
DeleteLatency	Durasi rata-rata operasi penghapusan.	Aurora MySQL	Milidetik
DeleteThroughput	Jumlah rata-rata kueri penghapusan per detik.	Aurora MySQL	Hitungan per detik
DiskQueueDepth	Jumlah permintaan baca/tulis tertunda yang menunggu untuk mengakses disk.	Aurora MySQL dan Aurora PostgreSQL	Hitungan
DMLLatency	Durasi rata-rata penyisipan, pembaruan, dan penghapusan.	Aurora MySQL	Milidetik
DMLThroughput	Jumlah rata-rata penyisipan, pembaruan, dan penghapusan per detik.	Aurora MySQL	Hitungan per detik
EngineUptime	Jumlah waktu yang digunakan untuk menjalankan instans.	Aurora MySQL dan Aurora PostgreSQL	Detik
FreeableMemory	Jumlah memori akses acak yang tersedia.	Aurora MySQL dan Aurora PostgreSQL	Byte

Metrik	Deskripsi	Berlaku untuk	Unit
`FreeEphemeralStorage`	Jumlah penyimpanan NVMe Efemeral yang tersedia.	Aurora PostgreSQL	Byte
`FreeLocalStorage`	Jumlah penyimpanan lokal yang tersedia. Berbeda dengan mesin DB lainnya, untuk instans DB Aurora, metrik ini melaporkan jumlah penyimpanan yang tersedia untuk setiap instans DB. Nilai ini bergantung pada kelas instans DB (untuk informasi harga, lihat halaman Harga Amazon RDS). Anda dapat meningkatkan jumlah ruang penyimpanan gratis untuk sebuah instans dengan memilih kelas instans DB yang lebih besar untuk instans Anda. (Hal ini tidak berlaku untuk Aurora Serverless v2.)	Aurora MySQL dan Aurora PostgreSQL	Byte
`InsertLatency`	Durasi rata-rata operasi penyisipan.	Aurora MySQL	Milidetik
`InsertThroughput`	Jumlah rata-rata operasi penyisipan per detik.	Aurora MySQL	Hitungan per detik
`LoginFailures`	Jumlah rata-rata percobaan masuk gagal per detik.	Aurora MySQL	Hitungan per detik

Metrik	Deskripsi	Berlaku untuk	Unit
`MaximumUsedTransactionIDs`	Umur ID transaksi tertua yang belum divakum, dalam transaksi. Jika nilai ini mencapai 2.146.483.648 (2^31 - 1.000.000), basis data akan dipaksa ke mode hanya baca untuk menghindari wraparound ID transaksi. Untuk informasi selengkapnya, lihat Preventing transaction ID wraparound failures dalam dokumentasi PostgreSQL.	Aurora PostgreSQL	Hitungan
`NetworkReceiveThroughput`	Jumlah hasil jaringan yang diterima dari klien oleh setiap instans di klaster DB Aurora . Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster Aurora.	Aurora MySQL dan Aurora PostgreSQL	Byte per detik (konsol menunjukkan Megabyte per detik)
`NetworkThroughput`	Jumlah hasil jaringan yang diterima dari dan dikirim ke klien oleh setiap instans di klaster DB Aurora . Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster Aurora.	Aurora MySQL dan Aurora PostgreSQL	Byte per detik

Metrik	Deskripsi	Berlaku untuk	Unit
NetworkTransmitThroughput	Jumlah throughput jaringan yang dikirim ke klien oleh setiap instans dalam klaster DB Aurora. Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster.	Aurora MySQL dan Aurora PostgreSQL	Byte per detik (konsol menunjukkan Megabyte per detik)
NumBinaryLogFiles	Jumlah file binlog yang dihasilkan.	Aurora MySQL	Hitungan
Queries	Rata-rata jumlah kueri yang dijalankan per detik.	Aurora MySQL	Hitungan per detik
RDSToAuroraPostgreSQLReplicaLag	Lag saat mereplikasi pembaruan dari instans RDS PostgreSQL primer ke simpul lain dalam klaster.	Replika untuk Aurora PostgreSQL	Detik
ReadIOPS	Jumlah rata-rata operasi I/O disk per detik, tetapi laporan membaca dan menulis secara terpisah, dalam interval 1 menit.	Aurora MySQL dan Aurora PostgreSQL	Hitungan per detik
ReadIOPSEphemeralStorage	Jumlah rata-rata operasi I/O baca disk ke penyimpanan NVMe Efemeral.	Aurora PostgreSQL	Hitungan per detik
ReadLatency	Jumlah waktu rata-rata yang digunakan per operasi I/O disk.	Aurora MySQL dan Aurora PostgreSQL	Detik

Metrik	Deskripsi	Berlaku untuk	Unit
`ReadLatencyEphemeralStorage`	Jumlah waktu rata-rata yang digunakan per operasi I/O baca disk untuk penyimpanan NVMe Efemeral.	Aurora PostgreSQL	Milidetik
`ReadThroughput`	Jumlah byte rata-rata yang dibaca dari disk per detik.	Aurora MySQL dan Aurora PostgreSQL	Byte per detik
`ReadThroughputEphemeralStorage`	Jumlah rata-rata byte yang dibaca dari disk per detik untuk penyimpanan NVMe Efemeral.	Aurora PostgreSQL	Byte per detik
`ReplicationSlotDiskUsage`	Jumlah ruang disk yang dikonsumsi oleh file slot replikasi.	Aurora PostgreSQL	Byte
`ResultSetCacheHitRatio`	Persentase permintaan yang dilayani oleh cache Resultset.	Aurora MySQL	Persentase
`RollbackSegmentHistoryListLength`	Log undo yang mencatat transaksi yang dilakukan dengan catatan yang ditandai hapus. Catatan ini dijadwalkan untuk diproses oleh operasi purge InnoDB.	Aurora MySQL	Hitungan
`RowLockTime`	Total waktu yang dihabiskan guna mendapatkan kunci baris untuk tabel InnoDB.	Aurora MySQL	Milidetik
`SelectLatency`	Jumlah waktu rata-rata untuk operasi pemilihan.	Aurora MySQL	Milidetik

Metrik	Deskripsi	Berlaku untuk	Unit
SelectThroughput	Jumlah rata-rata kueri pemilihan per detik.	Aurora MySQL	Hitungan per detik
StorageNetworkReceiveThroughput	Jumlah throughput jaringan yang diterima dari subsistem penyimpanan Aurora oleh setiap instans di klaster DB.	Aurora MySQL dan Aurora PostgreSQL	Byte per detik
StorageNetworkThroughput	Jumlah throughput jaringan yang diterima dari dan dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB.	Aurora MySQL dan Aurora PostgreSQL	Byte per detik
StorageNetworkTransmitThroughput	Jumlah throughput jaringan yang dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB.	Aurora MySQL dan Aurora PostgreSQL	Byte per detik
SumBinaryLogSize	Ukuran total file binlog.	Aurora MySQL	Byte
SwapUsage	Jumlah ruang swap yang digunakan. Metrik ini tidak tersedia untuk kelas instans DB berikut: • db.r3.*, db.r4.*, dan db.r7g.* (Aurora MySQL) • db.r7g.* (Aurora PostgreSQL)	Aurora MySQL dan Aurora PostgreSQL	Byte

Metrik	Deskripsi	Berlaku untuk	Unit
`TempStorageIOPS`	Jumlah IOPS untuk baca dan tulis pada penyimpanan lokal yang dilampirkan ke instans DB. Metrik ini merepresentasikan hitungan dan diukur sekali per detik. Metrik ini berlaku untuk Aurora Nirserver v1 dan Aurora Nirserver v2.	Aurora MySQL dan Aurora PostgreSQL	Hitungan per detik
`TempStorageThroughput`	Jumlah data yang ditransfer ke dan dari penyimpanan lokal yang terkait dengan instans DB. Metrik ini merepresentasikan byte dan diukur sekali per detik. Metrik ini berlaku untuk Aurora Nirserver v1 dan Aurora Nirserver v2.	Aurora MySQL dan Aurora PostgreSQL	Byte per detik

Metrik	Deskripsi	Berlaku untuk	Unit
`TransactionLogsDiskUsage`	Jumlah ruang disk yang digunakan oleh log transaksi di instans DB Aurora PostgreSQL. Metrik ini dihasilkan hanya ketika Aurora PostgreSQL menggunakan replikasi logis atau. AWS Database Migration Service Secara default, Aurora PostgreSQL menggunakan catatan log, bukan log transaksi. Saat log transaksi tidak digunakan, nilai untuk metrik ini adalah `-1`.	Primer untuk Aurora PostgreSQL	Byte
`UpdateLatency`	Jumlah waktu rata-rata yang digunakan untuk operasi pembaruan.	Aurora MySQL	Milidetik
`UpdateThroughput`	Jumlah rata-rata pembaruan per detik.	Aurora MySQL	Hitungan per detik
`WriteIOPS`	Jumlah catatan penulisan penyimpanan Aurora yang dihasilkan per detik. Metrik ini kurang lebih adalah jumlah catatan log yang dihasilkan oleh basis data. Metrik ini tidak sesuai dengan penulisan halaman 8K, dan tidak sesuai dengan paket jaringan yang dikirim.	Aurora MySQL dan Aurora PostgreSQL	Hitungan per detik

Metrik	Deskripsi	Berlaku untuk	Unit
`WriteIOPSEphemeralStorage`	Jumlah rata-rata operasi I/O tulis disk ke penyimpanan NVMe Efemeral.	Aurora PostgreSQL	Hitungan per detik
`WriteLatency`	Jumlah waktu rata-rata yang digunakan per operasi I/O disk.	Aurora MySQL dan Aurora PostgreSQL	Detik
`WriteLatencyEphemeralStorage`	Jumlah waktu rata-rata yang digunakan per operasi I/O tulis disk untuk penyimpanan NVMe Efemeral.	Aurora PostgreSQL	Milidetik
`WriteThroughput`	Jumlah rata-rata byte yang ditulis ke penyimpanan persisten setiap detik.	Aurora MySQL dan Aurora PostgreSQL	Byte per detik
`WriteThroughputEphemeralStorage`	Jumlah rata-rata byte yang ditulis ke disk per detik untuk penyimpanan NVMe Efemeral.	Aurora PostgreSQL	Byte per detik

Metrik CloudWatch penggunaan Amazon untuk

`AWS/Usage`Namespace di Amazon CloudWatch menyertakan metrik penggunaan tingkat akun untuk kuota layanan Amazon RDS Anda. CloudWatch mengumpulkan metrik penggunaan secara otomatis untuk semua. Wilayah AWS

Untuk informasi selengkapnya, lihat metrik CloudWatch penggunaan di Panduan CloudWatch Pengguna Amazon. Untuk informasi selengkapnya tentang kuota, lihat Kuota dan batasan untuk Amazon Aurora dan Meminta peningkatan kuota dalam Panduan Pengguna Service Quotas.

Metrik	Deskripsi	Unit*
DBCluster ParameterGroups	Jumlah grup parameter klaster DB di Akun AWS Anda. Hitungan tidak termasuk grup parameter default.	Hitungan
DBClusters	Jumlah klaster DB Amazon Aurora di Akun AWS Anda.	Hitungan
DBInstances	Jumlah instans DB di Akun AWS Anda.	Hitungan
DBParamet erGroups	Jumlah grup parameter DB di Akun AWS Anda. Hitungan tidak termasuk grup parameter DB default.	Hitungan
DBSubnetGroups	Jumlah grup subnet DB di Akun AWS Anda. Hitungan tidak termasuk grup subnet default.	Hitungan
ManualClu sterSnapshots	Jumlah snapshot klaster DB yang dibuat secara manual di Akun AWS Anda. Hitungan tidak termasuk snapshot yang tidak valid.	Hitungan
OptionGroups	Jumlah grup opsi di Akun AWS Anda. Hitungan tidak termasuk grup opsi default.	Hitungan
ReservedD BInstances	Jumlah instans DB yang dialokasikan di Akun AWS Anda. Hitungan tidak termasuk instans yang dipensiunkan atau ditolak.	Hitungan

* Amazon RDS tidak mempublikasikan unit untuk metrik penggunaan. CloudWatch Unit hanya muncul dalam dokumentasi.

Dimensi-dimensi Amazon CloudWatch untuk Aurora

Anda dapat memfilter data metrik Aurora dengan menggunakan dimensi apa pun dalam tabel berikut.

Dimensi	Memfilter data yang diminta untuk . . .
DBInstanceIdentifier	Instans DB tertentu.
DBClusterIdentifier	Klaster DB Aurora tertentu.

Dimensi	Memfilter data yang diminta untuk . . .
`DBClusterIdentifier, Role`	Klaster DB Aurora tertentu, dengan mengagregasikan metrik itu dengan peran instans (PENULIS/PEMBACA). Misalnya, Anda dapat mengagregasikan metrik-metrik untuk semua instans PEMBACA milik sebuah klaster.
`DbClusterIdentifier, EngineName`	Kombinasi nama klaster dan mesin DB Aurora tertentu. Misalnya, Anda dapat menampilkan metrik `VolumeReadIOPs` untuk klaster `ams1` dan mesin `aurora`.
`DatabaseClass`	Semua instans dalam kelas basis data. Misalnya, Anda dapat mengagregasikan metrik-metrik untuk semua instans kelas basis data `db.r5.large` .
`EngineName`	Hanya nama mesin diidentifikasi. Misalnya, Anda dapat mengagregasikan metrik-metrik untuk semua instans yang memiliki nama mesin `aurora-postgresql` .
`SourceRegion`	Hanya Wilayah yang ditentukan. Misalnya, Anda dapat mengagregasikan metrik-metrik untuk semua instans basis data dalam Wilayah `us-east-1` .

Ketersediaan metrik Aurora di konsol Amazon RDS

Tidak semua metrik yang disediakan oleh Amazon Aurora tersedia di konsol Amazon RDS. Anda dapat melihat metrik ini menggunakan alat seperti AWS CLI CloudWatch dan API. Selain itu, beberapa metrik di konsol Amazon RDS ditunjukkan hanya untuk kelas instans tertentu, atau dengan nama dan unit ukur yang berbeda-beda.

Topik

- Metrik Aurora tersedia dalam tampilan Jam Terakhir
- Metrik Aurora yang tersedia dalam kasus tertentu
- Metrik Aurora yang tidak tersedia di konsol

Metrik Aurora tersedia dalam tampilan Jam Terakhir

Anda dapat melihat subset metrik Aurora yang dikategorikan dalam tampilan Jam Terakhir default di konsol Amazon RDS. Daftar tabel berikut mencantumkan kategori dan metrik terkait yang ditampilkan di konsol Amazon RDS untuk instans Aurora.

Kategori	Metrik

SQL	`ActiveTransactions`
	`BlockedTransactions`
	`BufferCacheHitRatio`
	`CommitLatency`
	`CommitThroughput`
	`DatabaseConnections`
	`DDLLatency`
	`DDLThroughput`
	`Deadlocks`
	`DMLLatency`
	`DMLThroughput`
	`LoginFailures`
	`ResultSetCacheHitRatio`
	`SelectLatency`
	`SelectThroughput`

Sistem	`AuroraReplicaLag` `AuroraReplicaLagMaximum`

Kategori	Metrik

	`AuroraReplicaLagMinimum`
	
	`CPUCreditBalance`
	
	`CPUCreditUsage`
	
	`CPUUtilization`
	
	`FreeableMemory`
	
	`FreeLocalStorage` (Hal ini tidak berlaku untuk Aurora Serverless v2.)
	
	`NetworkReceiveThroughput`

Deployment	`AuroraReplicaLag`
	
	`BufferCacheHitRatio`
	
	`ResultSetCacheHitRatio`
	
	`SelectThroughput`

Metrik Aurora yang tersedia dalam kasus tertentu

Selain itu, beberapa metrik Aurora ditunjukkan hanya untuk kelas instans tertentu, atau hanya untuk instans DB, atau dengan nama dan unit ukur yang berbeda-beda:

- Metrik `CPUCreditBalance` dan `CPUCreditUsage` hanya ditampilkan untuk kelas instans `db.t2` Aurora MySQL dan kelas instans `db.t3` Aurora PostgreSQL.
- Metrik berikut ini ditampilkan dengan nama yang berbeda-beda, seperti yang tercantum:

Metrik	Nama tampilan
`AuroraReplicaLagMaximum`	Lag replika maksimum
`AuroraReplicaLagMinimum`	Lag replika minimum

Metrik	Nama tampilan
DDLThroughput	DDL
NetworkReceiveThroughput	Throughput jaringan
VolumeBytesUsed	[Ditagih] Byte volume yang digunakan
VolumeReadIOPs	[Ditagih] IOPS baca volume
VolumeWriteIOPs	[Ditagih] IOPS tulis volume

- Metrik berikut berlaku untuk seluruh klaster DB Aurora, tetapi ditampilkan hanya saat melihat instans DB untuk klaster DB Aurora di konsol Amazon RDS:
  - VolumeBytesUsed
  - VolumeReadIOPs
  - VolumeWriteIOPs
- Metrik berikut ditampilkan dalam megabyte, bukan byte, di konsol Amazon RDS:
  - FreeableMemory
  - FreeLocalStorage
  - NetworkReceiveThroughput
  - NetworkTransmitThroughput
- Metrik berikut berlaku untuk cluster DB PostgreSQL Aurora dengan Aurora Optimized Reads:
  - AuroraOptimizedReadsCacheHitRatio
  - FreeEphemeralStorage
  - ReadIOPSEphemeralStorage
  - ReadLatencyEphemeralStorage
  - ReadThroughputEphemeralStorage
  - WriteIOPSEphemeralStorage
  - WriteLatencyEphemeralStorage
  - WriteThroughputEphemeralStorage

Metrik Aurora yang tidak tersedia di konsol

Metrik Aurora berikut tidak tersedia di konsol Amazon RDS:

- AuroraBinlogReplicaLag
- DeleteLatency
- DeleteThroughput
- EngineUptime
- InsertLatency
- InsertThroughput
- NetworkThroughput
- Queries
- UpdateLatency
- UpdateThroughput

CloudWatch Metrik Amazon untuk Performance Insights

Performance Insights secara otomatis menerbitkan beberapa metrik ke Amazon. CloudWatch Data yang sama dapat ditanyakan dari Performance Insights, tetapi memiliki metrik CloudWatch memudahkan untuk menambahkan alarm. CloudWatch Ini juga memudahkan untuk menambahkan metrik ke CloudWatch Dasbor yang ada.

Metrik	Deskripsi
DBLoad	Jumlah sesi aktif untuk mesin DB. Biasanya, Anda menginginkan data untuk jumlah rata-rata sesi aktif. Dalam Wawasan Performa, data ini dikueri sebagai `db.load.avg` .
DBLoadCPU	Jumlah sesi aktif dengan jenis peristiwa tunggu berupa CPU. Dalam Wawasan Performa, data ini dikueri sebagai `db.load.avg` , difilter berdasarkan jenis peristiwa tunggu `CPU`.
LoadNonCPU DB	Jumlah sesi aktif dengan jenis peristiwa tunggu bukan CPU.

> **Note**
>
> Metrik ini dipublikasikan CloudWatch hanya jika ada beban pada instans DB.

Anda dapat memeriksa metrik ini menggunakan CloudWatch konsol, the AWS CLI, atau CloudWatch API. Anda juga dapat memeriksa metrik penghitung Performance Insights lainnya menggunakan fungsi matematika metrik khusus. Untuk informasi selengkapnya, lihat Menanyakan metrik penghitung Performance Insights lainnya di CloudWatch.

Misalnya, Anda bisa mendapatkan statistik untuk `DBLoad` metrik dengan menjalankan get-metric-statisticsperintah.

```
aws cloudwatch get-metric-statistics \
    --region us-west-2 \
    --namespace AWS/RDS \
    --metric-name DBLoad  \
    --period 60 \
    --statistics Average \
    --start-time 1532035185 \
    --end-time 1532036185 \
    --dimensions Name=DBInstanceIdentifier,Value=db-loadtest-0
```

Contoh ini menghasilkan output yang terlihat seperti berikut.

```
{
  "Datapoints": [
  {
  "Timestamp": "2021-07-19T21:30:00Z",
  "Unit": "None",
  "Average": 2.1
  },
  {
  "Timestamp": "2021-07-19T21:34:00Z",
  "Unit": "None",
  "Average": 1.7
  },
  {
  "Timestamp": "2021-07-19T21:35:00Z",
  "Unit": "None",
  "Average": 2.8
  },
```

```
{
"Timestamp": "2021-07-19T21:31:00Z",
"Unit": "None",
"Average": 1.5
},
{
"Timestamp": "2021-07-19T21:32:00Z",
"Unit": "None",
"Average": 1.8
},
{
"Timestamp": "2021-07-19T21:29:00Z",
"Unit": "None",
"Average": 3.0
},
{
"Timestamp": "2021-07-19T21:33:00Z",
"Unit": "None",
"Average": 2.4
}
],
"Label": "DBLoad"
}
```


Untuk informasi selengkapnya CloudWatch, lihat Apa itu Amazon CloudWatch? di Panduan CloudWatch Pengguna Amazon.

Menanyakan metrik penghitung Performance Insights lainnya di CloudWatch

Anda dapat melakukan kueri, alarm, dan grafik pada metrik Performance Insights RDS. CloudWatch Anda dapat mengakses informasi tentang Anda dengan menggunakan fungsi matematika `DB_PERF_INSIGHTS` metrik untuk CloudWatch. Fungsi ini memungkinkan Anda menggunakan metrik Performance Insights yang tidak dilaporkan secara langsung CloudWatch untuk membuat deret waktu baru.

Anda dapat menggunakan fungsi Matematika Metrik baru dengan mengklik menu tarik-turun Tambahkan Matematika di layar Select metric di CloudWatch konsol. Anda dapat menggunakannya untuk membuat alarm dan grafik pada metrik Performance Insights atau pada kombinasi dan metrik CloudWatch Performance Insights, termasuk alarm resolusi tinggi untuk metrik sub-menit. Anda juga dapat menggunakan fungsi secara terprogram dengan menyertakan ekspresi Matematika Metrik dalam permintaan. `get-metric-data` Untuk informasi selengkapnya, lihat Sintaks dan fungsi

matematika metrik dan Membuat alarm pada metrik penghitung Performance Insights dari database. AWS

Metrik penghitung Wawasan Performa

Metrik penghitung adalah metrik performa sistem operasi dan basis data di dasbor Wawasan Performa. Untuk membantu mengidentifikasi dan menganalisis masalah performa, Anda dapat mengaitkan metrik penghitung dengan muatan DB. Anda dapat menambahkan fungsi statistik ke metrik untuk mendapatkan nilai metrik. Misalnya, fungsi yang didukung untuk metrik `os.memory.active` adalah `.avg`, `.min`, `.max`, `.sum`, dan `.sample_count`.

Metrik penghitung dikumpulkan sekali setiap menit. Kumpulan metrik OS bergantung pada apakah Pemantauan yang Ditingkatkan diaktifkan atau dinonaktifkan. Jika Pemantauan yang Ditingkatkan dinonaktifkan, metrik OS dikumpulkan sekali setiap menit. Jika Pemantauan yang Ditingkatkan diaktifkan, metrik OS dikumpulkan untuk periode waktu yang dipilih. Untuk informasi selengkapnya tentang cara mengaktifkan atau menonaktifkan Pemantauan yang Ditingkatkan, lihat Mengaktifkan dan menonaktifkan Pemantauan yang Ditingkatkan.

Topik

- Penghitung sistem operasi Wawasan Performa
- Penghitung Wawasan Performa untuk Aurora MySQL
- Penghitung Wawasan Performa untuk Aurora PostgreSQL

Penghitung sistem operasi Wawasan Performa

Penghitung sistem operasi berikut, yang diawali dengan `os`, tersedia di Wawasan Performa untuk Aurora PostgreSQL dan Aurora MySQL.

Anda dapat menggunakan API `ListAvailableResourceMetrics` untuk mengetahui daftar metrik penghitung yang tersedia untuk instans DB Anda. Untuk informasi selengkapnya, lihat ListAvailableResourceMetricsdi panduan Referensi API Amazon RDS Performance Insights.

Penghitung	Jenis	Metrik	Deskripsi
Aktif	Memori	os.memory.active	Jumlah memori yang ditetapkan, dalam kilobyte.

Penghitung	Jenis	Metrik	Deskripsi
Buffer	Memori	os.memory.buffers	Jumlah memori yang digunakan untuk buffering permintaan I/O sebelum menulis ke perangkat penyimpanan, dalam kilobyte.
Di-cache	Memori	os.memory.cached	Jumlah memori yang digunakan untuk meng-cache I/O berbasis sistem file, dalam kilobyte.
Cache DB	Memori	os.memory.db.cache	Jumlah memori yang digunakan untuk cache halaman oleh proses basis data termasuk tmpfs (shmem), dalam byte.
Ukuran Set Residen DB	Memori	os.memory.db.residentSetSize	Jumlah memori yang digunakan untuk cache anonim dan swap oleh proses basis data tidak termasuk tmpfs (shmem), dalam byte.
Swap DB	Memori	os.memory.db.swap	Jumlah memori yang digunakan untuk swap berdasarkan proses basis data, dalam byte.

Penghitung	Jenis	Metrik	Deskripsi
Kotor	Memori	os.memory.dirty	Jumlah halaman memori dalam RAM yang telah diubah, tetapi tidak ditulis ke blok data terkaitnya dalam penyimpanan, dalam kilobyte.
Kosong	Memori	os.memory.free	Jumlah memori yang tidak ditetapkan, dalam kilobyte.
Halaman Besar Kosong	Memori	os.memori. hugePagesFree	Jumlah halaman besar yang kosong. Halaman besar adalah fitur dari kernel Linux.
Rsvd Halaman Besar	Memori	os.memori. hugePagesRsvd	Jumlah halaman besar yang khusus.
Ukuran Halaman Besar	Memori	os.memori. hugePagesSize	Ukuran untuk setiap unit halaman besar, dalam kilobyte.
Surplus Halaman Besar	Memori	os.memori. hugePagesSurp	Jumlah halaman besar surplus yang tersedia terhadap total.
Total Halaman Besar	Memori	os.memori. hugePagesTotal	Jumlah total halaman besar.

Penghitung	Jenis	Metrik	Deskripsi
Nonaktif	Memori	os.memory.inactive	Jumlah halaman memori yang jarang digunakan, dalam kilobyte.
Dipetakan	Memori	os.memory.mapped	Jumlah total konten sistem file yang memorinya dipetakan di dalam ruang alamat proses, dalam kilobyte.
Jumlah Kill Kehabisan Memori	Memori	os.memori. outOfMemoryKillCount	Jumlah kill OOM yang terjadi selama interval pengumpulan terakhir.
Tabel Halaman	Memori	os.memory.pageTables	Jumlah memori yang digunakan berdasarkan tabel halaman, dalam kilobyte.
Lempengan	Memori	os.memory.slab	Jumlah struktur data kernel yang dapat digunakan kembali, dalam kilobyte.
Total	Memori	os.memory.total	Jumlah total memori, dalam kilobyte.
Writeback	Memori	os.memory.writeback	Jumlah halaman kotor dalam RAM yang masih ditulis ke penyimpanan cadangan, dalam kilobyte.

Penghitung	Jenis	Metrik	Deskripsi
Tamu	Penggunaan CPU	os.cpuUtilization.guest	Persentase CPU yang digunakan oleh program tamu.
Idle	Penggunaan CPU	os.cpuUtilization.idle	Persentase CPU saat idle.
Irq	Penggunaan CPU	os.cpuUtilization.irq	Persentase CPU yang digunakan oleh gangguan perangkat lunak.
Nice	Penggunaan CPU	os.cpuUtilization.nice	Persentase CPU yang digunakan oleh program yang berjalan pada prioritas terendah.
Steal	Penggunaan CPU	os.cpuUtilization.steal	Persentase CPU yang digunakan oleh mesin virtual lainnya.
Sistem	Penggunaan CPU	os.cpuUtilization.system	Persentase CPU yang digunakan oleh kernel.
Total	Penggunaan CPU	os.cpuUtilization.total	Total persentase CPU yang digunakan. Nilai ini mencakup nilai nice.
Pengguna	Penggunaan CPU	os.cpuUtilization.user	Persentase CPU yang digunakan oleh program pengguna.

Penghitung	Jenis	Metrik	Deskripsi
Tunggu	Penggunaan CPU	os.cpuUtilization.wait	Persentase CPU yang tidak digunakan saat menunggu akses I/O.
Penyimpanan Aurora Penyimpanan Aurora Byte Rx	IO Disk	os.Diskio.AuroraStorage. auroraStorageBytesRx	Jumlah byte yang diterima untuk penyimpanan Aurora per detik.
Penyimpanan Aurora Penyimpanan Aurora Byte Tx	IO Disk	os.Diskio.AuroraStorage. auroraStorageBytesTx	Jumlah byte yang diunggah untuk penyimpanan Aurora per detik.
Kedalaman Antrean Disk Penyimpanan Aurora	IO Disk	os.Diskio.AuroraStorage. diskQueueDepth	Panjang antrean disk penyimpanan Aurora.
PS IOs Baca Penyimpanan Aurora	IO Disk	os.diskIO.auroraStorage.readIOsPS	Jumlah operasi baca per detik.
Latensi Baca Penyimpanan Aurora	IO Disk	os.diskIO.auroraStorage.readLatency	Waktu yang berlalu antara pengiriman permintaan I/O baca dan penyelesaiannya, dalam milidetik.
Throughtput Baca Penyimpanan Aurora	IO Disk	os.diskIO.auroraStorage.readThroughput	Jumlah throughput jaringan yang digunakan oleh permintaan ke klaster DB, dalam byte per detik.
PS IO Tulis Penyimpanan Aurora	IO Disk	os.diskIO.auroraStorage.writeIOsPS	Jumlah operasi tulis per detik.

Penghitung	Jenis	Metrik	Deskripsi
Latensi Tulis Penyimpanan Aurora	IO Disk	os.diskIO.auroraStorage.writeLatency	Waktu yang berlalu antara pengiriman permintaan I/O tulis dan penyelesaiannya, dalam milidetik.
Throughput Tulis Penyimpanan Aurora	IO Disk	os.diskIO.auroraStorage.writeThroughput	Jumlah throughput jaringan yang digunakan oleh respons dari klaster DB, dalam byte per detik.
Len Antrean Rata-rata Rdstemp	IO Disk	os.Diskio.RDStemp.avgQueueLen	Jumlah permintaan yang menunggu dalam antrean perangkat I/O.
Sz Req Rata-rata Rdstemp	IO Disk	os.Diskio.RDStemp.avgReqSz	Jumlah permintaan yang menunggu dalam antrean perangkat I/O.
Menunggu Rdstemp	IO Disk	os.diskIO.rdstemp.await	Jumlah milidetik yang diperlukan untuk merespons permintaan, termasuk waktu antrean dan waktu layanan.
PS IO Baca Rdstemp	IO Disk	os.diskIO.rdstemp.readIOsPS	Jumlah operasi baca per detik.
KB Baca Rdstemp	IO Disk	os.diskIO.rdstemp.readKb	Jumlah total kilobyte yang dibaca.

Penghitung	Jenis	Metrik	Deskripsi
PS KB Baca Rdstemp	IO Disk	os.diskIO.rdstemp.readKbPS	Jumlah kilobyte yang dibaca per detik.
PS Rrqm Rdstemp	IO Disk	os.diskIO.rdstemp.rrqmPS	Jumlah permintaan baca gabungan yang diantrekan per detik.
TPS Rdstemp	IO Disk	os.diskIO.rdstemp.tps	Jumlah transaksi I/O per detik.
Utilitas Rdstemp	IO Disk	os.diskIO.rdstemp.util	Persentase waktu CPU saat permintaan dikeluarkan.
PS IO Tulis Rdstemp	IO Disk	os.diskIO.rdstemp.writeIOsPS	Jumlah operasi tulis per detik.
KB Tulis Rdstemp	IO Disk	os.diskIO.rdstemp.writeKb	Jumlah total kilobyte yang ditulis.
PS KB Tulis Rdstemp	IO Disk	os.diskIO.rdstemp.writeKbPS	Jumlah kilobyte yang ditulis per detik.
PS Wrqm Rdstemp	IO Disk	os.diskIO.rdstemp.wrqmPS	Jumlah permintaan tulis gabungan yang diantrekan per detik.
Diblokir	Tugas	os.tasks.blocked	Jumlah tugas yang diblokir.
Berjalan	Tugas	os.tasks.running	Jumlah tugas yang berjalan.
Tidur	Tugas	os.tasks.sleeping	Jumlah tugas yang tidur.

Penghitung	Jenis	Metrik	Deskripsi
Dihentikan	Tugas	os.tasks.stopped	Jumlah tugas yang dihentikan.
Total	Tugas	os.tasks.total	Jumlah total tugas.
Zombie	Tugas	os.tasks.zombie	Jumlah tugas turunan yang tidak aktif dengan tugas induk aktif.
Satu	Menit Rata-Rata Muatan	os. loadAvera geMinute.satu	Jumlah proses yang meminta waktu CPU selama satu menit terakhir.
Lima belas	Menit Rata-Rata Muatan	os. loadAvera geMinute.lima belas	Jumlah proses yang meminta waktu CPU selama 15 menit terakhir.
Lima	Menit Rata-Rata Muatan	os. loadAvera geMinute.lima	Jumlah proses yang meminta waktu CPU selama 5 menit terakhir.
Di-cache	Swap	os.swap.cached	Jumlah memori swap, dalam kilobyte, yang digunakan sebagai memori cache.
Kosong	Swap	os.swap.free	Jumlah memori swap yang kosong, dalam kilobyte.

Penghitung	Jenis	Metrik	Deskripsi
Masuk	Swap	os.swap.in	Jumlah memori, dalam kilobyte, yang ditukar ke dalam dari disk.
Keluar	Swap	os.swap.out	Jumlah memori, dalam kilobyte, yang ditukar ke luar dari disk.
Total	Swap	os.swap.total	Jumlah total memori swap yang tersedia dalam kilobyte.
File Maks	Sistem File	os.fileSys.maxFiles	Jumlah maksimum file yang dapat dibuat untuk sistem file.
File yang Digunakan	Sistem File	os.fileSys.usedFiles	Jumlah file dalam sistem file.
Persen File Digunakan	Sistem File	os.Filesys. usedFilePercent	Persentase file tersedia yang digunakan.
Persen Digunakan	Sistem File	os.fileSys.usedPercent	Persentase ruang disk sistem file yang digunakan.
Digunakan	Sistem File	os.fileSys.used	Jumlah ruang disk yang digunakan oleh file dalam sistem file, dalam kilobyte.

Penghitung	Jenis	Metrik	Deskripsi
Total	Sistem File	os.fileSys.total	Jumlah total ruang disk yang tersedia untuk sistem file, dalam kilobyte.
Rx	Jaringan	os.network.rx	Jumlah byte yang diterima per detik.
Tx	Jaringan	os.network.tx	Jumlah byte yang diunggah per detik.
Penggunaan Acu	Umum	os.general.acuUtilization	Persentase kapasitas saat ini dari kapasitas maksimum yang dikonfigurasi.
Acu Maks yang Dikonfigurasi	Umum	os.umum. maxConfiguredAcu	Kapasitas maksimum yang dikonfigurasi oleh pengguna, di ACU.
Acu Min yang Dikonfigurasi	Umum	os.umum. minConfiguredAcu	Kapasitas minimum yang dikonfigurasi oleh pengguna, di ACU.
Jumlah vCPU	Umum	os.general.numVCPUs	Jumlah CPU virtual untuk instans DB.
Kapasitas Basis Data Nirserver	Umum	os.umum. serverlessDatabaseCapacity	Kapasitas instans saat ini, dalam ACU.

Penghitung Wawasan Performa untuk Aurora MySQL

Penghitung basis data berikut ini tersedia di Wawasan Performa untuk Aurora MySQL.

Topik

- Penghitung native untuk Aurora MySQL
- Penghitung non-native untuk Aurora MySQL

Penghitung native untuk Aurora MySQL

Metrik native ditentukan oleh mesin basis data, bukan Amazon Aurora. Anda dapat menemukan definisi metrik native ini di Server status variables dalam dokumentasi MySQL.

Penghitung	Jenis	Unit	Metrik
Com_analyze	SQL	Kueri per detik	db.SQL.Com_analyze
Com_optimize	SQL	Kueri per detik	db.SQL.Com_optimize
Com_select	SQL	Kueri per detik	db.SQL.Com_select
Innodb_rows_deleted	SQL	Baris per detik	db.SQL.Innodb_rows_deleted
Innodb_rows_inserted	SQL	Baris per detik	db.SQL.Innodb_rows_inserted
Innodb_rows_read	SQL	Baris per detik	db.SQL.Innodb_rows_read
Innodb_rows_updated	SQL	Baris per detik	db.SQL.Innodb_rows_updated
Kueri	SQL	Kueri per detik	db.SQL.Queries
Pertanyaan	SQL	Kueri per detik	db.SQL.Questions
Select_full_join	SQL	Kueri per detik	db.SQL.Select_full_join

Penghitung	Jenis	Unit	Metrik
Select_full_range_join	SQL	Kueri per detik	db.SQL.Select_full_range_join
Select_range	SQL	Kueri per detik	db.SQL.Select_range
Select_range_check	SQL	Kueri per detik	db.SQL.Select_range_check
Select_scan	SQL	Kueri per detik	db.SQL.Select_scan
Slow_queries	SQL	Kueri per detik	db.SQL.Slow_queries
Sort_merge_passes	SQL	Kueri per detik	db.SQL.Sort_merge_passes
Sort_range	SQL	Kueri per detik	db.SQL.Sort_range
Sort_rows	SQL	Kueri per detik	db.SQL.Sort_rows
Sort_scan	SQL	Kueri per detik	db.SQL.Sort_scan
Total_query_time	SQL	Milidetik	db.SQL.Total_query_time
Table_locks_immediate	Kunci	Permintaan per detik	db.Locks.Table_locks_immediate
Table_locks_waited	Kunci	Permintaan per detik	db.Locks.Table_locks_waited
Innodb_row_lock_time	Kunci	Milidetik (rata-rata)	db.Locks.Innodb_row_lock_time

Penghitung	Jenis	Unit	Metrik
Aborted_clients	Pengguna	Koneksi	db.Users.Aborted_clients
Aborted_connects	Pengguna	Koneksi	db.Users.Aborted_connects
Koneksi	Pengguna	Koneksi	db.Users.Connections
External_threads_connected	Pengguna	Koneksi	db.Users.External_threads_connected
max_connections	Pengguna	Koneksi	db.User.max_connections
Threads_connected	Pengguna	Koneksi	db.Users.Threads_connected
Threads_created	Pengguna	Koneksi	db.Users.Threads_created
Threads_running	Pengguna	Koneksi	db.Users.Threads_running
Created_tmp_disk_tables	Temp	Tabel per detik	db.Temp.Created_tmp_disk_tables
Created_tmp_tables	Temp	Tabel per detik	db.Temp.Created_tmp_tables
Innodb_buffer_pool_pages_data	Cache	Halaman	db.Cache.Innodb_buffer_pool_pages_data
Innodb_buffer_pool_pages_total	Cache	Halaman	db.Cache.Innodb_buffer_pool_pages_total
Innodb_buffer_pool_read_requests	Cache	Halaman per detik	db.Cache.Innodb_buffer_pool_read_requests
Innodb_buffer_pool_reads	Cache	Halaman per detik	db.Cache.Innodb_buffer_pool_reads
Opened_tables	Cache	Tabel	db.Cache.Opened_tables
Opened_table_definitions	Cache	Tabel	db.Cache.Opened_table_definitions
Qcache_hits	Cache	Kueri	db.Cache.Qcache_hits

Penghitung non-native untuk Aurora MySQL

Metrik penghitung non-native adalah penghitung yang ditentukan oleh Amazon RDS. Metrik non-asli bisa berupa metrik yang Anda dapatkan dengan kueri tertentu. Metrik non-native juga bisa berupa metrik turunan, dengan dua penghitung native atau lebih digunakan dalam penghitungan untuk mengetahui rasio, tingkat hit, atau latensi.

Penghitung	Jenis	Metrik	Deskripsi	Definisi
innodb_buffer_pool_hits	Cache	db.Cache.innoDB_buffer_pool_hits	Jumlah bacaan yang dapat dipenuhi oleh InnoDB dari kumpulan buffer.	`innodb_buffer_pool_read_requests - innodb_buffer_pool_reads`
innodb_buffer_pool_hit_rate	Cache	db.Cache.innoDB_buffer_pool_hit_rate	Jumlah baca yang dapat dipenuhi oleh InnoDB dari kumpulan buffer.	`100 * innodb_buffer_pool_read_requests / (innodb_buffer_pool_read_requests + innodb_buffer_pool_reads)`
innodb_buffer_pool_usage	Cache	db.Cache.innoDB_buffer_pool_usage	Persentase kumpulan buffer InnoDB yang berisi data (halaman). **Note** Saat menggunakan tabel terkompresi, nilai ini bisa bervariasi.	`Innodb_buffer_pool_pages_data / Innodb_buffer_pool_pages_total * 100.0`

Penghitung	Jenis	Metrik	Deskripsi	Definisi
			Untuk informasi selengkapnya, lihat informasi tentang `Innodb_buffer_pool_pages_data` dan `Innodb_buffer_pool_pages_total` di Server status variables dalam dokumentasi MySQL.	
query_cache_hit_rate	Cache	db.Cache.query_cache_hit_rate	Rasio hit untuk cache set hasil MySQL (cache kueri).	`Qcache_hits / (QCache_hits + Com_select) * 100`
innodb_rows_changed	SQL	db.SQL.innodb_rows_changed	Total operasi baris InnoDB.	`db.SQL.Innodb_rows_inserted + db.SQL.Innodb_rows_deleted + db.SQL.Innodb_rows_updated`

Penghitung	Jenis	Metrik	Deskripsi	Definisi
active_transactions	Transak	db.Transactions.active_transactions	Total transaksi aktif.	`SELECT COUNT(1) AS active_transactions FROM INFORMATION_SCHEMA.INNODB_TRX`
trx_rseg_history_len	Transak	db.Transactions.trx_rseg_history_len	Daftar halaman log undo untuk transaksi yang telah dijalankan yang dipertahankan oleh sistem transaksi InnoDB untuk menerapkan kontrol konkurensi multi-versi. Untuk informasi selengkapnya tentang detail data log undo, lihat https://dev.mysql.com/doc/refman/8.0/en/innodb-multi-versioning.html dalam dokumentasi MySQL.	`SELECT COUNT AS trx_rseg_history_len FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='trx_rseg_history_len'`
innodb_deadlocks	Kunci	db.Locks.innodb_deadlocks	Jumlah total penguncian.	`SELECT COUNT AS innodb_deadlocks FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_deadlocks'`

Penghitung	Jenis	Metrik	Deskripsi	Definisi
innodb_lock_timeouts	Kunci	db.Locks.innodb_lock_timeouts	Jumlah total penguncian yang habis waktunya.	`SELECT COUNT AS innodb_lock_timeouts FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_timeouts'`
innodb_row_lock_waits	Kunci	db.Locks.innodb_row_lock_waits	Jumlah total kunci baris yang menghasilkan peristiwa tunggu.	`SELECT COUNT AS innodb_row_lock_waits FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='lock_row_lock_waits'`

Penghitung Wawasan Performa untuk Aurora PostgreSQL

Penghitung basis data berikut tersedia di Wawasan Performa untuk Aurora PostgreSQL.

Topik

- Penghitung Native untuk Aurora PostgreSQL
- Penghitung non-native untuk Aurora PostgreSQL

Penghitung Native untuk Aurora PostgreSQL

Metrik native ditentukan oleh mesin basis data, bukan Amazon Aurora. Anda dapat menemukan definisi untuk metrik native ini dalam Viewing Statistics dalam dokumentasi PostgreSQL.

Penghitung	Jenis	Unit	Metrik
queries_started	SQL	Kueri per detik	db.SQL.queries
total_query_time	SQL	Milidetik	db.SQL.total_query_time
tup_deleted	SQL	Tuple per detik	db.SQL.tup_deleted
tup_fetched	SQL	Tuple per detik	db.SQL.tup_fetched
tup_inserted	SQL	Tuple per detik	db.SQL.tup_inserted
tup_returned	SQL	Tuple per detik	db.SQL.tup_returned
tup_updated	SQL	Tuple per detik	db.SQL.tup_updated
blks_hit	Cache	Blok per detik	db.Cache.blks_hit
buffers_alloc	Cache	Blok per detik	db.Cache.buffers_alloc
buffers_checkpoint	Titik pemeriksaan	Blok per detik	db.Checkpoint.buffers_checkpoint
checkpoints_req	Titik pemeriksaan	Titik pemeriksaan per menit	db.Checkpoint.checkpoints_req
checkpoint_sync_time	Titik pemeriksaan	Milidetik per titik pemeriksaan	db.Checkpoint.checkpoint_sync_time
checkpoints_timed	Titik pemeriksaan	Titik pemeriksaan per menit	db.Checkpoint.checkpoints_timed
checkpoint_write_time	Titik pemeriksaan	Milidetik per titik pemeriksaan	db.Checkpoint.checkpoint_write_time
maxwritten_clean	Titik pemeriksaan	Penghentian pembersihan Bgwriter per menit	db.Checkpoint.maxwritten_clean

Penghitung	Jenis	Unit	Metrik
penguncian	Konkurensi	Penguncian per menit	db.Concurrency.deadlocks
blk_read_time	I/O	Milidetik	db.IO.blk_read_time
blks_read	I/O	Blok per detik	db.IO.blks_read
buffers_backend	I/O	Blok per detik	db.IO.buffers_backend
buffers_backend_fsync	I/O	Blok per detik	db.IO.buffers_backend_fsync
buffers_clean	I/O	Blok per detik	db.IO.buffers_clean
idle_in_transaction_aborted_count	Status	Sesi	db.State.idle_in_transaction_aborted_count
idle_in_transaction_count	Status	Sesi	db.State.idle_in_transaction_count
idle_in_transaction_max_time	Status	Detik	db.State.idle_in_transaction_max_time
temp_bytes	Temp	Byte per detik	db.Temp.temp_bytes
temp_files	Temp	File per menit	db.Temp.temp_files
active_transactions	Transaksi	Transaksi	db.Transactions.active_transactions
blocked_transactions	Transaksi	Transaksi	db.Transactions.blocked_transactions
duration_commits	Transaksi	Milidetik	db.Transactions.duration_commits

Penghitung	Jenis	Unit	Metrik
max_used_xact_ids	Transaksi	Transaksi	db.Transactions.max_used_xact_ids
xact_commit	Transaksi	Commit per detik	db.Transactions.xact_commit
xaxt_rollback	Transaksi	Pemulihan per detik	db.Transactions.xact_rollback
max_connections	Pengguna	Koneksi	db.User.max_connections
numbackends	Pengguna	Koneksi	db.User.numbackends
total_auth_attempts	Pengguna	Koneksi per menit	db.User.total_auth_attempts
archived_count	WAL	File per menit	db.WAL.archived_count
archive_failed_count	WAL	File per menit	db.WAL.archive_failed_count

Penghitung non-native untuk Aurora PostgreSQL

Metrik penghitung non-native adalah penghitung yang ditentukan oleh Amazon Aurora. Metrik non-asli bisa berupa metrik yang Anda dapatkan dengan kueri tertentu. Metrik non-native juga bisa berupa metrik turunan, dengan dua penghitung native atau lebih digunakan dalam penghitungan untuk mengetahui rasio, tingkat hit, atau latensi.

Penghitung	Jenis	Metrik	Deskripsi	Definisi
logical_reads	SQL	db.SQL.logical_reads	Jumlah total blok yang dicapai dan dibaca.	`blks_hit + blks_read`

Penghitung	Jenis	Metrik	Deskripsi	Definisi
checkpoint_sync_latency	Titik pemeriksaan	db.Checkpoint.checkpoint_sync_latency	Jumlah total waktu yang telah dihabiskan di bagian pemrosesan titik pemeriksaan tempat file disinkronkan ke disk.	`checkpoint_sync_time / (checkpoints_timed + checkpoints_req)`
checkpoint_write_latency	Titik pemeriksaan	db.Checkpoint.checkpoint_write_latency	Jumlah total waktu yang telah dihabiskan di bagian pemrosesan titik pemeriksaan tempat file ditulis ke disk.	`checkpoint_write_time / (checkpoints_timed + checkpoints_req)`
commit_latency	Transaksi	db.Transactions.commit_latency	Durasi rata-rata operasi commit.	`db.Transactions.duration_commits / db.Transactions.xact_commit`
local_blks_read	I/O	db.IO.local_blks_read	Jumlah total blok lokal yang dibaca.	-
local_blk_read_time	I/O	db.IO.local_blk_read_time	Jika `track_io_timing` diaktifkan, ini akan melacak total waktu yang dihabiskan untuk membaca blok file data lokal, dalam milidetik, jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.	-
orcache_blks_hit	I/O	db.IO.orcache_blks_hit	Jumlah total hit blok bersama dari cache pembacaan yang dioptimalkan.	-

Penghitung	Jenis	Metrik	Deskripsi	Definisi
orcache_blk_read_time	I/O	db.IO.orcache_blk_read_time	Jika `track_io_timing` diaktifkan, ini akan melacak total waktu yang dihabiskan untuk membaca blok file data dari cache Optimized Reads, dalam milidetik, jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.	-
read_latency	I/O	db.IO.read_latency	Waktu yang dihabiskan untuk membaca blok file data oleh backend dalam instans ini.	`blk_read_time / blks_read`
storage_blks_read	I/O	db.IO.storage_blks_read	Jumlah total blok bersama yang dibaca dari penyimpanan Aurora.	-
storage_blk_read_time	I/O	db.IO.storage_blk_read_time	Jika `track_io_timing` diaktifkan, ini akan melacak total waktu yang dihabiskan untuk membaca blok file data dari penyimpanan Aurora, dalam milidetik, jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.	-

Statistik SQL untuk Wawasan Performa

Statistik SQL adalah metrik terkait performa tentang kueri SQL yang dikumpulkan oleh Wawasan Performa. Wawasan Performa mengumpulkan statistik untuk setiap detik yang digunakan untuk menjalankan kueri dan untuk setiap panggilan SQL. Statistik SQL adalah rata-rata untuk rentang waktu yang dipilih.

Digest SQL adalah gabungan semua kueri yang memiliki pola tertentu, tetapi tidak harus memiliki nilai literal yang sama. Digest menggantikan nilai literal dengan tanda tanya. Sebagai contoh, `SELECT * FROM emp WHERE lname= ?`. Digest ini mungkin terdiri dari kueri turunan berikut:

```
SELECT * FROM emp WHERE lname = 'Sanchez'
SELECT * FROM emp WHERE lname = 'Olagappan'
SELECT * FROM emp WHERE lname = 'Wu'
```

Semua mesin mendukung statistik SQL untuk kueri digest.

Untuk informasi dukungan wilayah, mesin DB, dan kelas instans untuk fitur ini, lihat Dukungan kelas instans, Wilayah, dan mesin DB Amazon Aurora untuk fitur Wawasan Performa

Topik

- Statistik SQL untuk Aurora MySQL
- Statistik SQL untuk Aurora PostgreSQL

Statistik SQL untuk Aurora MySQL

Aurora MySQL mengumpulkan statistik SQL hanya pada tingkat digest. Tidak ada statistik yang ditampilkan di tingkat pernyataan.

Topik

- Statistik digest untuk Aurora MySQL
- Statistik per detik untuk Aurora MySQL
- Statistik per panggilan untuk Aurora MySQL

Statistik digest untuk Aurora MySQL

Wawasan Performa mengumpulkan statistik digest SQL dari tabel `events_statements_summary_by_digest`. Tabel `events_statements_summary_by_digest` dikelola oleh basis data Anda.

Tabel digest tidak memiliki kebijakan pengosongan. Jika tabel penuh, AWS Management Console menunjukkan pesan berikut:

```
Performance Insights is unable to collect SQL Digest statistics on new queries because
 the table events_statements_summary_by_digest is full.
Please truncate events_statements_summary_by_digest table to clear the issue. Check the
 User Guide for more details.
```

Dalam situasi ini, Aurora MySQL tidak melacak kueri SQL. Untuk mengatasi masalah ini, Wawasan Performa secara otomatis memotong tabel digest jika kedua kondisi berikut terpenuhi:

- Tabel penuh.
- Wawasan Performa mengelola Skema Performa secara otomatis.

  Untuk manajemen otomatis, parameter `performance_schema` harus diatur ke `0` dan Sumber tidak boleh diatur ke `user`. Jika Wawasan Performa tidak mengelola Skema Performa secara otomatis, lihat Mengaktifkan Skema Performa untuk Wawasan Performa di Aurora MySQL.

Di AWS CLI, periksa sumber nilai parameter dengan menjalankan perintah describe-db-parameters.

Statistik per detik untuk Aurora MySQL

Statistik SQL berikut ini tersedia untuk instans Aurora MySQL.

Metrik	Unit
db.sql_tokenized.stats.count_star_per_sec	Panggilan per detik
db.sql_tokenized.stats.sum_timer_wait_per_sec	Eksekusi aktif rata-rata per detik (AAE)
db.sql_tokenized.stats.sum_select_full_join_per_sec	Memilih penggabungan penuh per detik

Metrik	Unit
db.sql_tokenized.stats.sum_select_range_check_per_sec	Memilih pemeriksaan rentang per detik
db.sql_tokenized.stats.sum_select_scan_per_sec	Memilih pemindaian per detik
db.sql_tokenized.stats.sum_sort_merge_passes_per_sec	Mengurutkan pass penggabungan per detik
db.sql_tokenized.stats.sum_sort_scan_per_sec	Mengurutkan pemindaian per detik
db.sql_tokenized.stats.sum_sort_range_per_sec	Mengurutkan rentang per detik
db.sql_tokenized.stats.sum_sort_rows_per_sec	Mengurutkan baris per detik
db.sql_tokenized.stats.sum_rows_affected_per_sec	Baris yang terpengaruh per detik
db.sql_tokenized.stats.sum_rows_examined_per_sec	Baris yang diperiksa per detik
db.sql_tokenized.stats.sum_rows_sent_per_sec	Baris yang dikirim per detik
db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_sec	Tabel disk sementara yang dibuat per detik
db.sql_tokenized.stats.sum_created_tmp_tables_per_sec	Tabel disk sementara yang dibuat per detik
db.sql_tokenized.stats.sum_lock_time_per_sec	Waktu pencungian per detik (dalam md)

Statistik per panggilan untuk Aurora MySQL

Metrik berikut menyediakan statistik per panggilan untuk pernyataan SQL.

Metrik	Unit
db.sql_tokenized.stats.sum_timer_wait_per_call	Latensi rata-rata per panggilan (dalam md)
db.sql_tokenized.stats.sum_select_full_join_per_call	Memilih penggabungan penuh per panggilan
db.sql_tokenized.stats.sum_select_range_check_per_call	Memilih pemeriksaan rentang per panggilan
db.sql_tokenized.stats.sum_select_scan_per_call	Memilih pemindaian per panggilan
db.sql_tokenized.stats.sum_sort_merge_passes_per_call	Mengurutkan pass penggabungan per panggilan
db.sql_tokenized.stats.sum_sort_scan_per_call	Mengurutkan pemindaian per panggilan
db.sql_tokenized.stats.sum_sort_range_per_call	Mengurutkan rentang per panggilan
db.sql_tokenized.stats.sum_sort_rows_per_call	Mengurutkan baris per panggilan
db.sql_tokenized.stats.sum_rows_affected_per_call	Baris yang terpengaruh per panggilan
db.sql_tokenized.stats.sum_rows_examined_per_call	Baris yang diperiksa per panggilan
db.sql_tokenized.stats.sum_rows_sent_per_call	Baris yang terkirim per panggilan
db.sql_tokenized.stats.sum_created_tmp_disk_tables_per_call	Tabel disk sementara yang dibuat per panggilan
db.sql_tokenized.stats.sum_created_tmp_tables_per_call	Tabel sementara yang dibuat per panggilan
db.sql_tokenized.stats.sum_lock_time_per_call	Waktu penguncian per panggilan (dalam md)

Statistik SQL untuk Aurora PostgreSQL

Untuk setiap panggilan SQL dan untuk setiap detik eksekusi kueri, Wawasan Performa mengumpulkan statistik SQL. Semua mesin Aurora mengumpulkan statistik hanya pada tingkat digest.

Berikut ini, Anda dapat menemukan informasi tentang statistik tingkat digest untuk Aurora PostgreSQL.

Topik

- Statistik digest untuk Aurora PostgreSQL
- Statistik digest per detik untuk Aurora PostgreSQL
- Statistik digest per panggilan untuk AuroraPostgreSQL

Statistik digest untuk Aurora PostgreSQL

Untuk menampilkan statistik digest SQL, pustaka `pg_stat_statements` harus dimuat. Untuk klaster DB Aurora PostgreSQL yang kompatibel dengan PostgreSQL 10, pustaka ini dimuat secara default. Untuk klaster DB Aurora PostgreSQL yang kompatibel dengan PostgreSQL 9.6, aktifkan pustaka ini secara manual. Untuk mengaktifkannya secara manual, tambahkan `pg_stat_statements` ke `shared_preload_libraries` di grup parameter DB yang terkait dengan instans DB. Lalu reboot instans DB Anda. Untuk informasi selengkapnya, lihat Bekerja dengan grup parameter.

> **Note**
>
> Wawasan Performa hanya dapat mengumpulkan statistik untuk kueri dalam `pg_stat_activity` yang tidak terpotong. Secara default, basis data PostgreSQL memotong kueri yang lebih panjang dari 1.024 byte. Untuk menambah ukuran kueri, ubah parameter `track_activity_query_size` dalam grup parameter DB yang terkait dengan instans DB Anda. Jika Anda mengubah parameter ini, instans DB harus di-reboot.

Statistik digest per detik untuk Aurora PostgreSQL

Statistik digest SQL berikut tersedia untuk instans DB Aurora PostgreSQL.

Metrik	Unit
db.sql_tokenized.stats.calls_per_sec	Panggilan per detik
db.sql_tokenized.stats.rows_per_sec	Baris per detik
db.sql_tokenized.stats.total_time_per_sec	Eksekusi aktif rata-rata per detik (AAE)
db.sql_tokenized.stats.shared_blks_hit_per_sec	Hit blokir per detik
db.sql_tokenized.stats.shared_blks_read_per_sec	Pembacaan blokir per detik
db.sql_tokenized.stats.shared_blks_dirtied_per_sec	Blokir kotor per detik
db.sql_tokenized.stats.shared_blks_written_per_sec	Penulisan blokir per detik
db.sql_tokenized.stats.local_blks_hit_per_sec	Hit blokir lokal per detik
db.sql_tokenized.stats.local_blks_read_per_sec	Pembacaan blokir lokal per detik
db.sql_tokenized.stats.local_blks_dirtied_per_sec	Blokir kotor lokal per detik
db.sql_tokenized.stats.local_blks_written_per_sec	Penulisan blokir lokal per detik
db.sql_tokenized.stats.temp_blks_written_per_sec	Penulisan sementara per detik
db.sql_tokenized.stats.temp_blks_read_per_sec	Pembacaan sementara per detik
db.sql_tokenized.stats.blk_read_time_per_sec	Pembacaan serentak rata-rata per detik
db.sql_tokenized.stats.blk_write_time_per_sec	Penulisan serentak rata-rata per detik

Statistik digest per panggilan untuk AuroraPostgreSQL

Metrik berikut menyediakan statistik per panggilan untuk pernyataan SQL.

Metrik	Unit
db.sql_tokenized.stats.rows_per_call	Baris per panggilan
db.sql_tokenized.stats.avg_latency_per_call	Latensi rata-rata per panggilan (dalam md)
db.sql_tokenized.stats.shared_blks_hit_per_call	Hit blokir per panggilan
db.sql_tokenized.stats.shared_blks_read_per_call	Pembacaan blokir per panggilan
db.sql_tokenized.stats.shared_blks_written_per_call	Penulisan blokir per panggilan
db.sql_tokenized.stats.shared_blks_dirtied_per_call	Blokir kotor per panggilan
db.sql_tokenized.stats.local_blks_hit_per_call	Hit blokir lokal per panggilan
db.sql_tokenized.stats.local_blks_read_per_call	Pembcaan blokir lokal per panggilan
db.sql_tokenized.stats.local_blks_dirtied_per_call	Blokir kotor lokal per panggilan
db.sql_tokenized.stats.local_blks_written_per_call	Penulisan blokir lokal per panggilan
db.sql_tokenized.stats.temp_blks_written_per_call	Penulisan blokir sementara per panggilan
db.sql_tokenized.stats.temp_blks_read_per_call	Pembacaan blokir sementara per panggilan
db.sql_tokenized.stats.blk_read_time_per_call	Waktu baca per panggilan (dalam md)
db.sql_tokenized.stats.blk_write_time_per_call	Waktu tulis per panggilan (dalam md)

Untuk informasi selengkapnya tentang metrik ini, lihat pg_stat_statements dalam dokumentasi PostgreSQL.

Metrik OS dalam Pemantauan yang Disempurnakan

Amazon Aurora menyediakan metrik secara real-time untuk sistem operasi (OS) tempat klaster DB Anda berjalan. Aurora memberikan metrik dari Enhanced Monitoring ke akun Amazon Logs Anda. CloudWatch Tabel berikut mencantumkan metrik OS yang tersedia menggunakan Amazon CloudWatch Logs.

Topik

- Metrik OS untuk Aurora

Metrik OS untuk Aurora

Grup	Metrik	Nama konsol	Deskripsi
General	engine	Tidak berlaku	Mesin basis data untuk instans DB.
	instanceID	Tidak berlaku	Pengidentifikasi instans DB.
	instanceResourceID	Tidak berlaku	Pengidentifikasi tetap untuk instans DB yang unik untuk Wilayah AWS, juga digunakan sebagai pengidentifikasi log stream.
	numVCPUs	Tidak berlaku	Jumlah CPU virtual untuk instans DB.
	timestamp	Tidak berlaku	Waktu pengambilan metrik.
	uptime	Tidak berlaku	Jumlah waktu saat instans DB telah aktif.

Grup	Metrik	Nama konsol	Deskripsi
	`version`	Tidak berlaku	Versi format JSON aliran metrik OS.
`cpuUtilization`	`guest`	CPU Tamu	Persentase CPU yang digunakan oleh program tamu.
	`idle`	CPU Idle	Persentase CPU saat idle.
	`irq`	CPU IRQ	Persentase CPU yang digunakan oleh gangguan perangkat lunak.
	`nice`	CPU Nice	Persentase CPU yang digunakan oleh program yang berjalan pada prioritas terendah.
	`steal`	CPU Steal	Persentase CPU yang digunakan oleh mesin virtual lainnya.
	`system`	CPU Sistem	Persentase CPU yang digunakan oleh kernel.
	`total`	CPU Total	Total persentase CPU yang digunakan. Nilai ini mencakup nilai `nice`.
	`user`	Pengguna CPU	Persentase CPU yang digunakan oleh program pengguna.
	`wait`	CPU Tunggu	Persentase CPU yang tidak digunakan saat menunggu akses I/O.
`diskIO`	`avgQueueLen`	Ukuran Antrean Rata-rata	Jumlah permintaan yang menunggu dalam antrean perangkat I/O.
	`avgReqSz`	Ukuran Permintaan Rata-rata	Ukuran permintaan rata-rata, dalam kilobyte.

Grup	Metrik	Nama konsol	Deskripsi
	`await`	I/O Disk Tunggu	Jumlah milidetik yang diperlukan untuk merespons permintaan, termasuk waktu antrean dan waktu layanan.
	`device`	Tidak berlaku	Pengidentifikasi perangkat disk yang digunakan.
	`readIOsPS`	IO/d Baca	Jumlah operasi baca per detik.
	`readKb`	Total Baca	Jumlah total kilobyte yang dibaca.
	`readKbPS`	Kb/d Baca	Jumlah kilobyte yang dibaca per detik.
	`readLatency`	Latensi Baca	Waktu yang berlalu antara pengiriman permintaan I/O baca dan penyelesaiannya, dalam milidetik. Metrik ini hanya tersedia untuk Amazon Aurora.
	`readThroughput`	Throughput Baca	Jumlah throughput jaringan yang digunakan oleh permintaan ke klaster DB, dalam byte per detik. Metrik ini hanya tersedia untuk Amazon Aurora.
	`rrqmPS`	Rrqms	Jumlah permintaan baca gabungan yang diantrekan per detik.
	`tps`	TPS	Jumlah transaksi I/O per detik.
	`util`	Penggunaan I/O Disk	Persentase waktu CPU saat permintaan dikeluarkan.
	`writeIOsPS`	IO/d Tulis	Jumlah operasi tulis per detik.
	`writeKb`	Total Tulis	Jumlah total kilobyte yang ditulis.

Grup	Metrik	Nama konsol	Deskripsi
	writeKbPS	Kb/d Tulis	Jumlah kilobyte yang ditulis per detik.
	writeLatency	Latensi Tulis	Waktu yang berlalu antara pengiriman permintaan I/O tulis dan penyelesaiannya, dalam milidetik. Metrik ini hanya tersedia untuk Amazon Aurora.
	writeThroughput	Throughput Tulis	Jumlah throughput jaringan yang digunakan oleh respons dari klaster DB, dalam byte per detik. Metrik ini hanya tersedia untuk Amazon Aurora.
	wrqmPS	Wrqms	Jumlah permintaan tulis gabungan yang diantrekan per detik.
fileSys	maxFiles	Inode Maks	Jumlah maksimum file yang dapat dibuat untuk sistem file.
	mountPoint	Tidak berlaku	Jalur ke sistem file.
	name	Tidak berlaku	Nama sistem file.
	total	Total Sistem File	Jumlah total ruang disk yang tersedia untuk sistem file, dalam kilobyte.
	used	Sistem file yang digunakan	Jumlah ruang disk yang digunakan oleh file dalam sistem file, dalam kilobyte.
	usedFilePercent	Inode yang Digunakan	Persentase file tersedia yang digunakan.
	usedFiles	% yang Digunakan	Jumlah file dalam sistem file.

Grup	Metrik	Nama konsol	Deskripsi
	`usedPercent`	Sistem file yang digunakan	Persentase ruang disk sistem file yang digunakan.
`loadAverageMinute`	`fifteen`	Rata-rata Muatan 15 menit	Jumlah proses yang meminta waktu CPU selama 15 menit terakhir.
	`five`	Rata-rata Muatan 5 menit	Jumlah proses yang meminta waktu CPU selama 5 menit terakhir.
	`one`	Rata-rata Muatan 1 menit	Jumlah proses yang meminta waktu CPU selama satu menit terakhir.
`memory`	`active`	Memori Aktif	Jumlah memori yang ditetapkan, dalam kilobyte.
	`buffers`	Memori yang Di-buffer	Jumlah memori yang digunakan untuk buffering permintaan I/O sebelum menulis ke perangkat penyimpanan, dalam kilobyte.
	`cached`	Memori yang Di-cache	Jumlah memori yang digunakan untuk meng-cache file I/O berbasis sistem file.
	`dirty`	Memori Kotor	Jumlah halaman memori dalam RAM yang telah diubah, tetapi tidak ditulis ke blok data terkaitnya dalam penyimpanan, dalam kilobyte.
	`free`	Memori Kosong	Jumlah memori yang tidak ditetapkan, dalam kilobyte.

Grup	Metrik	Nama konsol	Deskripsi
	`hugePagesFree`	Halaman Besar Kosong	Jumlah halaman besar yang kosong. Halaman besar adalah fitur dari kernel Linux.
	`hugePagesRsvd`	Rsvd Halaman Besar	Jumlah halaman besar yang khusus.
	`hugePagesSize`	Ukuran Halaman Besar	Ukuran untuk setiap unit halaman besar, dalam kilobyte.
	`hugePagesSurp`	Surplus Halaman Besar	Jumlah halaman besar surplus yang tersedia terhadap total.
	`hugePagesTotal`	Total Halaman Besar	Jumlah total halaman besar.
	`inactive`	Memori Tidak Aktif	Jumlah halaman memori yang jarang digunakan, dalam kilobyte.
	`mapped`	Memori yang Dipetakan	Jumlah total konten sistem file yang memorinya dipetakan di dalam ruang alamat proses, dalam kilobyte.
	`pageTables`	Tabel Halaman	Jumlah memori yang digunakan berdasarkan tabel halaman, dalam kilobyte.
	`slab`	Memori Slab	Jumlah struktur data kernel yang dapat digunakan kembali, dalam kilobyte.
	`total`	Memori Total	Jumlah total memori, dalam kilobyte.

Grup	Metrik	Nama konsol	Deskripsi
	writeback	Memori Writeback	Jumlah halaman kotor dalam RAM yang masih ditulis ke penyimpanan cadangan, dalam kilobyte.
network	interface	Tidak berlaku	Pengidentifikasi untuk antarmuka jaringan yang digunakan untuk instans DB.
	rx	RX	Jumlah byte yang diterima per detik.
	tx	TX	Jumlah byte yang diunggah per detik.
processList	cpuUsedPc	CPU %	Persentase CPU yang digunakan oleh proses.
	id	Tidak berlaku	Pengidentifikasi proses.
	memoryUsedPc	MEM%	Persentase memori yang digunakan oleh proses.
	name	Tidak berlaku	Nama proses.
	parentID	Tidak berlaku	ID proses untuk proses induk dari proses.
	rss	RES	Jumlah RAM yang dialokasikan untuk proses, dalam kilobyte.
	tgid	Tidak berlaku	ID grup utas, yaitu angka yang mewakili ID proses di tempat utas berada. ID ini digunakan untuk mengelompokkan utas dari proses yang sama.
	vss	VIRT	Jumlah memori virtual yang dialokasikan untuk proses, dalam kilobyte.
swap	swap	Swap	Jumlah memori swap yang tersedia, dalam kilobyte.

Grup	Metrik	Nama konsol	Deskripsi
	`swap in`	Swap dalam	Jumlah memori, dalam kilobyte, yang ditukar ke dalam dari disk.
	`swap out`	Swap luar	Jumlah memori, dalam kilobyte, yang ditukar ke luar dari disk.
	`free`	Swap Kosong	Jumlah memori swap yang kosong, dalam kilobyte.
	`committed`	Swap Komit	Jumlah memori swap, dalam kilobyte, yang digunakan sebagai memori cache.
`tasks`	`blocked`	Tugas Diblokir	Jumlah tugas yang diblokir.
	`running`	Tugas Berjalan	Jumlah tugas yang berjalan.
	`sleeping`	Tugas Tidur	Jumlah tugas yang tidur.
	`stopped`	Tugas Dihentikan	Jumlah tugas yang dihentikan.
	`total`	Total Tugas	Jumlah total tugas.
	`zombie`	Tugas Zombie	Jumlah tugas turunan yang tidak aktif dengan tugas induk aktif.

Memantau peristiwa, log, dan aliran di klaster DB Amazon Aurora

Saat Anda memantau basis data dan solusi Anda yang AWS lain, tujuan Anda adalah mempertahankan hal-hal berikut:

- Keandalan
- Ketersediaan
- Performa
- Keamanan

Memantau metrik di klaster Amazon Aurora menjelaskan cara memantau klaster menggunakan metrik. Solusi lengkap juga harus memantau peristiwa database, file log, dan aliran aktivitas. AWS menyediakan Anda dengan alat pemantauan berikut:

- Amazon EventBridge adalah layanan bus acara tanpa server yang memudahkan untuk menghubungkan aplikasi Anda dengan data dari berbagai sumber. EventBridge memberikan aliran data real-time dari aplikasi Anda sendiri, aplikasi S oftware-as-a -Service (SaaS), dan layanan. AWS EventBridge merutekan data tersebut ke target seperti AWS Lambda. Dengan demikian, Anda dapat memantau peristiwa yang terjadi di layanan dan membangun arsitektur berbasis peristiwa. Untuk informasi selengkapnya, lihat Panduan EventBridge Pengguna Amazon.
- Amazon CloudWatch Logs menyediakan cara untuk memantau, menyimpan, dan mengakses file log Anda dari instans AWS CloudTrail,, dan sumber lainnya. Amazon CloudWatch Logs dapat memantau informasi dalam file log dan memberi tahu Anda ketika ambang batas tertentu terpenuhi. Anda juga dapat mengarsipkan data log dalam penyimpanan yang sangat tahan lama. Untuk informasi selengkapnya, lihat Panduan Pengguna Amazon CloudWatch Logs.
- AWS CloudTrailmenangkap panggilan API dan peristiwa terkait yang dibuat oleh atau atas nama Anda Akun AWS. CloudTrail mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Anda dapat mengidentifikasi pengguna dan akun mana yang dipanggil AWS, alamat IP sumber dari mana panggilan dilakukan, dan kapan panggilan terjadi. Untuk informasi selengkapnya, silakan lihat Panduan Pengguna AWS CloudTrail.
- Aliran Aktivitas Basis Data adalah fitur Amazon Aurora yang menyediakan aliran aktivitas mendekati waktu nyata di klaster DB Anda. Amazon Aurora mendorong aktivitas ke aliran data Amazon Kinesis. Aliran Kinesis dibuat secara otomatis. Dari Kinesis, Anda dapat mengonfigurasi

AWS layanan seperti Amazon Data Firehose dan AWS Lambda menggunakan aliran dan menyimpan data.

Topik

- Melihat log, peristiwa, dan stream di konsol Amazon RDS
- Memantau peristiwa Amazon Aurora
- Memantau file log Amazon Aurora
- Memantau panggilan API Amazon Aurora di AWS CloudTrail
- Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data

Melihat log, peristiwa, dan stream di konsol Amazon RDS

Amazon RDS terintegrasi dengan Layanan AWS untuk menampilkan informasi tentang log, peristiwa, dan stream aktivitas basis data di konsol RDS.

Tab Log & peristiwa untuk klaster DB Aurora Anda menampilkan informasi berikut:

- Kebijakan dan aktivitas penskalaan otomatis – Menampilkan kebijakan dan aktivitas yang berkaitan dengan fitur Auto Scaling Aurora. Informasi ini hanya muncul di tab Log & peristiwa pada tingkat klaster.
- Alarm Amazon CloudWatch – Menampilkan semua alarm metrik yang telah Anda konfigurasikan untuk instans DB di klaster Aurora Anda. Jika Anda belum mengonfigurasikan alarm, Anda dapat membuatnya di konsol RDS.
- Peristiwa terbaru – Menampilkan ringkasan peristiwa (perubahan lingkungan) untuk instans atau klaster DB Aurora Anda. Untuk informasi selengkapnya, lihat Melihat peristiwa Amazon RDS.
- Log – Menampilkan file log basis data yang dihasilkan oleh instans DB di klaster Aurora Anda. Untuk informasi selengkapnya, lihat Memantau file log Amazon Aurora.

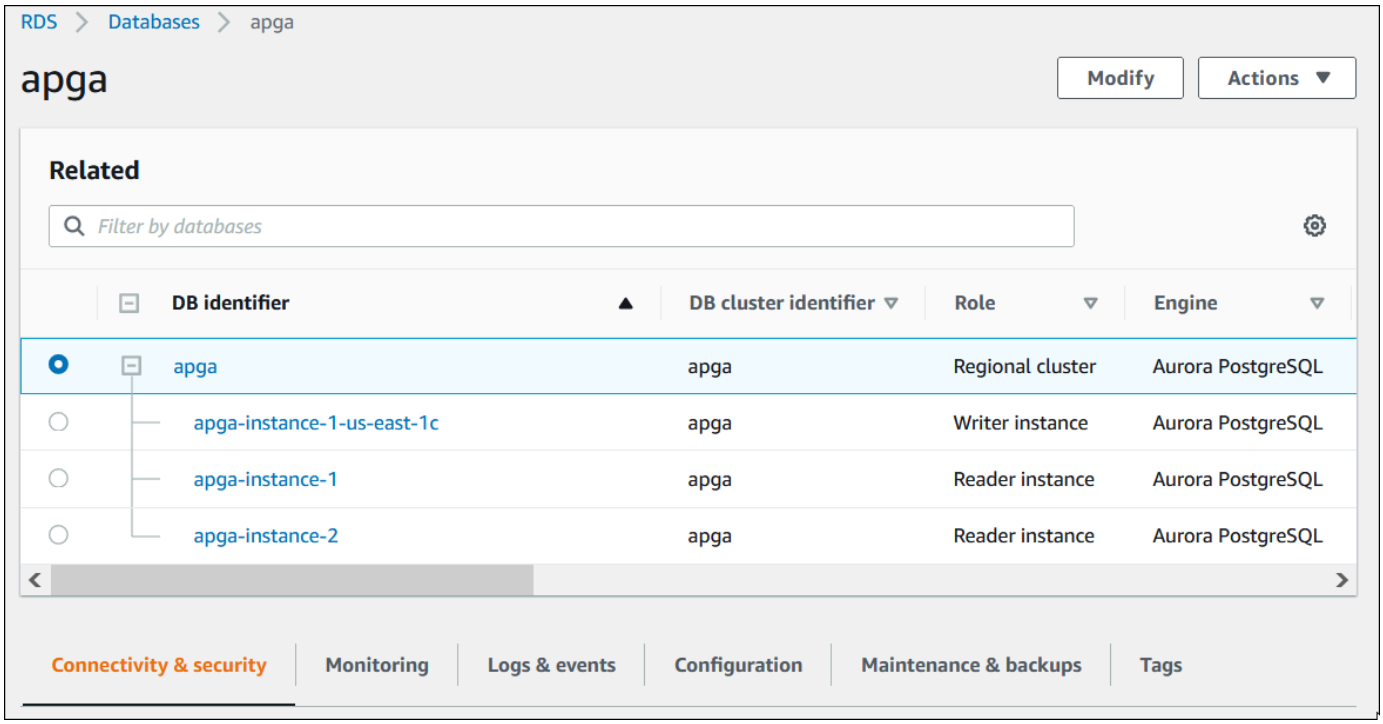
Tab Konfigurasi akan menayangkan informasi tentang stream aktivitas basis data.

Untuk melihat log, peristiwa, dan stream bagi klaster DB Aurora Anda di konsol RDS

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.

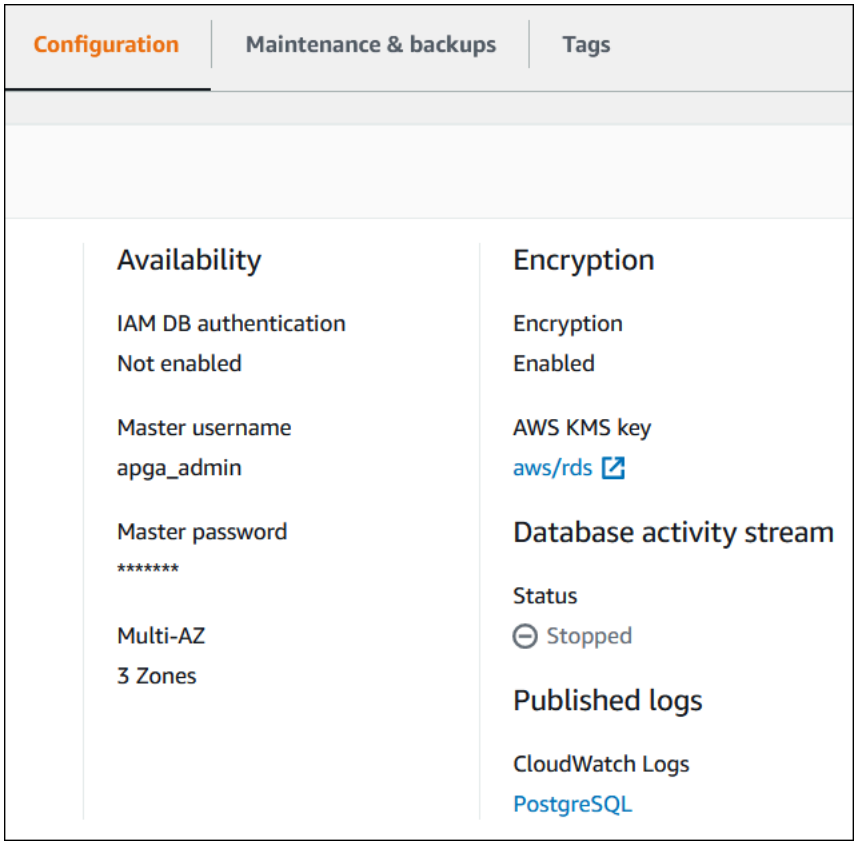
3. Pilih nama klaster DB Aurora yang ingin dipantau.

   Halaman basis data akan muncul. Contoh berikut menunjukkan klaster DB Amazon Aurora PostgreSQL bernama apga.



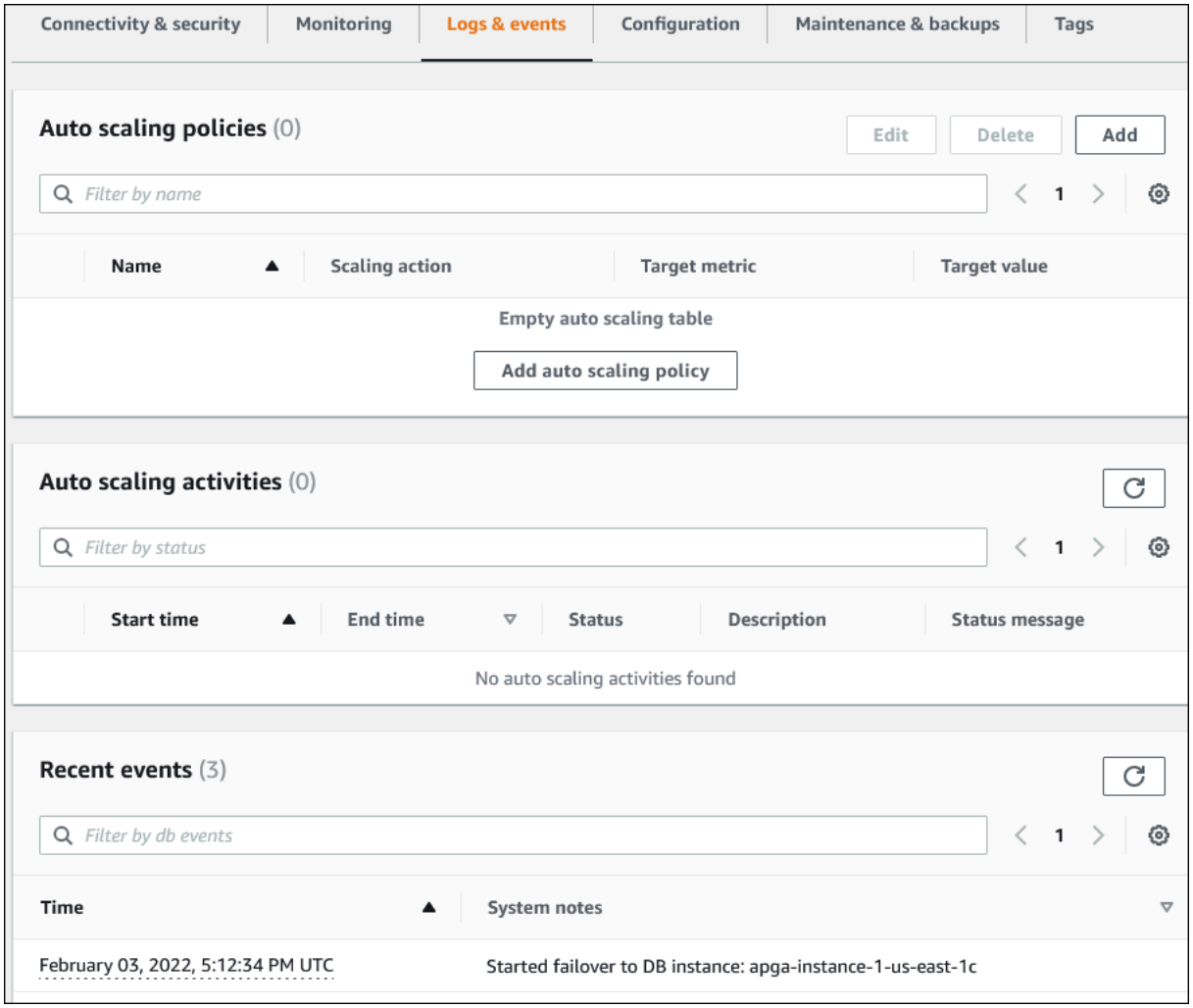
4. Gulir turun dan pilih Konfigurasi.

   Contoh berikut menunjukkan status stream aktivitas basis data untuk klaster Anda.



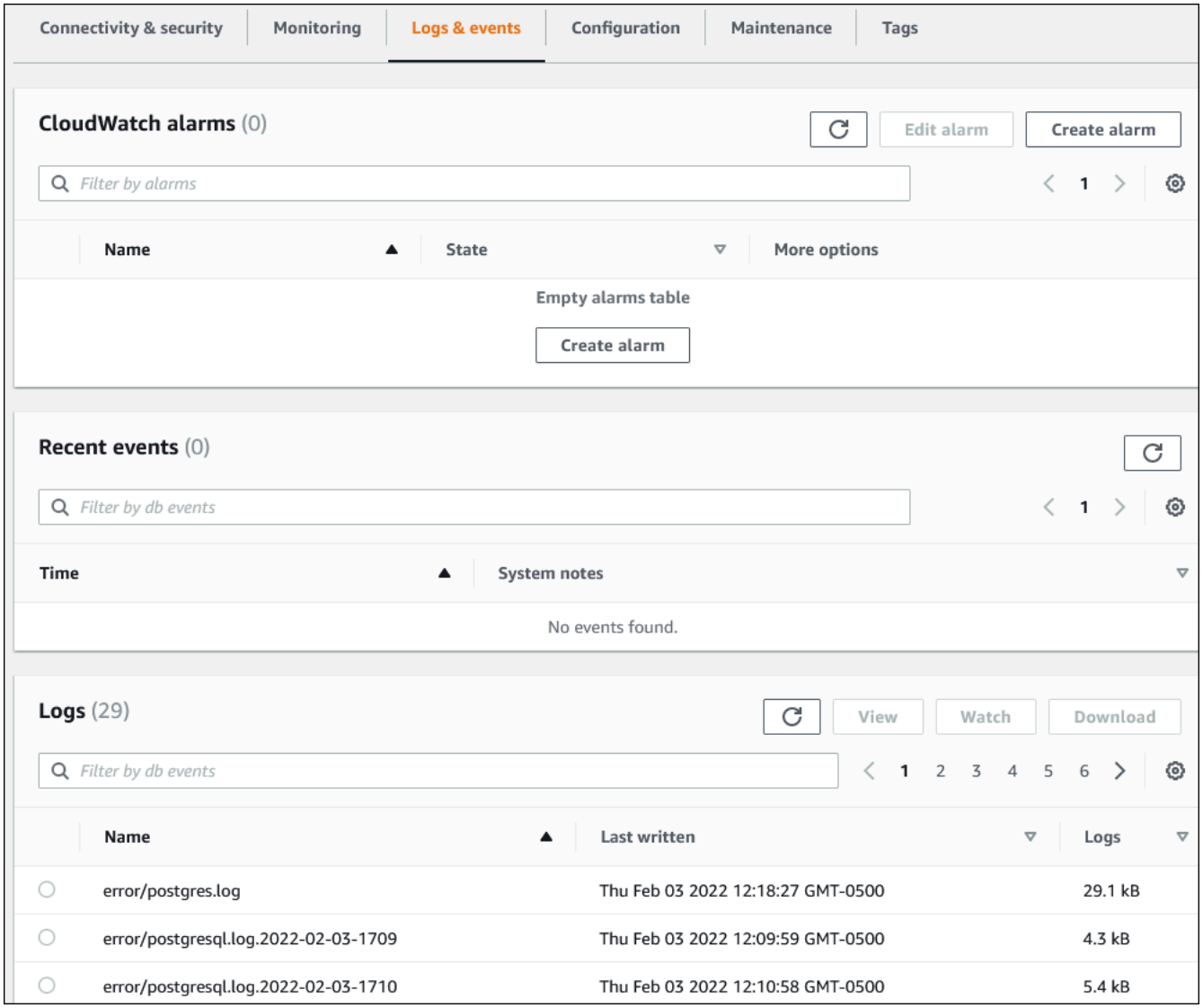
5. Pilih Log & peristiwa.

   Bagian Log & peristiwa akan muncul.



6. Pilih instans DB di klaster Aurora Anda, lalu pilih Log & peristiwa untuk instans itu.

   Contoh berikut menunjukkan perbedaan isi antara halaman instans DB dan halaman klaster DB. Halaman instans DB menampilkan log dan alarm.


Connectivity & security
Monitoring
Logs & events
Configuration
Maintenance
Tags
CloudWatch alarms (0)
Edit alarm
Create alarm
Filter by alarms
1
Name
State
More options
Empty alarms table
Create alarm
Recent events (0)
Filter by db events
1
Time
System notes
No events found.
Logs (29)
View
Watch
Download
Filter by db events
1 2 3 4 5 6
Name
Last written
Logs
error/postgres.log
Thu Feb 03 2022 12:18:27 GMT-0500
29.1 kB
error/postgresql.log.2022-02-03-1709
Thu Feb 03 2022 12:09:59 GMT-0500
4.3 kB
error/postgresql.log.2022-02-03-1710
Thu Feb 03 2022 12:10:58 GMT-0500
5.4 kB

Memantau peristiwa Amazon Aurora

Sesi kejadian menunjukkan perubahan dalam lingkungan. Hal ini dapat berupa lingkungan AWS, layanan atau aplikasi mitra SaaS, atau aplikasi atau layanan kustom. Untuk mengetahui deskripsi peristiwa Aurora, lihat Kategori peristiwa dan pesan peristiwa Amazon RDS.

Topik

- Ikhtisar peristiwa untuk Aurora
- Melihat peristiwa Amazon RDS
- Bekerja dengan pemberitahuan peristiwa Amazon RDS
- Membuat aturan yang dipicu pada peristiwa Amazon Aurora
- Kategori peristiwa dan pesan peristiwa Amazon RDS

Ikhtisar peristiwa untuk Aurora

Peristiwa RDS menunjukkan perubahan di lingkungan Aurora. Misalnya, Amazon Aurora menghasilkan peristiwa saat klaster DB di-patching. Amazon Aurora mengirimkan peristiwa ke CloudWatch Events dan EventBridge hampir secara real-time.

> **Note**
>
> Amazon RDS menyiarkan peristiwa semaksimal mungkin. Sebaiknya Anda menghindari menulis program yang mengandalkan urutan atau keberadaan peristiwa pemberitahuan, karena program tersebut mungkin tidak berurutan atau hilang.

Amazon RDS mencatat peristiwa yang berhubungan dengan sumber daya berikut:

- Klaster DB

  Untuk daftar peristiwa klaster, lihat Peristiwa klaster DB.
- Instans DB

  Untuk daftar peristiwa instans DB, lihat Peristiwa instans DB.
- Grup parameter DB

  Untuk daftar peristiwa grup parameter DB, lihat Peristiwa grup parameter DB.

- Grup keamanan DB

  Untuk daftar peristiwa grup keamanan DB, lihat Peristiwa grup keamanan DB.
- Snapshot klaster DB

  Untuk daftar peristiwa snapshot klaster DB, lihat Peristiwa snapshot klaster DB.
- Peristiwa Proksi RDS

  Untuk daftar peristiwa Proksi RDS, lihat Peristiwa Proksi RDS.
- Peristiwa deployment blue/green

  Untuk daftar peristiwa deployment blue/green, lihat Peristiwa deployment blue/green.

Informasi ini mencakup hal berikut:

- Tanggal dan waktu peristiwa
- Nama sumber dan jenis sumber peristiwa
- Pesan yang terkait dengan peristiwa
- Pemberitahuan peristiwa mencakup tag sejak pesan dikirim dan mungkin tidak mencerminkan tag pada saat peristiwa terjadi

Melihat peristiwa Amazon RDS

Anda dapat mengambil informasi peristiwa berikut untuk sumber daya Amazon Aurora:

- Nama sumber daya
- Jenis sumber daya
- Waktu peristiwa
- Ringkasan pesan peristiwa

Akses peristiwa melalui AWS Management Console, yang menampilkan peristiwa selama 24 jam terakhir. Anda juga dapat mengambil peristiwa dengan menggunakan perintah AWS CLI describe-events, atau operasi DescribeEvents RDS API. Jika menggunakan AWS CLI atau RDS API untuk melihat peristiwa, Anda dapat mengambil peristiwa hingga 14 hari terakhir.

> **Note**
>
> Jika perlu menyimpan peristiwa dalam jangka waktu yang lebih lama, Anda dapat mengirimkan peristiwa Amazon RDS ke CloudWatch Events. Untuk informasi selengkapnya, lihat Membuat aturan yang dipicu pada peristiwa Amazon Aurora

Untuk deskripsi peristiwa Amazon Aurora, lihat Kategori peristiwa dan pesan peristiwa Amazon RDS.

Untuk mengakses informasi mendetail tentang peristiwa yang menggunakan AWS CloudTrail, termasuk parameter permintaan, lihat Peristiwa CloudTrail.

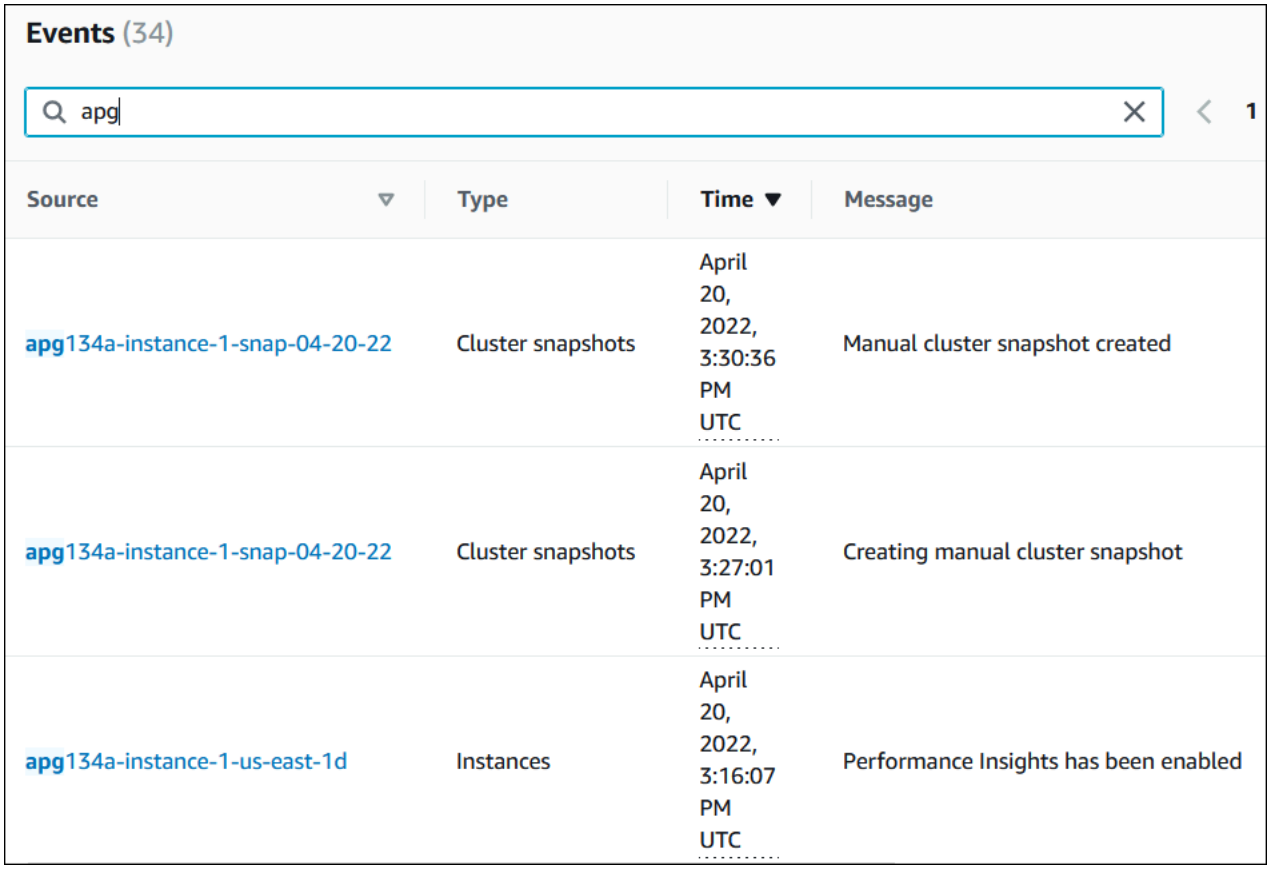
Konsol

Untuk melihat semua peristiwa Amazon RDS selama 24 jam terakhir

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Peristiwa.

   Peristiwa yang tersedia akan muncul dalam daftar.
3. (Opsional) Masukkan istilah pencarian untuk memfilter hasil.

   Contoh berikut menunjukkan daftar peristiwa yang difilter oleh karakter **apg**.



AWS CLI

Untuk melihat semua peristiwa yang dihasilkan dalam satu jam terakhir, panggil describe-events tanpa parameter.

```
aws rds describe-events
```

Output sampel berikut menunjukkan bahwa instans klaster DB telah dihentikan.

```
{
    "Events": [
        {
            "EventCategories": [
                "recovery"
            ],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mycluster-instance-1",
            "Date": "2022-04-20T15:02:38.416Z",
```

```
            "Message": "Recovery of the DB instance has started. Recovery time will
 vary with the amount of data to be recovered.",
            "SourceIdentifier": "mycluster-instance-1"
        }, ...
```

Untuk melihat semua peristiwa Amazon RDS selama 10.080 menit terakhir (7 hari), panggil perintah describe-events AWS CLI dan atur parameter `--duration` ke `10080`.

```
aws rds describe-events --duration 10080
```

Contoh berikut menunjukkan peristiwa dalam rentang waktu yang ditentukan untuk instans DB *test-instance*.

```
aws rds describe-events \
    --source-identifier test-instance \
    --source-type db-instance \
    --start-time 2022-03-13T22:00Z \
    --end-time 2022-03-13T23:59Z
```

Output sampel berikut menampilkan status cadangan.

```
{
    "Events": [
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Backing up DB instance",
            "Date": "2022-03-13T23:09:23.983Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
        },
        {
            "SourceType": "db-instance",
            "SourceIdentifier": "test-instance",
            "EventCategories": [
                "backup"
            ],
            "Message": "Finished DB Instance backup",
            "Date": "2022-03-13T23:15:13.049Z",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance"
```

```
        }
    ]
}
```

API

Anda dapat melihat semua peristiwa instans Amazon RDS selama 14 hari terakhir dengan memanggil operasi DescribeEvents RDS API dan mengatur parameter `Duration` ke `20160`.

Bekerja dengan pemberitahuan peristiwa Amazon RDS

Amazon RDS menggunakan Amazon Simple Notification Service (Amazon SNS) untuk memberikan pemberitahuan saat peristiwa Amazon RDS terjadi. Pemberitahuan ini bisa dalam bentuk pemberitahuan apa pun yang didukung oleh Amazon SNS untuk Wilayah AWS, seperti email, pesan teks, atau panggilan ke titik akhir HTTP.

Topik

- Ikhtisar pemberitahuan peristiwa Amazon RDS
- Memberikan izin untuk menerbitkan pemberitahuan ke topik Amazon SNS
- Berlangganan pemberitahuan peristiwa Amazon RDS
- Tag dan atribut pemberitahuan peristiwa Amazon RDS
- Mencantumkan langganan pemberitahuan peristiwa Amazon RDS
- Mengubah langganan pemberitahuan peristiwa Amazon RDS
- Menambahkan pengidentifikasi sumber ke langganan pemberitahuan peristiwa Amazon RDS
- Menghapus pengidentifikasi sumber dari langganan pemberitahuan peristiwa Amazon RDS
- Mencantumkan kategori pemberitahuan peristiwa Amazon RDS
- Menghapus langganan pemberitahuan peristiwa Amazon RDS

Ikhtisar pemberitahuan peristiwa Amazon RDS

Amazon RDS mengelompokkan peristiwa ke dalam beberapa kategori langganan sehingga Anda dapat menerima pemberitahuan saat suatu peristiwa dalam kategori tersebut terjadi.

Topik

- Sumber daya RDS memenuhi syarat untuk langganan peristiwa
- Proses dasar untuk berlangganan pemberitahuan peristiwa Amazon RDS
- Pengiriman pemberitahuan peristiwa RDS
- Penagihan untuk pemberitahuan peristiwa Amazon RDS
- Contoh peristiwa Aurora

Sumber daya RDS memenuhi syarat untuk langganan peristiwa

Untuk Amazon Aurora, peristiwa terjadi baik di klaster DB maupun tingkat instans DB. Anda dapat berlangganan kategori peristiwa untuk sumber daya berikut:

- Instans DB
- Klaster DB
- Snapshot klaster DB
- Grup parameter DB
- Grup keamanan DB
- Proksi RDS
- Versi mesin kustom

Misalnya, jika berlangganan kategori pencadangan untuk instans DB tertentu, Anda akan diberi tahu setiap kali ada peristiwa terkait pencadangan yang memengaruhi instans DB. Jika berlangganan kategori perubahan konfigurasi untuk instans DB, Anda akan diberi tahu saat instans DB diubah. Anda juga menerima pemberitahuan saat langganan pemberitahuan peristiwa berubah.

Sebaiknya Anda membuat beberapa langganan yang berbeda. Misalnya, Anda dapat membuat satu langganan yang menerima semua pemberitahuan peristiwa untuk semua instans DB dan langganan lain yang hanya mencakup peristiwa penting untuk sebagian instans DB. Untuk langganan kedua, tentukan satu atau beberapa instans DB dalam filter.

Proses dasar untuk berlangganan pemberitahuan peristiwa Amazon RDS

Proses untuk berlangganan pemberitahuan peristiwa Amazon RDS adalah sebagai berikut:

1. Anda membuat langganan pemberitahuan peristiwa Amazon RDS dengan menggunakan konsol Amazon RDS, AWS CLI, atau API.

   Amazon RDS menggunakan ARN topik Amazon SNS untuk mengidentifikasi setiap langganan. Konsol Amazon RDS membuat ARN untuk Anda saat Anda membuat langganan. Buat ARN dengan menggunakan konsol Amazon SNS, AWS CLI, atau Amazon SNS API.
2. Amazon RDS mengirimkan email persetujuan atau pesan SMS ke alamat yang Anda kirim bersama langganan Anda.
3. Anda mengonfirmasi langganan Anda dengan memilih tautan di pemberitahuan yang Anda terima.

4. Konsol Amazon RDS memperbarui bagian Langganan Peristiwa Saya dengan status langganan Anda.
5. Amazon RDS mulai mengirim pemberitahuan ke alamat yang Anda berikan saat membuat langganan.

Untuk mempelajari manajemen identitas dan akses saat menggunakan Amazon SNS, lihat Manajemen identitas dan akses di Amazon SNS di Panduan Developer Amazon Simple Notification Service.

Anda dapat menggunakan AWS Lambda untuk memproses pemberitahuan peristiwa dari instans DB. Untuk informasi lebih lanjut, lihat Menggunakan AWS Lambda dengan Amazon RDS di Panduan Developer AWS Lambda.

Pengiriman pemberitahuan peristiwa RDS

Amazon RDS mengirimkan pemberitahuan ke alamat yang Anda berikan saat membuat langganan. Pemberitahuan dapat mencakup atribut pesan yang berisi metadata terstruktur tentang pesan tersebut. Untuk informasi selengkapnya tentang atribut pesan, lihat Kategori peristiwa dan pesan peristiwa Amazon RDS.

Pemberitahuan peristiwa mungkin memerlukan waktu hingga lima menit untuk dikirimkan.

> **Important**
>
> Amazon RDS tidak menjamin urutan peristiwa yang dikirim dalam aliran peristiwa. Urutan peristiwa dapat berubah.

Ketika Amazon SNS mengirimkan pemberitahuan ke titik akhir HTTP atau HTTPS langganan, pesan POST yang dikirim ke titik akhir memiliki isi pesan yang berisi dokumen JSON. Untuk informasi selengkapnya, lihat Pesan Amazon SNS dan format JSON di Panduan Developer Amazon Simple Notification Service.

Anda dapat mengonfigurasi SNS untuk memberi tahu Anda dengan pesan teks. Untuk informasi selengkapnya, lihat Pesan teks seluler (SMS) dalam Panduan Developer Amazon Simple Notification Service.

Untuk menonaktifkan pemberitahuan tanpa menghapus langganan, pilih Tidak untuk Aktif di konsol Amazon RDS. Atau Anda dapat mengatur parameter `Enabled` ke `false` menggunakan AWS CLI atau Amazon RDS API.

Penagihan untuk pemberitahuan peristiwa Amazon RDS

Penagihan untuk pemberitahuan peristiwa Amazon RDS melalui Amazon SNS. Biaya Amazon SNS berlaku saat menggunakan pemberitahuan peristiwa. Untuk informasi selengkapnya tentang penagihan Amazon SNS, lihat Harga Amazon Simple Notification Service.

Contoh peristiwa Aurora

Contoh berikut menggambarkan berbagai jenis peristiwa Aurora dalam format JSON. Untuk tutorial yang menunjukkan cara menangkap dan melihat peristiwa dalam format JSON, lihat Tutorial: Mencatat perubahan status instans DB menggunakan Amazon EventBridge.

Topik

- Contoh peristiwa klaster DB
- Contoh peristiwa grup parameter DB
- Contoh peristiwa snapshot klaster DB

Contoh peristiwa klaster DB

Berikut ini adalah contoh peristiwa klaster DB dalam format JSON. Peristiwa ini menunjukkan bahwa klaster bernama `my-db-cluster` di-patch. ID peristiwanya adalah `RDS-EVENT-0173`.

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster"
  ],
  "detail": {
    "EventCategories": [
```

```
      "notification"
    ],
    "SourceType": "CLUSTER",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-db-cluster",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Database cluster has been patched",
    "SourceIdentifier": "my-db-cluster",
    "EventID": "RDS-EVENT-0173"
  }
}
```

Contoh peristiwa grup parameter DB

Berikut ini adalah contoh peristiwa grup parameter DB dalam format JSON. Peristiwa ini menunjukkan bahwa parameter `time_zone` telah diperbarui dalam grup parameter `my-db-param-group`. ID peristiwanya adalah RDS-EVENT-0037.

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Parameter Group Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group"
  ],
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PARAM",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:pg:my-db-param-group",
    "Date": "2018-10-06T12:26:13.882Z",
    "Message": "Updated parameter time_zone to UTC with apply method immediate",
    "SourceIdentifier": "my-db-param-group",
    "EventID": "RDS-EVENT-0037"
  }
}
```

Contoh peristiwa snapshot klaster DB

Berikut ini adalah contoh peristiwa snapshot klaster DB dalam format JSON. Peristiwa ini menunjukkan pembuatan snapshot bernama `my-db-cluster-snapshot`. ID peristiwanya adalah RDS-EVENT-0074.

```
{
  "version": "0",
  "id": "844e2571-85d4-695f-b930-0153b71dcb42",
  "detail-type": "RDS DB Cluster Snapshot Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-cluster-snapshot"
  ],
  "detail": {
    "EventCategories": [
      "backup"
    ],
    "SourceType": "CLUSTER_SNAPSHOT",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster-snapshot:rds:my-db-
cluster-snapshot",
    "Date": "2018-10-06T12:26:13.882Z",
    "SourceIdentifier": "my-db-cluster-snapshot",
    "Message": "Creating manual cluster snapshot",
    "EventID": "RDS-EVENT-0074"
  }
}
```

Memberikan izin untuk menerbitkan pemberitahuan ke topik Amazon SNS

Untuk memberikan izin Amazon RDS untuk menerbitkan pemberitahuan ke topik Amazon Simple Notification Service (Amazon SNS), lampirkan kebijakan AWS Identity and Access Management (IAM) ke topik tujuan. Untuk informasi selengkapnya tentang izin, lihat Contoh kasus untuk kontrol akses Amazon Simple Notification Service di Panduan Developer Amazon Simple Notification Service.

Secara default, topik Amazon SNS memiliki kebijakan yang mengizinkan semua sumber daya Amazon RDS dalam akun yang sama untuk menerbitkan pemberitahuan ke akun tersebut. Anda dapat melampirkan kebijakan kustom untuk mengizinkan pemberitahuan lintas akun, atau untuk membatasi akses ke sumber daya tertentu.

Berikut ini adalah contoh kebijakan IAM yang Anda lampirkan ke topik Amazon SNS tujuan. Ini membatasi topik ke instans DB dengan nama yang cocok dengan awalan yang ditentukan. Untuk menggunakan kebijakan ini, tentukan nilai berikut:

- `Resource` – Amazon Resource Name (ARN) untuk topik Amazon SNS Anda
- `SourceARN` – ARN sumber data RDS Anda
- `SourceAccount` – ID Akun AWS Anda

Untuk melihat daftar jenis sumber daya dan ARN-nya, lihat Sumber Daya yang Ditentukan oleh Amazon RDS di Referensi Otorisasi Layanan.

```
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "events.rds.amazonaws.com"
      },
      "Action": [
        "sns:Publish"
      ],
      "Resource": "arn:aws:sns:us-east-1:123456789012:topic_name",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:prefix-*"
        },
```

```
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        }
      }
    }
  ]
}
```

Berlangganan pemberitahuan peristiwa Amazon RDS

Cara termudah untuk membuat langganan adalah dengan konsol RDS. Jika memilih untuk membuat langganan pemberitahuan peristiwa menggunakan CLI atau API, Anda harus membuat topik Amazon Simple Notification Service dan berlangganan topik tersebut dengan konsol Amazon SNS atau Amazon SNS API. Anda juga akan perlu mempertahankan Amazon Resource Name (ARN) topik tersebut karena digunakan saat mengirim perintah CLI atau operasi API. Untuk informasi tentang cara membuat topik SNS dan berlanggan, lihat Mulai menggunakan Amazon SNS dalam Panduan Developer Amazon Simple Notification Service.

Anda dapat menentukan jenis sumber yang pemberitahuannya ingin dikirim dan sumber Amazon RDS yang memicu peristiwa.

Jenis sumber

Jenis sumber. Misalnya, Jenis sumber mungkin berupa Instans. Anda harus memilih jenis sumber.

Sumber daya yang akan disertakan

Sumber daya Amazon RDS yang menghasilkan peristiwa. Misalnya, Anda dapat memilih Pilih instans tertentu, lalu myDBInstance1.

Tabel berikut menjelaskan hasil saat Anda menentukan atau tidak menentukan ***Sumber Daya*** yang akan disertakan.

Sumber daya yang akan disertakan	Deskripsi	Contoh
Ditentukan	RDS memberi tahu Anda tentang semua peristiwa hanya untuk sumber daya yang ditentukan.	Jika Jenis sumber Anda berupa Instans dan sumber daya Anda adalah MyDBInstance1, RDS akan memberi tahu Anda tentang semua peristiwa hanya untuk myDBInstance1.
Tidak ditentukan	RDS memberi tahu Anda tentang peristiwa untuk jenis sumber yang ditentukan untuk semua sumber daya Amazon RDS.	Jika Jenis sumber Anda berupa Instans, RDS akan memberi tahu

Sumber daya yang akan disertakan	Deskripsi	Contoh
		Anda tentang semua peristiwa terkait instans di akun Anda.

Secara default, pelanggan topik Amazon SNS menerima setiap pesan yang diterbitkan untuk topik tersebut. Untuk menerima subset pesan saja, pelanggan harus menetapkan kebijakan filter untuk langganan topik. Untuk informasi selengkapnya tentang pemfilteran pesan SNS, lihat Pemfilteran pesan Amazon SNS di Panduan Developer Amazon Notification Service

Konsol

Untuk berlangganan pemberitahuan peristiwa RDS

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Langganan peristiwa.
3. Di panel Langganan peristiwa, pilih Buat langganan peristiwa.
4. Masukkan detail langganan Anda sebagai berikut:

   a. Untuk Nama, masukkan nama langganan pemberitahuan peristiwa.

   b. Untuk Kirim pemberitahuan ke, lakukan salah satu langkah berikut:

      - Pilih Topik email baru. Masukkan nama topik email Anda dan daftar penerima. Sebaiknya Anda mengonfigurasi langganan peristiwa ke alamat email yang sama dengan kontak akun utama Anda. Rekomendasi, peristiwa layanan, dan pesan kesehatan pribadi dikirim menggunakan saluran yang berbeda. Langganan ke alamat email yang sama memastikan bahwa semua pesan digabungkan di satu lokasi.
      - Pilih Amazon Resource Name (ARN). Kemudian, pilih Amazon SNS ARN untuk topik Amazon SNS.

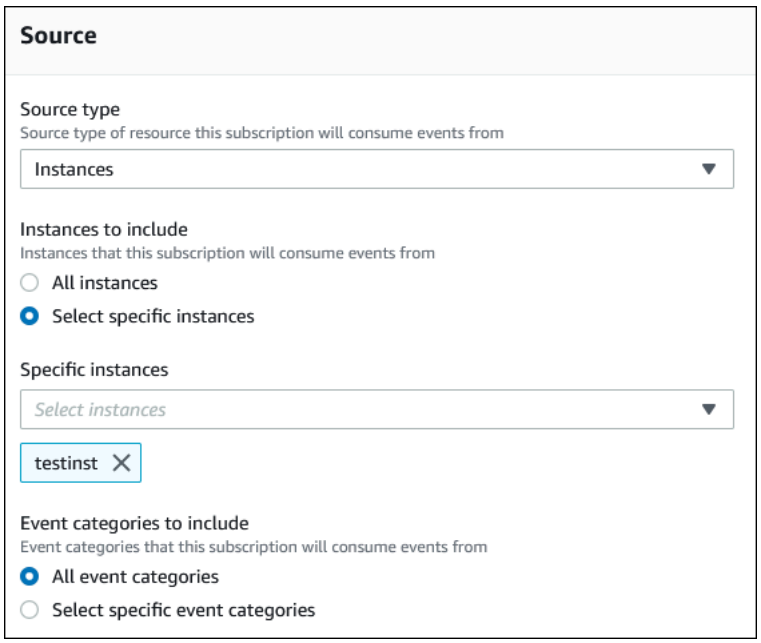
        Jika Anda ingin menggunakan topik yang telah diaktifkan untuk enkripsi di sisi server (SSE), beri Amazon RDS izin yang diperlukan untuk mengakses AWS KMS key. Untuk informasi selengkapnya, lihat Mengaktifkan kompatibilitas antara sumber peristiwa dari

layanan AWS dan topik terenkripsi di Panduan Developer Amazon Simple Notification Service.

c. Untuk Jenis sumber, pilih jenis sumber. Misalnya, pilih Klaster atau Snapshot klaster.

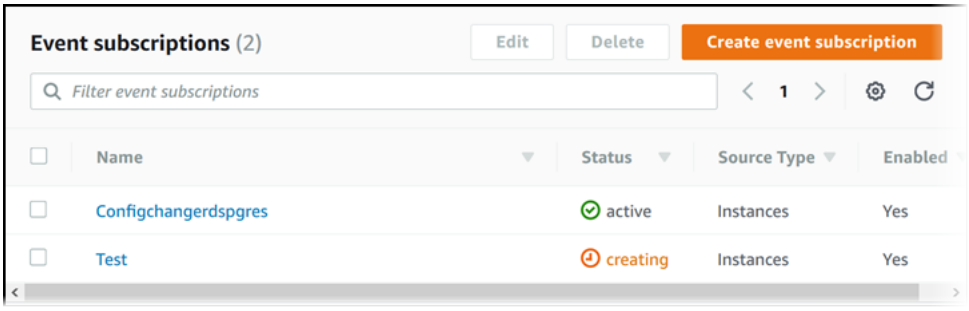
d. Pilih kategori dan sumber daya peristiwa yang pemberitahuan peristiwanya ingin diterima.

Contoh berikut mengonfigurasi pemberitahuan peristiwa untuk instans DB bernama `testinst`.



e. Pilih Buat.

Konsol Amazon RDS menunjukkan bahwa langganan sedang dibuat.



AWS CLI

Untuk berlangganan pemberitahuan peristiwa RDS, gunakan perintah AWS CLI `create-event-subscription`. Sertakan parameter wajib berikut:

- `--subscription-name`

- `--sns-topic-arn`

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-event-subscription \
    --subscription-name myeventsubscription \
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS \
    --enabled
```

Untuk Windows:

```
aws rds create-event-subscription ^
    --subscription-name myeventsubscription ^
    --sns-topic-arn arn:aws:sns:us-east-1:123456789012:myawsuser-RDS ^
    --enabled
```

API

Untuk berlangganan pemberitahuan peristiwa Amazon RDS, panggil fungsi Amazon RDS API `CreateEventSubscription`. Sertakan parameter wajib berikut:

- `SubscriptionName`
- `SnsTopicArn`

Tag dan atribut pemberitahuan peristiwa Amazon RDS

Saat Amazon RDS mengirimkan pemberitahuan peristiwa ke Amazon Simple Notification Service (SNS) atau Amazon EventBridge, pemberitahuan tersebut berisi atribut pesan dan tag peristiwa. RDS mengirimkan atribut pesan secara terpisah bersama dengan pesan, sedangkan tag peristiwa berada dalam isi pesan. Gunakan atribut pesan dan tag Amazon RDS untuk menambahkan metadata ke sumber daya Anda. Anda dapat mengubah tag ini dengan notasi Anda sendiri tentang instans DB, klaster Aurora, dan sebagainya. Untuk informasi selengkapnya tentang cara memberikan tag pada sumber daya Amazon RDS, lihat Memberi tag pada sumber daya Amazon RDS.

Secara default, Amazon SNS dan Amazon EventBridge menerima setiap pesan yang dikirim kepada mereka. SNS dan EventBridge dapat memfilter pesan dan mengirim pemberitahuan ke mode komunikasi yang diinginkan, seperti email, pesan teks, atau panggilan ke titik akhir HTTP.

> **Note**
>
> Pemberitahuan yang dikirim dalam email atau pesan teks tidak akan memiliki tag peristiwa.

Tabel berikut menunjukkan atribut pesan untuk peristiwa RDS yang dikirim ke pelanggan topik.

Atribut peristiwa Amazon RDS	Deskripsi
EventID	ID untuk pesan peristiwa RDS, misalnya, RDS-EVENT-0006.
Sumber daya	Pengidentifikasi ARN untuk sumber daya yang memancarkan peristiwa, misalnya, `arn:aws:rds:ap-southeast-2:123456789012:db:database-1` .

Tag RDS menyediakan data tentang sumber daya yang terpengaruh oleh peristiwa layanan. RDS menambahkan status tag saat ini dalam isi pesan saat pemberitahuan dikirim ke SNS atau EventBridge.

Untuk informasi selengkapnya tentang cara memfilter atribut pesan untuk SNS, lihat Pemfilteran pesan Amazon SNS dalam Panduan Developer Amazon Simple Notification Service.

Untuk informasi selengkapnya tentang cara memfilter tag peristiwa untuk EventBridge, lihat Pemfilteran konten dalam pola peristiwa Amazon EventBridge dalam Panduan Pengguna Amazon EventBridge.

Untuk informasi selengkapnya tentang cara memfilter tag berbasis payload untuk SNS, lihat https://aws.amazon.com/blogs/compute/introducing-payload-based-message-filtering-for-amazon-sns/

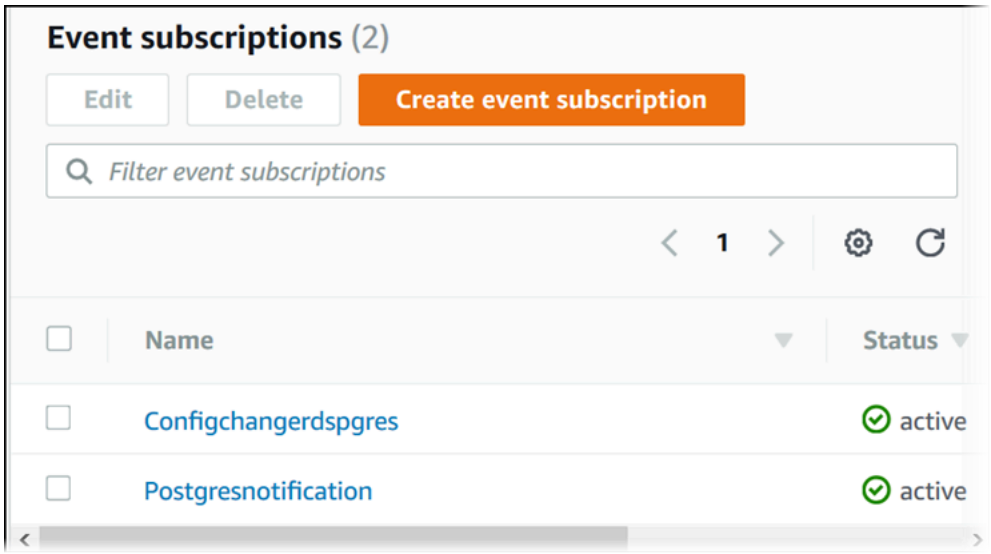
Mencantumkan langganan pemberitahuan peristiwa Amazon RDS

Anda dapat mencantumkan langganan pemberitahuan peristiwa Amazon RDS Anda saat ini.

Konsol

Untuk mencantumkan langganan pemberitahuan peristiwa Amazon RDS Anda saat ini

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Langganan peristiwa. Panel Langganan peristiwa menampilkan semua langganan pemberitahuan peristiwa.



AWS CLI

Untuk mencantumkan langganan pemberitahuan peristiwa Amazon RDS Anda saat ini, gunakan perintah AWS CLI `describe-event-subscriptions`.

Example

Contoh berikut menjelaskan semua langganan peristiwa.

```
aws rds describe-event-subscriptions
```

Contoh berikut menjelaskan `myfirsteventsubscription`.

```
aws rds describe-event-subscriptions --subscription-name myfirsteventsubscription
```

API

Untuk mencantumkan langganan pemberitahuan peristiwa Amazon RDS Anda saat ini, panggil tindakan Amazon RDS API DescribeEventSubscriptions.

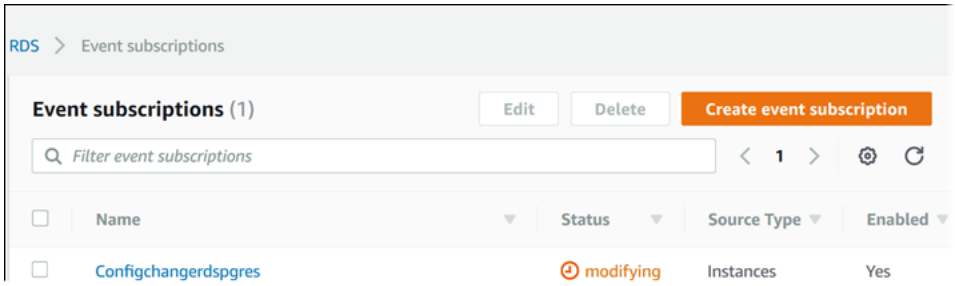
Mengubah langganan pemberitahuan peristiwa Amazon RDS

Setelah membuat langganan, Anda dapat mengubah nama langganan, pengidentifikasi sumber, kategori, atau ARN topik.

Konsol

Untuk mengubah langganan pemberitahuan peristiwa Amazon RDS

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Langganan peristiwa.
3. Di panel Langganan peristiwa, pilih langganan yang ingin diubah, lalu pilih Edit.
4. Buat perubahan pada langganan di bagian Target atau Sumber.
5. Pilih Edit. Konsol Amazon RDS menunjukkan bahwa langganan sedang diubah.



AWS CLI

Untuk mengubah langganan pemberitahuan peristiwa Amazon RDS, gunakan perintah AWS CLI `modify-event-subscription`. Sertakan parameter wajib berikut:

- `--subscription-name`

Example

Kode berikut mengaktifkan `myeventsubscription`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-event-subscription \
    --subscription-name myeventsubscription \
```

```
    --enabled
```

Untuk Windows:

```
aws rds modify-event-subscription ^
    --subscription-name myeventsubscription ^
    --enabled
```

API

Untuk mengubah peristiwa Amazon RDS, panggil operasi Amazon RDS API `ModifyEventSubscription`. Sertakan parameter wajib berikut:

- `SubscriptionName`

Menambahkan pengidentifikasi sumber ke langganan pemberitahuan peristiwa Amazon RDS

Anda dapat menambahkan pengidentifikasi sumber (sumber Amazon RDS yang menghasilkan peristiwa) ke langganan yang sudah ada.

Konsol

Anda dapat dengan mudah menambahkan atau menghapus ID sumber menggunakan konsol Amazon RDS dengan memilih atau membatalkan pilihan saat memodifikasi langganan. Untuk informasi selengkapnya, lihat Mengubah langganan pemberitahuan peristiwa Amazon RDS.

AWS CLI

Untuk menambahkan pengidentifikasi sumber ke langganan pemberitahuan peristiwa Amazon RDS, gunakan perintah AWS CLI `add-source-identifier-to-subscription`. Sertakan parameter wajib berikut:

- `--subscription-name`
- `--source-identifier`

Example

Contoh berikut menambahkan pengidentifikasi sumber `mysqldb` ke langganan `myrdseventsubscription`.

Untuk Linux, macOS, atau Unix:

```
aws rds add-source-identifier-to-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

Untuk Windows:

```
aws rds add-source-identifier-to-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

Untuk menambahkan pengidentifikasi sumber ke langganan pemberitahuan peristiwa Amazon RDS, panggil Amazon RDS API `AddSourceIdentifierToSubscription`. Sertakan parameter wajib berikut:

- `SubscriptionName`
- `SourceIdentifier`

Menghapus pengidentifikasi sumber dari langganan pemberitahuan peristiwa Amazon RDS

Anda dapat menghapus pengidentifikasi sumber (sumber Amazon RDS yang menghasilkan peristiwa) dari langganan jika Anda tidak ingin diberi tahu lagi tentang peristiwa sumber tersebut.

Konsol

Anda dapat dengan mudah menambahkan atau menghapus ID sumber menggunakan konsol Amazon RDS dengan memilih atau membatalkan pilihan saat memodifikasi langganan. Untuk informasi selengkapnya, lihat Mengubah langganan pemberitahuan peristiwa Amazon RDS.

AWS CLI

Untuk menghapus pengidentifikasi sumber dari langganan pemberitahuan peristiwa Amazon RDS, gunakan perintah AWS CLI `remove-source-identifier-from-subscription`. Sertakan parameter wajib berikut:

- `--subscription-name`
- `--source-identifier`

Example

Contoh berikut menghapus pengidentifikasi sumber `mysqldb` dari langganan `myrdseventsubscription`.

Untuk Linux, macOS, atau Unix:

```
aws rds remove-source-identifier-from-subscription \
    --subscription-name myrdseventsubscription \
    --source-identifier mysqldb
```

Untuk Windows:

```
aws rds remove-source-identifier-from-subscription ^
    --subscription-name myrdseventsubscription ^
    --source-identifier mysqldb
```

API

Untuk menghapus pengidentifikasi sumber dari langganan pemberitahuan peristiwa Amazon RDS, gunakan perintah Amazon RDS API `RemoveSourceIdentifierFromSubscription`. Sertakan parameter wajib berikut:

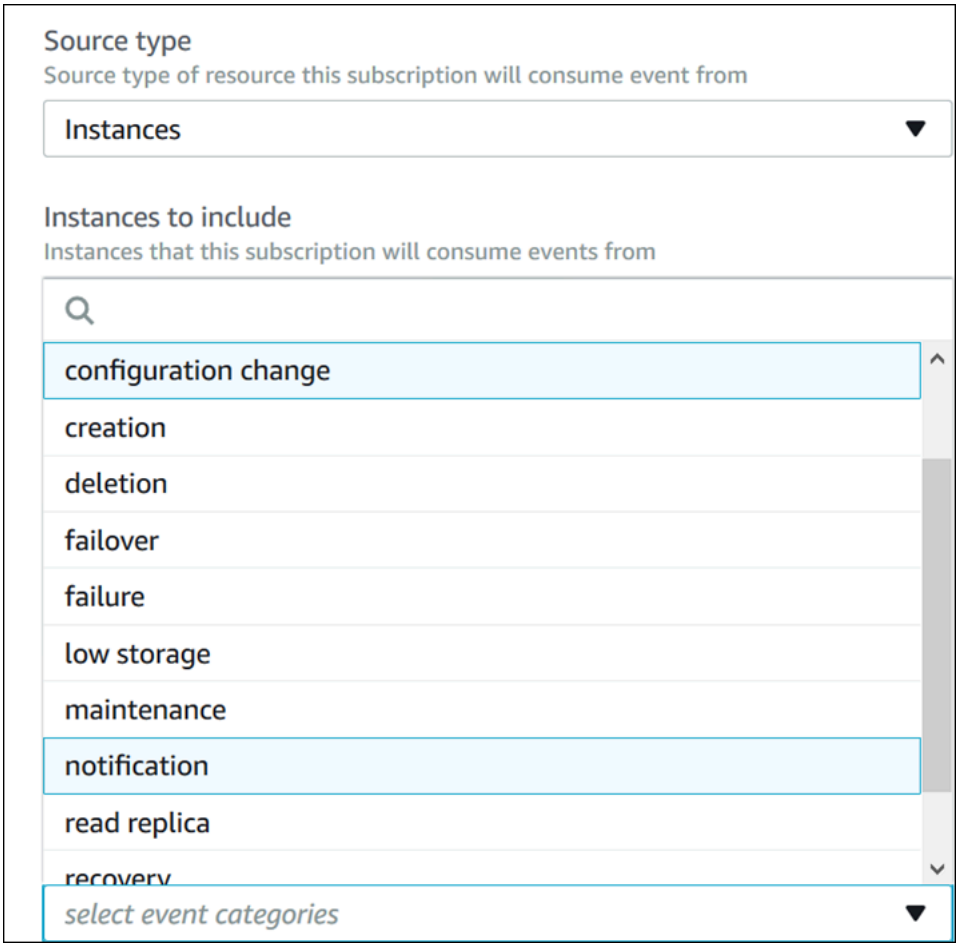
- `SubscriptionName`
- `SourceIdentifier`

Mencantumkan kategori pemberitahuan peristiwa Amazon RDS

Semua peristiwa untuk jenis sumber daya dikelompokkan dalam beberapa kategori. Untuk melihat daftar kategori yang tersedia, gunakan prosedur berikut.

Konsol

Jika Anda membuat atau memodifikasi langganan pemberitahuan peristiwa, kategori peristiwa akan ditampilkan di konsol Amazon RDS. Untuk informasi selengkapnya, lihat Mengubah langganan pemberitahuan peristiwa Amazon RDS.



AWS CLI

Untuk mencantumkan kategori pemberitahuan peristiwa Amazon RDS, gunakan perintah AWS CLI `describe-event-categories`. Perintah ini tidak memiliki parameter yang diperlukan.

Example

```
aws rds describe-event-categories
```

API

Untuk mencantumkan kategori pemberitahuan peristiwa Amazon RDS, gunakan perintah Amazon RDS API `DescribeEventCategories`. Perintah ini tidak memiliki parameter yang diperlukan.

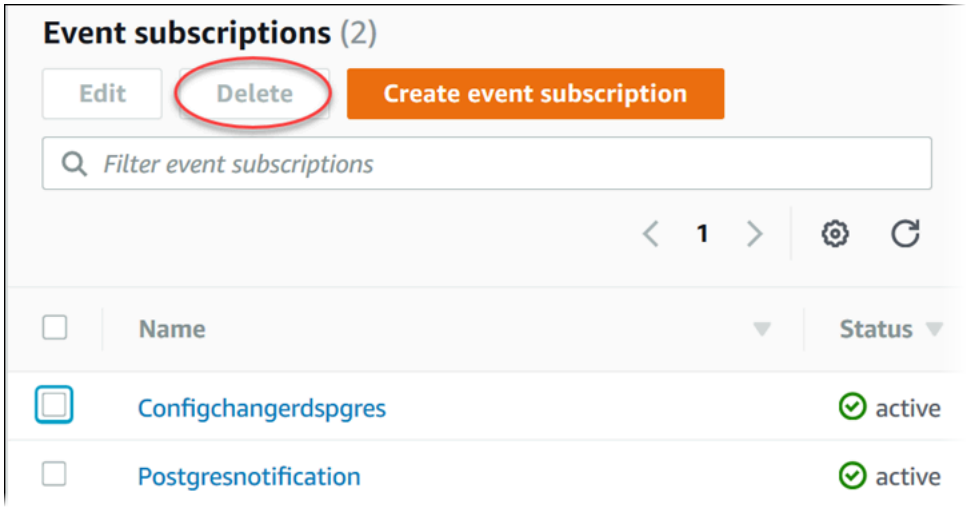
Menghapus langganan pemberitahuan peristiwa Amazon RDS

Anda dapat menghapus langganan jika sudah tidak membutuhkannya lagi. Semua pelanggan topik tidak akan lagi menerima pemberitahuan peristiwa yang ditentukan oleh langganan.

Konsol

Untuk menghapus langganan pemberitahuan peristiwa Amazon RDS

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Langganan Peristiwa DB.
3. Di panel Langganan Peristiwa DB Saya, pilih langganan yang ingin dihapus.
4. Pilih Hapus.
5. Konsol Amazon RDS menunjukkan bahwa langganan sedang diubah.



AWS CLI

Untuk menghapus langganan pemberitahuan peristiwa Amazon RDS, gunakan perintah AWS CLI `delete-event-subscription`. Sertakan parameter wajib berikut:

- `--subscription-name`

Example

Contoh berikut menghapus langganan `myrdssubscription`.

```
aws rds delete-event-subscription --subscription-name myrdssubscription
```

API

Untuk menghapus langganan pemberitahuan peristiwa Amazon RDS, gunakan perintah RDS API `DeleteEventSubscription`. Sertakan parameter wajib berikut:

- `SubscriptionName`

Membuat aturan yang dipicu pada peristiwa Amazon Aurora

Dengan Amazon CloudWatch Events dan Amazon EventBridge, Anda dapat mengotomatisasi layanan AWS dan merespons peristiwa sistem seperti masalah ketersediaan aplikasi atau perubahan sumber daya.

Topik

- Tutorial: Mencatat perubahan status instans DB menggunakan Amazon EventBridge

Tutorial: Mencatat perubahan status instans DB menggunakan Amazon EventBridge

Tutorial ini menjelaskan cara membuat fungsi AWS Lambda yang mencatat perubahan status untuk instans . Kemudian, Anda dapat membuat aturan yang menjalankan fungsi setiap kali terjadi perubahan status instans DB RDS yang ada. Tutorial ini mengasumsikan bahwa Anda memiliki instans pengujian kecil yang sedang berjalan yang dapat ditutup sementara.

> **Important**
>
> Jangan lakukan tutorial ini pada instans DB produksi yang sedang berjalan.

Topik

- Langkah 1: Buat fungsi AWS Lambda
- Langkah 2: Buat Aturan
- Langkah 3: Uji aturan

Langkah 1: Buat fungsi AWS Lambda

Buat fungsi Lambda untuk mencatat peristiwa perubahan status. Tetapkan fungsi ini saat membuat aturan.

Untuk membuat fungsi Lambda

1. Buka konsol AWS Lambda di https://console.aws.amazon.com/lambda/.
2. Jika baru menggunakan Lambda, Anda akan melihat halaman selamat datang. Pilih Mulai Sekarang. Atau, pilih Buat fungsi.
3. Pilih Tulis dari awal.

4. Di halaman Buat fungsi, lakukan langkah berikut:

   a. Masukkan nama dan deskripsi untuk fungsi Lambda. Misalnya, beri nama fungsi **`RDSInstanceStateChange`**.

   b. Dalam Runtime, pilih Node.js 16x.

   c. Untuk Arsitektur, pilih x86_64.

   d. Untuk Peran eksekusi, lakukan salah satu langkah berikut:

      - Pilih Buat peran baru dengan izin Lambda dasar.
      - Untuk Peran yang sudah ada, pilih Gunakan peran yang sudah ada. Pilih peran yang ingin digunakan.

   e. Pilih Buat fungsi.

5. Di halaman RDSInstanceStateChange, lakukan tindakan berikut:

   a. Di Sumber kode, pilih index.js.

   b. Di panel index.js, hapus kode yang ada.

   c. Masukkan kode berikut:

```
console.log('Loading function');

exports.handler = async (event, context) => {
    console.log('Received event:', JSON.stringify(event));
};
```

   d. Pilih Deploy.

Langkah 2: Buat Aturan

Buat aturan untuk menjalankan fungsi Lambda setiap kali Anda meluncurkan instans Amazon RDS.

Untuk membuat aturan EventBridge

1. Buka konsol Amazon EventBridge di https://console.aws.amazon.com/events/.
2. Di panel navigasi, pilih Aturan.
3. Pilih Buat aturan.
4. Masukkan nama dan deskripsi aturan. Misalnya, masukkan **`RDSInstanceStateChangeRule`**.
5. Pilih Aturan dengan pola peristiwa, lalu pilih Berikutnya.

6. Untuk Sumber peristiwa, pilih peristiwa AWS atau peristiwa mitra EventBridge.
7. Gulir ke bawah ke bagian Pola peristiwa.
8. Untuk Sumber peristiwa, pilih Layanan AWS.
9. Untuk layanan AWS, pilih Layanan Basis Data Relasional (RDS).
10. Untuk Jenis peristiwa, pilih Peristiwa Instans DB RDS.
11. Biarkan pola peristiwa default. Lalu pilih Selanjutnya.
12. Untuk Jenis target, pilih Layanan AWS.
13. Untuk Pilih target, pilih Fungsi Lambda.
14. Untuk Fungsi, pilih fungsi Lambda yang Anda buat. Lalu pilih Selanjutnya.
15. Di Konfigurasikan tag, pilih Berikutnya.
16. Tinjau langkah-langkah dalam aturan Anda. Kemudian, pilih Buat aturan.

Langkah 3: Uji aturan

Untuk menguji aturan, tutup instans DB RDS. Setelah menunggu beberapa menit untuk instans yang akan dinonaktifkan, verifikasi bahwa fungsi Lambda Anda sudah diinvokasi.

Untuk menguji aturan dengan menghentikan instans DB

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Hentikan suatu instans DB RDS.
3. Buka konsol Amazon EventBridge di https://console.aws.amazon.com/events/.
4. Di panel navigasi, pilih Aturan, pilih nama aturan yang Anda buat.
5. Dalam Detail aturan, pilih Pemantauan.

   Anda akan dialihkan ke konsol Amazon CloudWatch. Jika Anda tidak dialihkan, klik Lihat metrik di CloudWatch.
6. Dalam Semua metrik, pilih nama aturan yang Anda buat.

   Grafik harus menunjukkan bahwa aturan diinvokasi.
7. Di panel navigasi, pilih Grup log.
8. Pilih nama grup log untuk fungsi Lambda Anda (/aws/lambda/***function-name***).
9. Pilih nama log stream untuk melihat data yang disediakan oleh fungsi untuk instans yang Anda luncurkan. Anda akan melihat peristiwa yang mirip dengan yang berikut ini:

```
{
    "version": "0",
    "id": "12a345b6-78c9-01d2-34e5-123f4ghi5j6k",
    "detail-type": "RDS DB Instance Event",
    "source": "aws.rds",
    "account": "111111111111",
    "time": "2021-03-19T19:34:09Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:rds:us-east-1:111111111111:db:testdb"
    ],
    "detail": {
        "EventCategories": [
            "notification"
        ],
        "SourceType": "DB_INSTANCE",
        "SourceArn": "arn:aws:rds:us-east-1:111111111111:db:testdb",
        "Date": "2021-03-19T19:34:09.293Z",
        "Message": "DB instance stopped",
        "SourceIdentifier": "testdb",
        "EventID": "RDS-EVENT-0087"
    }
}
```


Untuk contoh peristiwa RDS lain dalam format JSON, lihat Ikhtisar peristiwa untuk Aurora.

10. (Opsional) Setelah selesai, Anda dapat membuka konsol Amazon RDS dan memulai instans yang Anda hentikan.

Kategori peristiwa dan pesan peristiwa Amazon RDS

Amazon RDS menghasilkan sejumlah besar peristiwa dalam kategori yang dapat Anda berlangganan menggunakan Konsol Amazon RDS, AWS CLI, atau API.

Topik

- Peristiwa klaster DB
- Peristiwa instans DB
- Peristiwa grup parameter DB
- Peristiwa grup keamanan DB
- Peristiwa snapshot klaster DB
- Peristiwa Proksi RDS
- Peristiwa deployment blue/green

Peristiwa klaster DB

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat klaster DB berupa jenis sumber.

> **Note**
>
> Tidak ada kategori peristiwa yang ada untuk Aurora Serverless dalam jenis peristiwa klaster DB. Peristiwa Aurora Nirserver berkisar antara RDS-EVENT-0141 hingga RDS-EVENT-0149.

Kategori	ID peristiwa RDS	Pesan	Catatan
perubahan konfigurasi	RDS-EVENT-0016	Atur ulang kredensial utama.	
perubahan konfigurasi	RDS-EVENT-0179	Aliran Aktivitas Basis Data dimulai pada klaster basis data Anda.	Untuk mengetahui informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.

Kategori	ID peristiwa RDS	Pesan	Catatan
perubahan konfigurasi	RDS-EVENT-0180	Aliran Aktivitas Basis Data dihentikan pada klaster basis data Anda.	Untuk mengetahui informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.
pembuatan	RDS-EVENT-0170	Klaster DB dibuat.	
penghapusan	RDS-EVENT-0171	Klaster DB dihapus.	
failover	RDS-EVENT-0069	Failover klaster gagal, periksa kondisi instans klaster dan coba lagi.	
failover	RDS-EVENT-0070	Mendorong primer sebelumnya lagi: *name*.	
failover	RDS-EVENT-0071	Menyelesaikan failover ke instans DB: *name*.	
failover	RDS-EVENT-0072	Memulai failover AZ yang sama ke instans DB: *name*.	
failover	RDS-EVENT-0073	Memulai failover lintas AZ ke instans DB: *name*.	

Kategori	ID peristiwa RDS	Pesan	Catatan
kegagalan	RDS-EVENT-0083	Amazon RDS tidak dapat membuat kredensial untuk mengakses Bucket Amazon S3 untuk klaster DB Anda *name*. Hal ini disebabkan peran IAM konsumsi snapshot S3 tidak dikonfigurasi dengan benar di akun Anda atau bucket Amazon S3 yang ditentukan tidak dapat ditemukan. Untuk detail selengkapnya, lihat bagian pemecahan masalah dalam dokumentasi Amazon RDS.	Untuk mengetahui informasi selengkapnya, lihat Migrasi fisik dari MySQL dengan menggunakan Percona XtraBackup dan Amazon S3 .
kegagalan	RDS-EVENT-0143	Klaster DB gagal diskalakan dari *units* ke *units* karena alasan ini: *reason*.	Penskalaan gagal untuk klaster DB Aurora Serverless.
kegagalan	RDS-EVENT-0354	Anda tidak dapat membuat klaster DB karena sumber daya yang tidak kompatibel. *pesan*.	*Pesan* ini mencakup detail tentang kegagalan.
kegagalan	RDS-EVENT-0355	Klaster DB tidak dapat dibuat karena batas sumber daya yang tidak memadai. *pesan*.	*Pesan* ini mencakup detail tentang kegagalan.

Kategori	ID peristiwa RDS	Pesan	Catatan
failover global	RDS-EVENT-0181	Switchover global ke klaster DB *name* di Wilayah *name* dimulai.	Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola"). Prosesnya dapat ditunda karena operasi lain berjalan di klaster DB.
failover global	RDS-EVENT-0182	Klaster DB primer lama *name* di Wilayah *name* berhasil dimatikan.	Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola"). Instans primer lama dalam basis data global tidak menerima penulisan. Semua volume disinkron kan.
failover global	RDS-EVENT-0183	Menunggu sinkronisasi data di seluruh anggota klaster global. Arus tertinggal di belakang klaster DB primer: *reason*.	Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola"). Kelambatan replikasi terjadi selama fase sinkronisasi failover basis data global.

Kategori	ID peristiwa RDS	Pesan	Catatan
failover global	RDS-EVENT-0184	Klaster DB primer baru *name* di Wilayah *name* berhasil dipromosikan.	Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola"). Topologi volume basis data global dibangun kembali dengan volume primer baru.
failover global	RDS-EVENT-0185	Switchover global ke klaster DB *name* di Wilayah *name* selesai.	Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola"). Switchover basis data global selesai di klaster DB primer. Replika mungkin membutuhkan waktu lama untuk online setelah failover selesai.
failover global	RDS-EVENT-0186	Switchover global ke klaster DB *name* di Wilayah *name* dibatalkan.	Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola").
failover global	RDS-EVENT-0187	Switchover global ke klaster DB *name* di Wilayah *name* gagal.	Peristiwa ini untuk operasi switchover (sebelumnya disebut "failover terencana terkelola").
failover global	RDS-EVENT-0238	Failover global ke klaster DB *name* di Wilayah *name* selesai.	

Kategori	ID peristiwa RDS	Pesan	Catatan
failover global	RDS-EVENT-0239	Failover global ke klaster DB *name* di Wilayah *name* gagal.	
failover global	RDS-EVENT-0240	Memulai sinkronisasi ulang anggota klaster DB *name* di Wilayah *name* setelah failover global.	
failover global	RDS-EVENT-0241	Menyelesaikan sinkronis asi ulang anggota klaster DB *name* di Wilayah *name* setelah failover global.	
pemeliharaan	RDS-EVENT-0156	Klaster DB memiliki peningkatan versi minor mesin DB yang tersedia.	
pemeliharaan	RDS-EVENT-0173	Versi mesin klaster basis data telah ditingkatkan.	Patching klaster DB telah selesai.
pemeliharaan	RDS-EVENT-0176	Versi mayor mesin klaster basis data telah ditingkat kan.	
pemeliharaan	RDS-EVENT-0286	Peningkatan versi mesin klaster basis data dimulai.	
pemeliharaan	RDS-EVENT-0287	Persyaratan peningkatan sistem operasi terdeteksi.	
pemeliharaan	RDS-EVENT-0288	Peningkatan sistem operasi klaster dimulai.	
pemeliharaan	RDS-EVENT-0289	Peningkatan sistem operasi klaster selesai.	

Kategori	ID peristiwa RDS	Pesan	Catatan
pemeliharaan	RDS-EVENT-0290	Klaster basis data telah di-patch: versi sumber *version_number* => *new_version_number* .	
pemberitahuan	RDS-EVENT-0076	Gagal bermigrasi dari *name* ke *name*. Alasan: *reason*.	Migrasi ke klaster DB Aurora gagal.
pemberitahuan	RDS-EVENT-0077	Gagal mengonversi *name*.*name* ke InnoDB. Alasan: *reason*.	Upaya untuk mengonversi tabel dari basis data sumber ke InnoDB gagal selama migrasi ke klaster DB Aurora.
pemberitahuan	RDS-EVENT-0085	Tidak dapat meningkatkan klaster DB *name* karena instans *name* memiliki status *name*. Selesaikan masalah atau hapus instans dan coba lagi.	Terjadi kesalahan saat mencoba untuk melakukan patch pada klaster DB Aurora. Periksa status instans Anda, selesaikan masalah, dan coba lagi. Untuk mengetahui informasi selengkapnya, lihat Memelihara klaster DB Amazon Aurora.
pemberitahuan	RDS-EVENT-0141	Menskalakan klaster DB dari *units* ke *units* karena alasan ini: *reason*.	Penskalaan dimulai untuk klaster DB Aurora Serverless.
pemberitahuan	RDS-EVENT-0142	Klaster DB telah diskalakan dari *units* ke *units*.	Penskalaan selesai untuk klaster DB Aurora Serverless.
pemberitahuan	RDS-EVENT-0144	Klaster DB sedang dijeda.	Jeda otomatis dimulai untuk klaster DB Aurora Nirserver.

Kategori	ID peristiwa RDS	Pesan	Catatan
pemberitahuan	RDS-EVENT-0145	Klaster DB dijeda.	Klaster DB Aurora Serverless telah dijeda.
pemberitahuan	RDS-EVENT-0146	Jeda dibatalkan untuk klaster DB.	Jeda dibatalkan untuk klaster DB Aurora Serverless.
pemberitahuan	RDS-EVENT-0147	Klaster DB sedang dilanjutkan kembali.	Operasi resume dimulai untuk klaster DB Aurora Serverless.
pemberitahuan	RDS-EVENT-0148	Klaster DB dilanjutkan kembali.	Operasi resume selesai untuk klaster DB Aurora Serverless.
pemberitahuan	RDS-EVENT-0149	Klaster DB telah diskalakan dari *`units`* ke *`units`*, tetapi penskalaan tidak lancar karena alasan ini: *`reason`*.	Penskalaan lancar selesai dengan opsi paksa untuk klaster DB Aurora Serverless. Koneksi mungkin terganggu sesuai kebutuhan.
pemberitahuan	RDS-EVENT-0150	Klaster DB dihentikan.	
pemberitahuan	RDS-EVENT-0151	Klaster DB dimulai.	
pemberitahuan	RDS-EVENT-0152	Perhentian klaster DB gagal.	
pemberitahuan	RDS-EVENT-0153	Klaster DB sedang dimulai karena melebihi waktu maksimal yang diizinkan untuk dihentikan.	

Kategori	ID peristiwa RDS	Pesan	Catatan
pemberitahuan	RDS-EVENT-0172	Mengganti nama klaster dari *name* menjadi *name*.	
pemberitahuan	RDS-EVENT-0234	Tugas ekspor gagal.	Tugas ekspor klaster DB gagal.
pemberitahuan	RDS-EVENT-0235	Tugas ekspor dibatalkan.	Tugas ekspor klaster DB dibatalkan.
pemberitahuan	RDS-EVENT-0236	Tugas ekspor selesai.	Tugas ekspor klaster DB selesai.

Peristiwa instans DB

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat instans DB merupakan jenis sumber.

Kategori	ID peristiwa RDS	Pesan	Catatan
ketersediaan	RDS-EVENT-0004	Instans DB dimatikan.	
ketersediaan	RDS-EVENT-0006	Instans DB dimulai ulang.	
ketersediaan	RDS-EVENT-0022	Terjadi kesalahan saat memulai ulang mysql: *pesan*.	Terjadi kesalahan saat memulai ulang Aurora MySQL atau RDS for MariaDB.
backtrack	RDS-EVENT-0131	Periode Backtrack aktual lebih kecil dari periode Backtrack target yang Anda tentukan. Pertimbangkan untuk mengurangi jumlah jam dalam periode Backtrack target Anda.	Untuk mengetahui informasi selengkapnya tentang backtracking, lihat Melakukan backtracking klaster DB Aurora.

Kategori	ID peristiwa RDS	Pesan	Catatan
backtrack	RDS-EVENT-0132	Periode Backtrack aktual sama dengan periode Backtrack target.	
perubahan konfigurasi	RDS-EVENT-0011	Diperbarui untuk menggunakan Parameter Group *nama* DB.	
perubahan konfigurasi	RDS-EVENT-0012	Menerapkan modifikasi pada kelas instans basis data.	
perubahan konfigurasi	RDS-EVENT-0014	Menyelesaikan penerapan modifikasi pada kelas instans DB.	
perubahan konfigurasi	RDS-EVENT-0017	Menyelesaikan penerapan modifikasi pada penyimpanan yang dialokasikan.	
perubahan konfigurasi	RDS-EVENT-0025	Menyelesaikan penerapan modifikasi untuk dikonversi menjadi instans DB Multi-AZ.	
perubahan konfigurasi	RDS-EVENT-0029	Menyelesaikan penerapan modifikasi untuk dikonversi menjadi instans DB standar (AZ Tunggal).	
perubahan konfigurasi	RDS-EVENT-0033	Ada *number* pengguna yang cocok dengan nama pengguna utama; hanya mengatur ulang yang tidak terikat dengan host tertentu.	

Kategori	ID peristiwa RDS	Pesan	Catatan
perubahan konfigurasi	RDS-EVENT-0067	Tidak dapat mengatur ulang kata sandi Anda. Informasi kesalahan: *pesan*.	
perubahan konfigurasi	RDS-EVENT-0078	Interval Pemantauan diubah menjadi *number*.	Konfigurasi Pemantaua n yang Ditingkatkan telah diubah.
perubahan konfigurasi	RDS-EVENT-0092	Menyelesaikan pembaruan grup parameter DB.	
pembuatan	RDS-EVENT-0005	Instans DB dibuat.	
penghapusan	RDS-EVENT-0003	Instans DB dihapus.	
kegagalan	RDS-EVENT-0035	Instans basis data dimasukkan dalam *state*. *pesan*.	Instans DB memiliki parameter yang tidak valid. Misalnya, jika instans DB tidak dapat dimulai karena parameter terkait memori diatur terlalu tinggi untuk kelas instans ini, tindakan Anda adalah memodifik asi parameter memori dan mem-boot ulang instans DB.
kegagalan	RDS-EVENT-0036	Instans basis data dalam *state*. *pesan*.	Instans DB ada di jaringan yang tidak kompatibel. Beberapa ID subnet yang ditentukan tidak valid atau tidak ada.

Kategori	ID peristiwa RDS	Pesan	Catatan
kegagalan	RDS-EVENT-0079	Amazon RDS tidak dapat membuat kredensial untuk pemantauan yang ditingkatkan dan fitur ini telah dinonaktifkan. Ini kemungkinan karena rds-monitoring-role tidak ada dan dikonfigurasi dengan benar di akun Anda. Untuk detail selengkapnya, lihat bagian pemecahan masalah dalam dokumentasi Amazon RDS.	Pemantauan yang Ditingkatkan tidak dapat diaktifkan tanpa peran IAM Pemantauan yang Ditingkatkan. Untuk mengetahui informasi tentang cara membuat peran IAM, lihat Untuk membuat peran IAM untuk pemantauan yang ditingkatkan Amazon RDS.
kegagalan	RDS-EVENT-0080	Amazon RDS tidak dapat mengonfigurasi pemantauan yang ditingkatkan pada instans Anda: *name* dan fitur ini telah dinonaktifkan. Ini kemungkinan karena rds-monitoring-role tidak ada dan dikonfigurasi dengan benar di akun Anda. Untuk detail selengkapnya, lihat bagian pemecahan masalah dalam dokumentasi Amazon RDS.	Pemantauan yang Ditingkatkan dinonaktifkan karena kesalahan terjadi saat perubahan konfigurasi. Sepertinya peran IAM Pemantauan yang Ditingkatkan tidak dikonfigurasi dengan benar. Untuk mengetahui informasi tentang cara membuat peran IAM pemantauan yang ditingkatkan, lihat Untuk membuat peran IAM untuk pemantauan yang ditingkatkan Amazon RDS.

Kategori	ID peristiwa RDS	Pesan	Catatan
kegagalan	RDS-EVENT-0082	Amazon RDS tidak dapat membuat kredensial untuk mengakses Bucket Amazon S3 untuk instans DB Anda *name*. Hal ini disebabkan peran IAM konsumsi snapshot S3 tidak dikonfigurasi dengan benar di akun Anda atau bucket Amazon S3 yang ditentukan tidak dapat ditemukan. Untuk detail selengkapnya, lihat bagian pemecahan masalah dalam dokumentasi Amazon RDS.	Aurora tidak dapat menyalin data cadangan dari bucket Amazon S3. Sepertinya izin Aurora untuk mengakses bucket Amazon S3 tidak dikonfigurasi dengan benar. Untuk mengetahui informasi selengkapnya, lihat Migrasi fisik dari MySQL dengan menggunakan Percona XtraBackup dan Amazon S3 .
kegagalan	RDS-EVENT-0254	Kuota penyimpanan yang mendasari untuk akun pelanggan ini telah melampaui batas. Tambah kuota penyimpanan yang diizinkan agar penskalaan dapat berjalan pada instans.	
kegagalan	RDS-EVENT-0353	Instans DB tidak dapat dibuat karena batas sumber daya tidak mencukupi. *pesan*.	*Pesan* ini mencakup detail tentang kegagalan.

Kategori	ID peristiwa RDS	Pesan	Catatan
penyimpanan rendah	RDS-EVENT-0007	Penyimpanan yang dialokasikan telah habis. Alokasikan penyimpanan tambahan untuk mengatasi masalah.	Penyimpanan yang dialokasikan untuk instans DB telah dipakai. Untuk mengatasi masalah ini, alokasikan penyimpanan tambahan untuk instans DB. Untuk mengetahui informasi selengkapnya, lihat Pertanyaan Umum tentang RDS. Anda dapat memantau ruang penyimpanan untuk instans DB menggunakan metrik Ruang Penyimpanan Kosong.
penyimpanan rendah	RDS-EVENT-0089	Kapasitas penyimpanan kosong untuk instans DB: *name* rendah pada *percentage* dari penyimpanan yang disediakan [Penyimpanan yang Disediakan: *size*, Ruang Kosong: *size*]. Anda mungkin ingin menambah penyimpanan yang disediakan untuk mengatasi masalah ini.	Instans DB telah menggunakan lebih dari 90% dari penyimpanan yang dialokasikan. Anda dapat memantau ruang penyimpanan untuk instans DB menggunakan metrik Ruang Penyimpanan Kosong.

Kategori	ID peristiwa RDS	Pesan	Catatan
penyimpanan rendah	RDS-EVENT-0227	Penyimpanan klaster Aurora Anda sangat rendah dengan hanya *amount* terabyte yang tersisa. Ambil tindakan untuk mengurangi muatan penyimpanan pada klaster Anda.	Ruang subsistem penyimpanan Aurora hampir habis.
pemeliharaan	RDS-EVENT-0026	Menerapkan patch off-line ke instans DB.	Pemeliharaan offline instans DB sedang berlangsung. Instans DB saat ini tidak tersedia.
pemeliharaan	RDS-EVENT-0027	Menyelesaikan penerapan patch off-line ke instans DB.	Pemeliharaan offline instans DB selesai. Instans DB kini tersedia.
pemeliharaan	RDS-EVENT-0047	Instans basis data di-patch.	
pemeliharaan	RDS-EVENT-0155	Instans DB memiliki peningkatan versi minor mesin DB yang tersedia.	
pemberitahuan	RDS-EVENT-0044	*pesan*	Ini adalah pemberitahuan yang dikeluarkan operator. Untuk mengetahui informasi selengkapnya, lihat pesan peristiwa.
pemberitahuan	RDS-EVENT-0048	Menunda peningkatan mesin basis data karena instans ini telah membaca replika yang perlu ditingkatkan terlebih dahulu.	Patching instans DB telah ditunda.

Kategori	ID peristiwa RDS	Pesan	Catatan
pemberitahuan	RDS-EVENT-0087	Instans DB dihentikan.	
pemberitahuan	RDS-EVENT-0088	Instans DB dimulai.	
replika baca	RDS-EVENT-0045	Replikasi telah berhenti.	Replikasi pada instans DB Anda telah dihentikan karena penyimpanan tidak mencukupi. Skalakan penyimpanan atau kurangi ukuran maksimum log redo agar replikasi berlanjut. Untuk mengakomodasi log redo ukuran %d MiB Anda memerlukan setidaknya %d MiB penyimpanan kosong.
replika baca	RDS-EVENT-0046	Replikasi untuk Replika Baca dilanjutkan.	Pesan ini muncul saat Anda pertama kali membuat replika baca, atau sebagai pesan pemantauan yang mengonfirmasi bahwa replikasi berfungsi dengan benar. Jika pesan ini mengikuti pemberitahuan `RDS-EVENT-0045` , replikasi telah dilanjutkan setelah kesalahan atau setelah replikasi dihentikan.
replika baca	RDS-EVENT-0057	Streaming replikasi telah dihentikan.	

Kategori	ID peristiwa RDS	Pesan	Catatan
pemulihan	RDS-EVENT-0020	Pemulihan instans DB telah dimulai. Waktu pemulihan beragam dengan jumlah data yang akan dipulihkan.	
pemulihan	RDS-EVENT-0021	Pemulihan instans DB selesai.	
pemulihan	RDS-EVENT-0023	Permintaan Snapshot Muncul: *pesan*.	Pencadangan manual telah diminta, tetapi Amazon RDS saat ini dalam proses pembuatan snapshot DB. Kirim permintaan lagi setelah Amazon RDS menyelesaikan snapshot DB.
pemulihan	RDS-EVENT-0052	Pemulihan instans Multi-AZ dimulai.	Waktu pemulihan beragam dengan jumlah data yang akan dipulihkan.
pemulihan	RDS-EVENT-0053	Pemulihan instans Multi-AZ selesai. Failover atau aktivasi tertunda.	
pemulihan	RDS-EVENT-0019	Dipulihkan dari instans DB *name* ke *name*.	Instans DB telah dipulihkan dari point-in-time cadangan.

Kategori	ID peristiwa RDS	Pesan	Catatan
patching keamanan	RDS-EVENT-0230	Pembaruan sistem tersedia untuk instans DB Anda. Untuk mengetahui informasi tentang cara menerapkan pembaruan, lihat 'Mempertahankan instans DB' di Panduan Pengguna RDS.	Patch Sistem Operasi baru tersedia. Pembaruan sistem operasi versi minor baru tersedia untuk instans DB Anda. Untuk mengetahui informasi tentang cara menerapkan pembaruan, lihat Bekerja dengan pembaruan sistem operasi.

Peristiwa grup parameter DB

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat grup parameter DB merupakan jenis sumber.

Kategori	ID peristiwa RDS	Pesan	Catatan
perubahan konfigurasi	RDS-EVENT-0037	Memperbarui parameter *name* ke *value* dengan metode penerapan *method*.	

Peristiwa grup keamanan DB

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat grup keamanan DB merupakan jenis sumber.

> **Note**
>
> Grup keamanan DB merupakan sumber daya untuk EC2-Classic. EC2-Classic sudah tidak digunakan lagi pada 15 Agustus 2022. Jika Anda belum bermigrasi dari EC2-Classic ke VPC, sebaiknya Anda bermigrasi sesegera mungkin. Untuk mengetahui informasi selengkapnya,

lihat Migrasi dari EC2-Classic ke VPC di Panduan Pengguna Amazon EC2 dan blog Jaringan EC2-Classic akan Segera Dihentikan – Berikut Cara Mempersiapkannya.

Kategori	ID peristiwa RDS	Pesan	Catatan
perubahan konfigurasi	RDS-EVENT-0038	Menerapkan perubahan ke grup keamanan.	
kegagalan	RDS-EVENT-0039	Mencabut otorisasi sebagai *`user`*.	Grup keamanan milik *`user`* tidak ada. Otorisasi untuk grup keamanan telah dicabut karena tidak valid.

Peristiwa snapshot klaster DB

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat snapshot klaster DB merupakan jenis sumber.

Kategori	ID peristiwa RDS	Pesan	Catatan
pencadangan	RDS-EVENT-0074	Membuat snapshot klaster manual.	
pencadangan	RDS-EVENT-0075	Snapshot klaster manual dibuat.	
pemberitahuan	RDS-EVENT-0162	Tugas ekspor snapshot klaster gagal.	
pemberitahuan	RDS-EVENT-0163	Tugas ekspor snapshot klaster dibatalkan.	
pemberitahuan	RDS-EVENT-0164	Tugas ekspor snapshot klaster selesai.	

Kategori	ID peristiwa RDS	Pesan	Catatan
pencadangan	RDS-EVENT-0168	Membuat snapshot klaster otomatis.	
pencadangan	RDS-EVENT-0169	Snapshot klaster otomatis dibuat.	

Peristiwa Proksi RDS

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat Proksi RDS berupa jenis sumber.

Kategori	ID peristiwa RDS	Pesan	Catatan
perubahan konfigurasi	RDS-EVENT-0204	RDS memodifikasi proksi DB *name*.	
perubahan konfigurasi	RDS-EVENT-0207	RDS memodifikasi titik akhir proksi DB *name*.	
perubahan konfigurasi	RDS-EVENT-0213	RDS mendeteksi penambahan instans DB dan secara otomatis menambahkannya ke grup target proksi DB *name*.	
perubahan konfigurasi	RDS-EVENT-0213	RDS mendeteksi pembuatan instans DB *name* dan secara otomatis menambahkannya ke grup target *name* proksi DB *name*.	
perubahan konfigurasi	RDS-EVENT-0214	RDS mendeteksi penghapusan instans DB *name* dan secara otomatis menghapusnya dari grup	

Kategori	ID peristiwa RDS	Pesan	Catatan
		target *name* proksi DB *name*.	
perubahan konfigurasi	RDS-EVENT-0215	RDS mendeteksi penghapusan klaster DB *name* dan secara otomatis menghapusnya dari grup target *name* proksi DB *name*.	
pembuatan	RDS-EVENT-0203	RDS membuat proksi DB *name*.	
pembuatan	RDS-EVENT-0206	RDS membuat titik akhir *name* untuk proksi DB *name*.	
penghapusan	RDS-EVENT-0205	RDS menghapus proksi DB *name*.	
penghapusan	RDS-EVENT-0208	RDS menghapus titik akhir *name* untuk proksi DB *name*.	

Kategori	ID peristiwa RDS	Pesan	Catatan
kegagalan	RDS-EVENT-0243	RDS gagal menyediak an kapasitas untuk proxy *name* karena tidak ada alamat IP yang cukup yang tersedia di subnet Anda: *name*. Untuk memperbai ki masalah ini, pastikan subnet Anda memiliki jumlah minimum alamat IP yang tidak digunakan seperti yang direkomen dasikan dalam dokumenta si Proksi RDS.	Untuk menentukan jumlah yang direkomendasikan untuk kelas instans Anda, lihat Perencanaan untuk kapasitas alamat IP.
kegagalan	RDS-EVENT-0275	*RDS membatasi beberapa koneksi ke nama proxy DB.* Jumlah permintaan koneksi simultan dari klien ke proxy telah melampaui batas.	

Peristiwa deployment blue/green

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat deployment blue/green merupakan jenis sumber.

Untuk mengetahui informasi selengkapnya tentang deployment blue/green, lihat Menggunakan Deployment Blue/Green Amazon RDS untuk pembaruan basis data.

Kategori	ID peristiwa Amazon RDS	Pesan	Catatan
pembuatan	RDS-EVENT-0244	Tugas deployment blue/ green selesai. Anda dapat membuat lebih banyak	

Kategori	ID peristiwa Amazon RDS	Pesan	Catatan
		modifikasi pada basis data lingkungan hijau atau beralih antar deployment.	
kegagalan	RDS-EVENT-0245	Pembuatan deployment blue/green gagal karena (instans/klaster) DB (sumber/target) tidak ditemukan.	
penghapusan	RDS-EVENT-0246	Deployment blue/green dihapus.	
pemberitahuan	RDS-EVENT-0247	Switchover dari *blue* ke *green* dimulai.	
pemberitahuan	RDS-EVENT-0248	Switchover selesai pada deployment blue/green.	
kegagalan	RDS-EVENT-0249	Switchover dibatalkan pada deployment blue/green.	
pemberitahuan	RDS-EVENT-0259	Switchover dari klaster DB *blue* ke *green* dimulai.	
pemberitahuan	RDS-EVENT-0260	Switchover dari klaster DB *blue* ke *green* selesai. Mengganti nama *blue* menjadi *blue-old* dan *green* menjadi *blue*.	
kegagalan	RDS-EVENT-0261	Switchover dari klaster DB *blue* ke *green* dibatalkan dikarenakan *reason*.	

Kategori	ID peristiwa Amazon RDS	Pesan	Catatan
pemberitahuan	RDS-EVENT-0311	Sinkronisasi urutan untuk switchover klaster DB *blue* ke *green* telah dimulai. Switchover saat menggunakan urutan dapat menyebabkan waktu henti yang diperpanjang.	
pemberitahuan	RDS-EVENT-0312	Sinkronisasi urutan untuk switchover klaster DB *blue* ke *green* telah selesai.	
kegagalan	RDS-EVENT-0314	Sinkronisasi urutan untuk switchover klaster DB *blue* ke *green* telah dibatalkan karena urutan gagal disinkronisasi.	

Memantau file log Amazon Aurora

Setiap mesin basis data RDS menghasilkan log yang dapat diakses untuk audit dan pemecahan masalah. Jenis log ini bergantung pada mesin basis data Anda.

Anda dapat mengakses log basis data menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau Amazon RDS API. Anda tidak dapat menampilkan, melihat, atau mengunduh log transaksi.

> **Note**
>
> Dalam beberapa kasus, log berisi data tersembunyi. Oleh karena itu, AWS Management Console mungkin menampilkan konten dalam file log, tetapi file log mungkin kosong saat Anda mengunduhnya.

Topik

- Melihat dan mencantumkan file log basis data
- Mengunduh file log basis data
- Melihat file log basis data
- Menerbitkan log basis data ke Log Amazon CloudWatch
- Membaca isi file log dengan menggunakan REST
- File log basis data Aurora MySQL
- File log basis data Aurora PostgreSQL

Melihat dan mencantumkan file log basis data

Anda dapat melihat file log basis data untuk mesin DB Amazon Aurora Anda dengan menggunakan AWS Management Console. Anda dapat mencantumkan file log apa yang dapat diunduh atau dipantau dengan menggunakan AWS CLI atau Amazon RDS API.

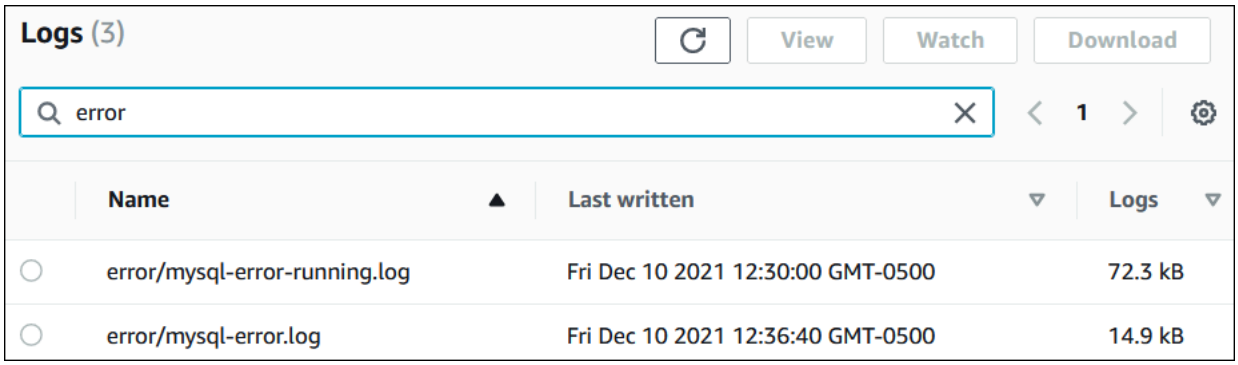
> **Note**
>
> Anda tidak dapat melihat file log untuk klaster DB Aurora Serverless v1 di konsol RDS. Namun, Anda dapat melihatnya di konsol Amazon CloudWatch di https://console.aws.amazon.com/cloudwatch/.

Konsol

Untuk melihat file log basis data

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih nama instans DB yang memiliki file log yang ingin dilihat.
4. Pilih tab Log & peristiwa.
5. Gulir ke bawah ke bagian Log.
6. (Opsional) Masukkan istilah pencarian untuk memfilter hasil.

   Contoh berikut mencantumkan log yang difilter dengan teks **`error`**.



7. Pilih log yang ingin dilihat, lalu pilih Lihat.

AWS CLI

Untuk mencantumkan file log basis data yang tersedia untuk instans DB, gunakan perintah AWS CLI `describe-db-log-files`.

Contoh berikut menampilkan daftar file log untuk instans DB yang bernama `my-db-instance`.

Example

```
aws rds describe-db-log-files --db-instance-identifier my-db-instance
```

RDS API

Untuk mencantumkan file log basis data yang tersedia untuk instans DB, gunakan tindakan `DescribeDBLogFiles` Amazon RDS API.

Mengunduh file log basis data

Anda dapat menggunakan AWS Management Console, AWS CLI atau API untuk mengunduh file log basis data.

Konsol

Untuk mengunduh file log basis data

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih nama instans DB yang memiliki file log yang ingin dilihat.
4. Pilih tab Log & peristiwa.
5. Gulir ke bawah ke bagian Log.
6. Di bagian Log pilih tombol di samping log yang ingin diunduh, lalu pilih Unduh.
7. Buka menu konteks (klik kanan) untuk tautan yang diberikan, lalu pilih Simpan Tautan Sebagai. Masukkan lokasi tempat file log ingin disimpan, lalu pilih Simpan.

AWS CLI

Untuk mengunduh file log basis data, gunakan perintah AWS CLI `download-db-log-file-portion`. Secara default, perintah ini hanya mengunduh bagian terbaru dari file log. Namun, Anda dapat mengunduh seluruh file dengan menentukan parameter `--starting-token 0`.

Contoh berikut menunjukkan cara mengunduh seluruh konten file log yang disebut log/ERROR.4 dan menyimpannya di dalam file lokal yang disebut errorlog.txt.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds download-db-log-file-portion \
    --db-instance-identifier myexampledb \
    --starting-token 0 --output text \
    --log-file-name log/ERROR.4 > errorlog.txt
```

Untuk Windows:

```
aws rds download-db-log-file-portion ^
    --db-instance-identifier myexampledb ^
    --starting-token 0 --output text ^
    --log-file-name log/ERROR.4 > errorlog.txt
```

RDS API

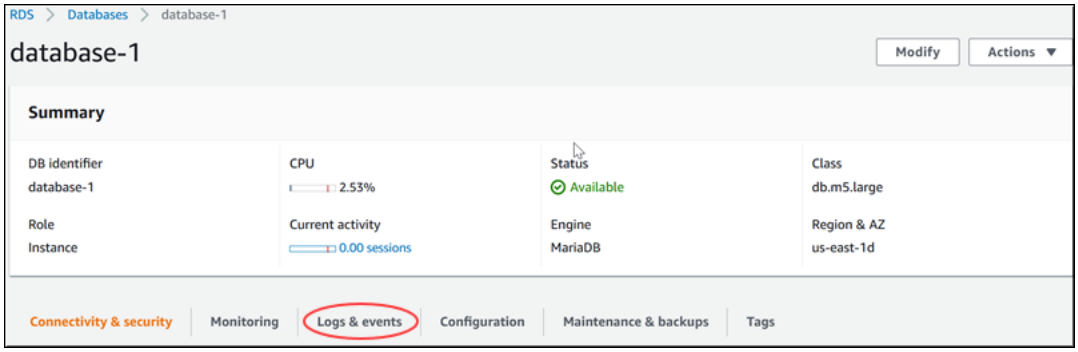
Untuk mengunduh file log basis data, gunakan tindakan `DownloadDBLogFilePortion` Amazon RDS API.

Melihat file log basis data

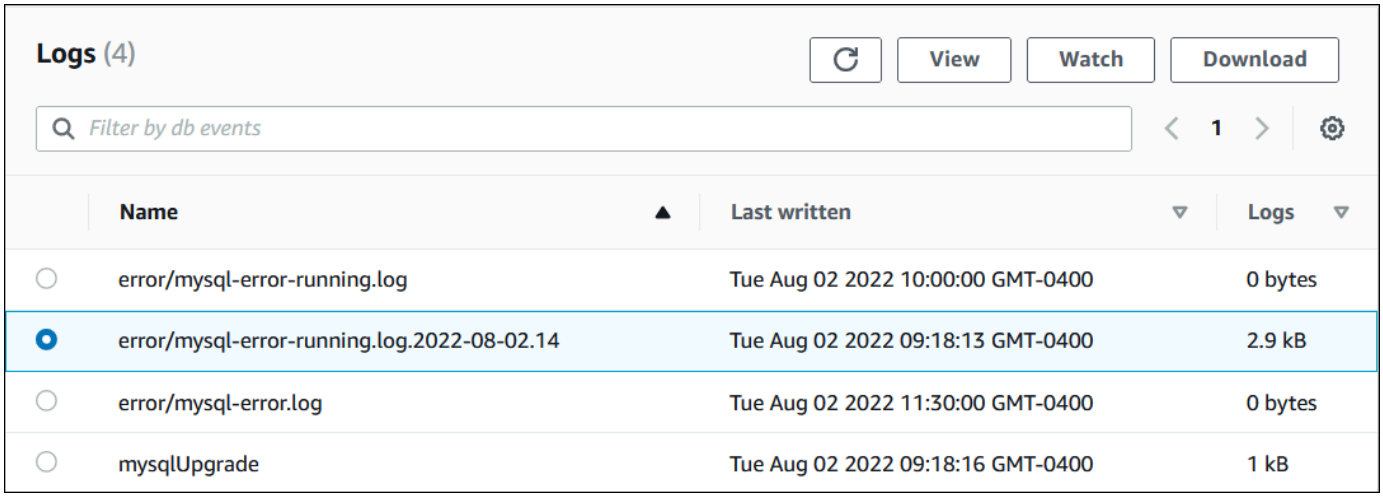
Melihat file log basis data sama seperti membuntuti file pada sistem UNIX atau Linux. Anda dapat melihat file log dengan menggunakan AWS Management Console. RDS menyegarkan ekor log setiap 5 detik.

Untuk melihat file log basis data

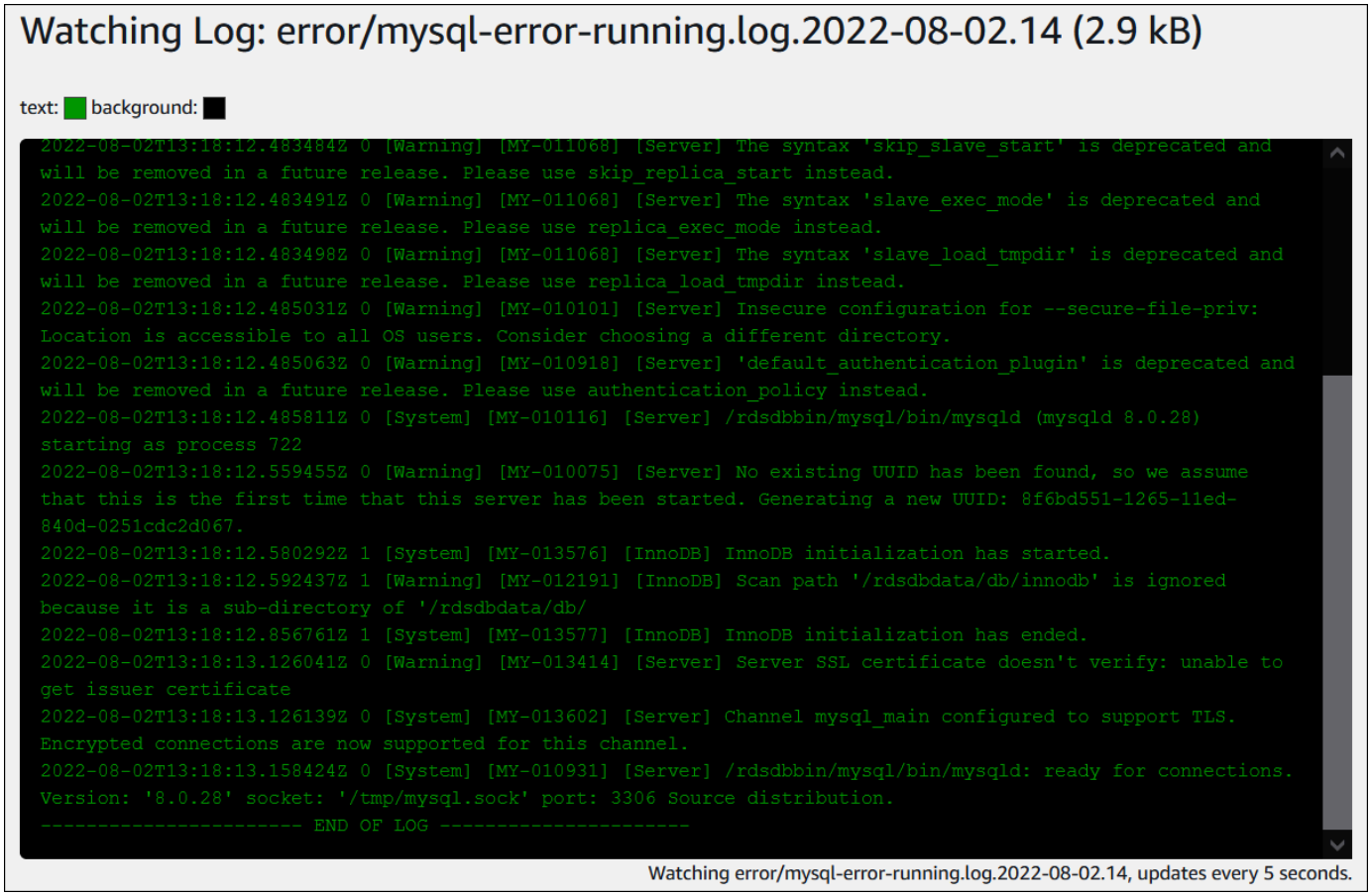
1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih nama instans DB yang memiliki file log yang ingin dilihat.
4. Pilih tab Log & peristiwa.



5. Di bagian Log, pilih file log, lalu pilih Lihat.



RDS menunjukkan ekor log, seperti pada contoh MySQL berikut.



Menerbitkan log basis data ke Log Amazon CloudWatch

Dalam basis data on-premise, log basis data berada pada sistem file. Amazon RDS tidak menyediakan akses host ke log basis data pada sistem file klaster DB Anda. Untuk karena itu, Amazon RDS mengizinkan Anda mengekspor log basis data ke Log Amazon CloudWatch. Dengan Log CloudWatch, Anda dapat melakukan analisis data log secara real-time. Anda dapat menyimpan data dalam penyimpanan yang sangat tahan lama dan mengelola data dengan Agen Log CloudWatch.

Topik

- Ikhtisar integrasi RDS dengan Log CloudWatch
- Memutuskan log yang akan diterbitkan ke Log CloudWatch
- Menentukan log yang akan diterbitkan ke Log CloudWatch
- Mencari dan memfilter log Anda di Log CloudWatch

Ikhtisar integrasi RDS dengan Log CloudWatch

Di Log CloudWatch, log stream adalah urutan peristiwa log yang berbagi sumber yang sama. Setiap sumber log terpisah di Log CloudWatch membentuk log stream terpisah. Grup log adalah grup log stream yang berbagi pengaturan retensi, pemantauan, dan kontrol akses yang sama.

Amazon Aurora terus mengalirkan data log klaster DB Anda ke grup log. Misalnya, Anda memiliki grup log `/aws/rds/cluster/`*`cluster_name`*`/`*`log_type`* untuk setiap jenis log yang Anda terbitkan. Grup log ini berada di Wilayah AWS yang sama dengan instans basis data yang menghasilkan log.

AWS mempertahankan data log yang diterbitkan ke Log CloudWatch selama untuk periode waktu yang tidak ditentukan kecuali Anda menentukan periode retensi. Untuk informasi selengkapnya, lihat Mengubah retensi data log di Log CloudWatch.

Memutuskan log yang akan diterbitkan ke Log CloudWatch

Setiap mesin basis data RDS mendukung kumpulan log-nya sendiri. Untuk mempelajari opsi untuk mesin basis data Anda, baca topik berikut:

- the section called “Menerbitkan log Aurora MySQL ke Log CloudWatch ”
- the section called “Menerbitkan log PostgreSQL Aurora ke Log CloudWatch ”

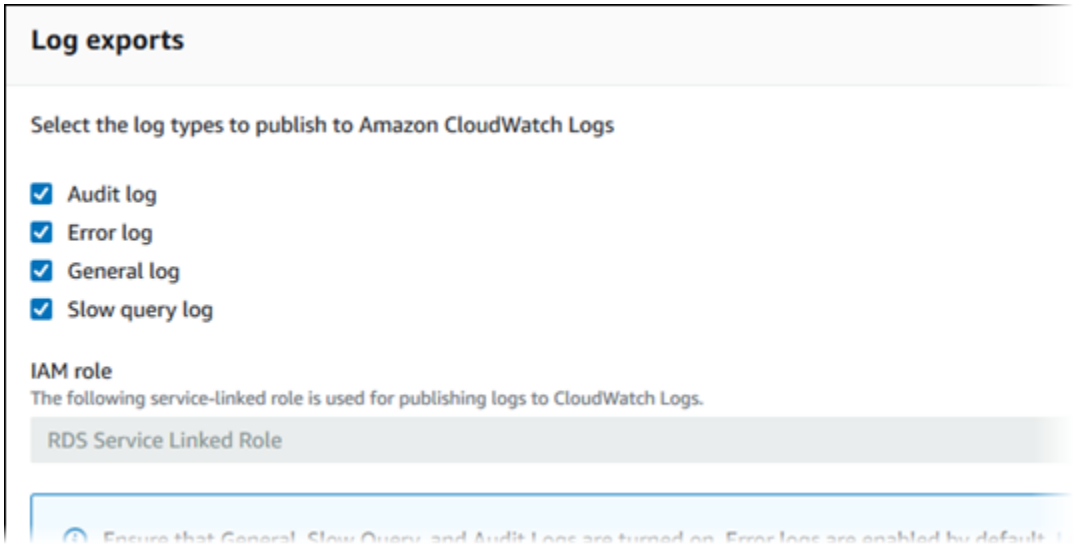
Menentukan log yang akan diterbitkan ke Log CloudWatch

Tentukan log yang akan diterbitkan di konsol. Pastikan Anda memiliki peran terkait layanan di AWS Identity and Access Management (IAM). Untuk informasi selengkapnya tentang peran terkait layanan, lihat Menggunakan peran tertaut layanan untuk Amazon Aurora.

Untuk menentukan log yang akan diterbitkan

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Lakukan salah satu dari langkah berikut:
   - Pilih Buat basis data.
   - Pilih basis data dari daftar, lalu pilih Ubah.
4. Di Ekspor Logs, pilih log yang akan diterbitkan.

Contoh berikut menentukan log audit, log kesalahan, log umum, dan log kueri lambat.



Mencari dan memfilter log Anda di Log CloudWatch

Anda dapat mencari entri log yang memenuhi kriteria tertentu menggunakan konsol Log CloudWatch. Anda dapat mengakses log baik melalui konsol RDS, yang mengarahkan Anda ke konsol Log CloudWatch, atau dari konsol Log CloudWatch secara langsung.

Untuk mencari log RDS menggunakan konsol RDS

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih klaster DB atau instans DB.
4. Pilih Konfigurasi.
5. Di bagian bawah Log yang diterbitkan, pilih log basis data yang ingin dilihat.

Untuk mencari log RDS menggunakan konsol Log CloudWatch

1. Buka konsol CloudWatch di https://console.aws.amazon.com/cloudwatch/.
2. Di panel navigasi, pilih Grup log.
3. Di kotak filter, masukkan **`/aws/rds`**.
4. Untuk Grup Log, pilih nama grup log yang berisi log stream yang akan dicari.
5. Untuk Log Stream, pilih nama log stream yang akan dicari.

6. Di bagian Peristiwa log, masukkan sintaks filter yang akan digunakan.

Untuk informasi selengkapnya, lihat Mencari dan memfilter data log di Panduan Pengguna Log Amazon CloudWatch. Untuk tutorial yang menjelaskan cara memantau log RDS, lihat Membangun pemantauan basis data proaktif untuk Amazon RDS dengan Log Amazon CloudWatch, AWS Lambda, dan Amazon SNS.

Membaca isi file log dengan menggunakan REST

Amazon RDS menyediakan titik akhir REST yang memungkinkan akses ke file log instans DB. Ini berguna jika Anda perlu menulis sebuah aplikasi untuk mengalirkan isi file log Amazon RDS.

Sintaksnya adalah:

```
GET /v13/downloadCompleteLogFile/DBInstanceIdentifier/LogFileName HTTP/1.1
Content-type: application/json
host: rds.region.amazonaws.com
```

Parameter berikut yang diperlukan:

- *DBInstanceIdentifier*—nama instans DB yang berisi file log yang ingin Anda unduh.
- *LogFileName*—nama file log yang akan diunduh.

Respons akan mengandung isi file log yang diminta, berupa sebuah aliran.

Contoh berikut mengunduh file log dengan nama log/ERROR.6 untuk instans DB yang bernama sampel-sql di wilayah us-west-2.

```
GET /v13/downloadCompleteLogFile/sample-sql/log/ERROR.6 HTTP/1.1
host: rds.us-west-2.amazonaws.com
X-Amz-Security-Token: AQoDYXdzEIH//////////
wEa0AIXLhngC5zp9CyB1R6abwKrXHVR5efnAVN3XvR7IwqKYalFSn6UyJuEFTft9nObglx4QJ+GXV9cpACkETq=
X-Amz-Date: 20140903T233749Z
X-Amz-Algorithm: AWS4-HMAC-SHA256
X-Amz-Credential: AKIADQKE4SARGYLE/20140903/us-west-2/rds/aws4_request
X-Amz-SignedHeaders: host
X-Amz-Content-SHA256: e3b0c44298fc1c229afbf4c8996fb92427ae41e4649b934de495991b7852b855
X-Amz-Expires: 86400
X-Amz-Signature: 353a4f14b3f250142d9afc34f9f9948154d46ce7d4ec091d0cdabbcf8b40c558
```

Jika Anda menentukan suatu instans DB yang tidak ada, respons akan terdiri atas kesalahan berikut:

- `DBInstanceNotFound`—*`DBInstanceIdentifier`* tidak mengacu ke instans DB yang ada. (Kode status HTTP: 404)

File log basis data Aurora MySQL

Anda dapat memantau log Aurora MySQL secara langsung melalui konsol Amazon RDS, Amazon RDS API, AWS CLI, atau AWS SDK. Anda juga dapat mengakses log MySQL dengan mengarahkan log ke tabel basis data di basis data utama dan mengkueri tabel tersebut. Anda dapat menggunakan utilitas mysqlbinlog untuk mengunduh log biner.

Untuk informasi selengkapnya tentang melihat, mengunduh, dan melihat log basis data berbasis file, lihat Memantau file log Amazon Aurora.

Topik

- Ikhtisar log basis data Aurora MySQL
- Menerbitkan log Aurora MySQL ke Log Amazon CloudWatch
- Mengelola log Aurora MySQL berbasis tabel
- Mengkonfigurasi pengelogan biner Aurora MySQL
- Mengakses log biner MySQL

Ikhtisar log basis data Aurora MySQL

Anda dapat memantau jenis file log Aurora MySQL berikut:

- Log kesalahan
- Log kueri lambat
- Log umum
- Log audit

Log kesalahan Aurora MySQL dihasilkan secara default. Anda dapat membuat kueri lambat dan log umum dengan mengatur parameter di grup parameter DB Anda.

Topik

- Log kesalahan Aurora MySQL
- Log umum dan kueri lambat Aurora MySQL
- Log audit Aurora MySQL
- Rotasi dan retensi log untuk Aurora MySQL

Log kesalahan Aurora MySQL

Kesalahan tulis Aurora MySQL dalam file `mysql-error.log`. Setiap file log memiliki jam pembuatan (dalam UTC) yang ditambahkan pada namanya. File log juga memiliki stempel waktu yang membantu Anda menentukan kapan entri log ditulis.

Aurora MySQL ditulis ke log kesalahan hanya saat dinyalakan, dimatikan, dan saat terjadi kesalahan. Instans DB dapat memakan waktu berjam-jam atau berhari-hari tanpa perlu menulis entri baru ke log kesalahan. Jika Anda melihat tidak ada entri terbaru, berarti server tidak mengalami kesalahan yang akan mengakibatkan entri log.

Secara desain, log kesalahan difilter sehingga hanya peristiwa tak terduga seperti kesalahan yang ditampilkan. Namun, log kesalahan juga berisi beberapa informasi basis data tambahan, misalnya kemajuan kueri, yang tidak ditampilkan. Oleh karena itu, bahkan tanpa kesalahan aktual, ukuran log kesalahan mungkin meningkat dikarenakan aktivitas basis data yang sedang berlangsung. Dan meskipun Anda mungkin melihat ukuran tertentu dalam byte atau kilobyte untuk log kesalahan di AWS Management Console, log tersebut mungkin memiliki 0 byte saat Anda mengunduhnya.

Aurora MySQL menulis `mysql-error.log` ke disk setiap 5 menit. Ini menambahkan konten log ke`mysql-error-running.log`.

Aurora MySQL merotasi file `mysql-error-running.log` setiap jam.

> **Note**
>
> Periode retensi log berbeda antara Amazon RDS dan Aurora.

Log umum dan kueri lambat Aurora MySQL

Anda dapat menulis log umum dan log kueri lambat Aurora MySQL ke file atau tabel basis data. Untuk melakukannya, atur parameter di grup parameter DB Anda. Untuk informasi tentang pembuatan dan modifikasi grup parameter DB, lihat Bekerja dengan grup parameter. Anda harus mengatur parameter ini sebelum dapat melihat log kueri lambat atau log umum di konsol Amazon RDS atau dengan menggunakan Amazon RDS API, Amazon RDS CLI, atau AWS SDK.

Anda dapat mengontrol pengelogan Aurora MySQL dengan menggunakan parameter dalam daftar ini:

- `slow_query_log`: Untuk membuat log kueri lambat, atur ke 1. Nilai default-nya adalah 0.

- `general_log`: Untuk membuat log umum, atur ke 1. Nilai default-nya adalah 0.
- `long_query_time`: Untuk mencegah kueri yang berjalan cepat masuk ke log kueri lambat, tentukan nilai untuk runtime kueri terpendek yang akan dicatat, dalam detik. Nilai default-nya adalah 10 detik; nilai minimumnya adalah 0. Jika log_output = FILE, Anda dapat menentukan nilai titik mengambang yang masuk ke resolusi mikrodetik. Jika log_output = TABLE, Anda harus menentukan nilai integer dengan resolusi kedua. Hanya kueri yang runtime-nya melampaui nilai `long_query_time` yang akan dicatat. Misalnya, mengatur `long_query_time` ke 0,1 akan mencegah pengelogan kueri apa pun yang berjalan kurang dari 100 milidetik.
- `log_queries_not_using_indexes`: Untuk mencatat semua kueri yang tidak menggunakan indeks pada log kueri lambat, atur ke 1. Kueri yang tidak menggunakan indeks dicatat meskipun runtime-nya kurang dari nilai parameter `long_query_time`. Nilai default-nya adalah 0.
- `log_output` *`option`*: Anda dapat menentukan salah satu opsi berikut untuk parameter `log_output`.
  - TABLE – Menulis kueri umum ke tabel `mysql.general_log`, dan kueri lambat ke tabel `mysql.slow_log`.
  - FILE – Menulis log umum dan log kueri lambat ke sistem file.
  - NONE – Menonaktifkan pengelogan.

  Untuk Aurora MySQL versi 2, nilai default untuk `log_output` adalah `FILE`.

Untuk informasi selengkapnya tentang log umum dan kueri lambat, buka topik berikut di dokumentasi MySQL:

- Log kueri lambat
- Log kueri umum

Log audit Aurora MySQL

Pengelogan audit untuk Aurora MySQL disebut Audit Lanjutan. Untuk mengaktifkan Audit Lanjutan, tetapkan parameter klaster DB tertentu. Untuk informasi selengkapnya, lihat Menggunakan Audit Lanjutan dengan klaster DB Amazon Aurora MySQL.

Rotasi dan retensi log untuk Aurora MySQL

Saat pengelogan diaktifkan, Amazon Aurora merotasi atau menghapus file log secara berkala. Langkah ini merupakan tindakan pencegahan untuk mengurangi kemungkinan file log besar

memblokir penggunaan basis data atau memengaruhi performa. Aurora MySQL menangani rotasi dan penghapusan sebagai berikut:

- Ukuran file log kesalahan Aurora MySQL dibatasi hingga tidak lebih dari 15 persen dari penyimpanan lokal untuk instans DB. Untuk mempertahankan ambang batas ini, log secara otomatis dirotasi setiap jam. Aurora MySQL menghapus log setelah 30 hari atau ketika 15% dari ruang disk tercapai. Jika ukuran file log gabungan melebihi ambang batas setelah file log lama dihapus, file log paling lama akan dihapus hingga ukuran file log tidak lagi melebihi ambang batas.
- Aurora MySQL menghapus log audit, umum, dan kueri lambat setelah 24 jam atau ketika 15% dari penyimpanan telah terpakai.
- Saat pengelogan `FILE` logging, file log umum dan kueri lambat akan diperiksa setiap jam dan file log yang berusia lebih dari 24 jam akan dihapus. Dalam beberapa kasus, ukuran file log gabungan yang tersisa setelah penghapusan mungkin melebihi ambang batas 15 persen dari ruang lokal instans DB. Dalam kasus ini, file log yang paling lama akan dihapus hingga ukuran file log tidak lagi melebihi ambang batas.
- Saat pengelogan `TABLE` diaktifkan, tabel log tidak dirotasi atau dihapus. Tabel log akan dipotong jika ukuran semua log yang digabungkan terlalu besar. Anda dapat berlangganan peristiwa `low_free_storage` yang akan diberitahukan saat tabel log harus dirotasi atau dihapus secara manual untuk mengosongkan ruang. Untuk informasi selengkapnya, lihat Bekerja dengan pemberitahuan peristiwa Amazon RDS.

  Anda dapat merotasi tabel `mysql.general_log` secara manual dengan memanggil prosedur `mysql.rds_rotate_general_log`. Anda dapat merotasi tabel `mysql.slow_log` dengan mengikuti prosedur `mysql.rds_rotate_slow_log`.

  Saat Anda merotasi tabel log secara manual, tabel log saat ini disalin ke tabel log cadangan dan entri di tabel log saat ini dihapus. Jika sudah ada, tabel log cadangan akan dihapus sebelum tabel log saat ini disalin ke cadangan. Anda dapat meminta tabel log cadangan jika diperlukan. Tabel log cadangan untuk tabel `mysql.general_log` bernama `mysql.general_log_backup`. Tabel log cadangan untuk tabel `mysql.slow_log` bernama `mysql.slow_log_backup`.
- Log audit Aurora MySQL dirotasi saat ukuran file mencapai 100 MB, dan dihapus setelah 24 jam.

Untuk bekerja dengan log dari konsol Amazon RDS, Amazon RDS API, Amazon RDS CLI, atau AWS SDK, atur parameter `log_output` ke FILE. Seperti log kesalahan Aurora MySQL, file log ini dirotasi setiap jam. File log yang dihasilkan selama 24 jam sebelumnya akan dipertahankan. Perhatikan bahwa periode retensi log berbeda antara Amazon RDS dan Aurora.

Menerbitkan log Aurora MySQL ke Log Amazon CloudWatch

Anda dapat mengonfigurasi klaster DB Aurora MySQL Anda untuk menerbitkan data log ke grup log di Log Amazon CloudWatch. Dengan Log CloudWatch, Anda dapat melakukan analisis data log secara real-time, dan menggunakan CloudWatch untuk membuat alarm dan melihat metrik. Anda dapat menggunakan Log CloudWatch untuk menyimpan data log Anda di dalam penyimpanan yang sangat tahan lama. Untuk informasi selengkapnya, lihat Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch .

Mengelola log Aurora MySQL berbasis tabel

Anda dapat mengarahkan log kueri umum dan lambat ke tabel pada instans DB dengan membuat grup parameter DB dan mengatur parameter server `log_output` ke `TABLE`. Kueri umum kemudian dicatat ke tabel `mysql.general_log`, dan kueri lambat dicatat ke tabel `mysql.slow_log`. Anda dapat membuat kueri tabel untuk mengakses informasi log. Mengaktifkan pencatatan ini akan meningkatkan jumlah data yang akan ditulis ke basis data, yang dapat menurunkan performa.

Log kueri umum dan lambat dinonaktifkan secara default. Untuk mengaktifkan pencatatan ke tabel, Anda juga harus mengatur parameter server `general_log` dan `slow_query_log` ke `1`.

Tabel log terus bertambah hingga aktivitas pencatatan masing-masing dinonaktifkan dengan mengatur ulang parameter yang sesuai ke `0`. Banyak data sering terakumulasi seiring berjalannya waktu, yang dapat menghabiskan cukup banyak ruang penyimpanan yang dialokasikan. Amazon Aurora tidak mengizinkan Anda memotong tabel log, tetapi Anda dapat memindahkan kontennya. Merotasi tabel akan menyimpan kontennya ke tabel cadangan, lalu membuat tabel log kosong yang baru. Anda dapat merotasi tabel log secara manual dengan mengikuti prosedur baris perintah berikut, yang perintahnya ditunjukkan oleh `PROMPT>`:

```
PROMPT> CALL mysql.rds_rotate_slow_log;
PROMPT> CALL mysql.rds_rotate_general_log;
```

Untuk menghapus data lama sepenuhnya dan mengosongkan kembali ruang disk, panggil prosedur yang sesuai dua kali secara berurutan.

Mengkonfigurasi pengelogan biner Aurora MySQL

Log biner adalah sekumpulan file log yang berisi informasi tentang modifikasi data yang dibuat ke instans server Aurora MySQL. Log biner berisi informasi seperti berikut:

- Peristiwa yang menggambarkan perubahan basis data seperti pembuatan tabel atau modifikasi baris
- Informasi tentang durasi setiap pernyataan yang memperbarui data
- Peristiwa untuk pernyataan yang bisa saja memperbarui data, tetapi tidak

Log biner mencatat pernyataan yang dikirim selama replikasi. Log ini juga diperlukan untuk beberapa operasi pemulihan. Untuk informasi selengkapnya, lihat Log Biner dan Ikhtisar Log Biner dalam dokumentasi MySQL.

Log biner hanya dapat diakses dari instans DB primer, bukan dari replika.

MySQL di Amazon Aurora mendukung format pengelogan biner berbasis baris, berbasis pernyataan, dan campuran. Kami merekomendasikan campuran kecuali Anda memerlukan format binlog tertentu. Untuk detail tentang format log biner Aurora MySQL lainnya, lihat Format pengelogan biner dalam dokumentasi MySQL.

Jika Anda berencana menggunakan replikasi, format pengelogan biner diperlukan karena menentukan catatan perubahan data yang dicatat di sumber dan dikirim ke target replikasi. Untuk informasi tentang kelebihan dan kelemahan format pengelogan biner lainnya untuk replikasi, lihat Kelebihan dan kelemahan replikasi berbasis pernyataan dan berbasis baris dalam dokumentasi MySQL.

> **Important**
>
> Mengatur format pengelogan biner ke berbasis baris dapat menghasilkan file log biner yang sangat besar. File log biner besar mengurangi jumlah penyimpanan yang tersedia untuk instans DB dan dapat meningkatkan jumlah waktu yang dibutuhkan untuk melakukan operasi pemulihan instans DB.
>
> Replikasi berbasis pernyataan dapat menyebabkan inkonsistensi antara instans DB dan replika baca. Untuk informasi selengkapnya, lihat Penentuan pernyataan yang aman dan tidak aman dalam pengelogan biner di dokumentasi MySQL.
>
> Mengaktifkan pengelogan biner akan meningkatkan jumlah operasi I/O disk tulis untuk instans DB. Anda dapat memantau penggunaan IOPS dengan `VolumeWriteIOPs` CloudWatch metrik.

Untuk mengatur format pengelogan biner MySQL

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Pilih grup parameter klaster DB, yang terkait dengan klaster DB, yang ingin dimodifikasi.

   Anda tidak dapat mengubah grup parameter default. Jika instans DB menggunakan grup parameter default, buat grup parameter baru dan hubungkan dengan instans DB.

   Untuk informasi selengkapnya tentang grup parameter, lihat Bekerja dengan grup parameter.
4. Dari Tindakan, pilih Edit.
5. Atur parameter `binlog_format` ke format pengelogan biner pilihan Anda (`ROW`, `STATEMENT`, atau `MIXED`). Anda juga dapat menggunakan nilai `OFF` untuk menonaktifkan pengelogan biner.

   > **Note**
   >
   > Pengaturan `binlog_format` ke `OFF` dalam kelompok parameter cluster DB menonaktifkan variabel `log_bin` sesi. Ini menonaktifkan logging biner pada cluster DB MySQL Aurora, yang pada gilirannya `binlog_format` mengatur ulang variabel sesi ke nilai default dalam database. `ROW`
6. Pilih Simpan perubahan untuk menyimpan pembaruan ke grup parameter klaster DB.

Setelah melakukan langkah-langkah ini, Anda harus me-reboot instans penulis di klaster DB untuk menerapkan perubahan. Dalam Aurora MySQL versi 2.09 dan yang lebih rendah, saat Anda me-reboot instans penulis, semua instans pembaca di klaster DB juga akan di-reboot. Dalam Aurora MySQL versi 2.10 dan yang lebih baru, Anda harus me-reboot semua instans pembaca secara manual. Untuk informasi selengkapnya, lihat Mem-boot ulang klaster DB Amazon Aurora atau instans DB Amazon Aurora.

> **Important**
>
> Mengubah grup parameter klaster DB memengaruhi semua klaster DB yang menggunakan grup parameter tersebut. Jika Anda ingin menentukan format pengelogan biner lainnya untuk klaster DB Aurora MySQL yang berbeda di Wilayah AWS, klaster DB harus menggunakan grup parameter klaster DB yang berbeda. Grup parameter ini mengidentifikasi format pengelogan yang berbeda. Tetapkan grup parameter klaster DB yang sesuai ke masing-

masing klaster DB. Untuk informasi selengkapnya tentang parameter Aurora MySQL, lihat Parameter konfigurasi Aurora MySQL.

Mengakses log biner MySQL

Anda dapat menggunakan utilitas mysqlbinlog untuk mengunduh atau mengalirkan log biner dari instans DB RDS untuk MySQL. Log biner diunduh ke komputer lokal, tempat Anda dapat melakukan tindakan seperti memutar ulang log menggunakan utilitas mysql. Untuk informasi selengkapnya tentang cara menggunakan utilitas mysqlbinlog, lihat Menggunakan mysqlbinlog untuk mencadangkan file log biner dalam dokumentasi MySQL.

Untuk menjalankan utilitas mysqlbinlog terhadap instans Amazon RDS, gunakan opsi berikut:

- `--read-from-remote-server` – Wajib diisi.
- `--host` – Nama DNS dari titik akhir instans.
- `--port` – Port yang digunakan oleh instans.
- `--user` – Pengguna MySQL yang telah diberi izin `REPLICATION SLAVE`.
- `--password` – Kata sandi untuk pengguna MySQL, atau hapus nilai kata sandi agar utilitas meminta Anda memasukkan kata sandi.
- `--raw` – Mengunduh file dalam format biner.
- `--result-file` – File lokal untuk menerima output mentah.
- `--stop-never` – Mengalirkan file log biner.
- `--verbose` – Jika Anda menggunakan format binlog `ROW`, sertakan opsi ini untuk melihat peristiwa baris sebagai pernyataan pseudo-SQL. Untuk informasi selengkapnya tentang opsi `--verbose`, lihat tampilan peristiwa baris mysqlbinlog dalam dokumentasi MySQL.
- Tentukan nama satu atau beberapa file log biner. Untuk mendapatkan daftar log yang tersedia, gunakan perintah SQL `SHOW BINARY LOGS`.

Untuk informasi selengkapnya tentang opsi mysqlbinlog, lihat mysqlbinlog — untuk memperoses file log biner dalam dokumentasi MySQL.

Contoh berikut menunjukkan cara menggunakan utilitas mysqlbinlog.

Untuk Linux, macOS, atau Unix:

```
mysqlbinlog \
    --read-from-remote-server \
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com \
    --port=3306  \
    --user ReplUser \
    --password \
    --raw \
    --verbose \
    --result-file=/tmp/ \
    binlog.00098
```

Untuk Windows:

```
mysqlbinlog ^
    --read-from-remote-server ^
    --host=MySQLInstance1.cg034hpkmmjt.region.rds.amazonaws.com ^
    --port=3306  ^
    --user ReplUser ^
    --password ^
    --raw ^
    --verbose ^
    --result-file=/tmp/ ^
    binlog.00098
```

Amazon RDS biasanya membersihkan log biner sesegera mungkin, tetapi log biner ini harus tetap tersedia di instans untuk diakses oleh mysqlbinlog. Untuk menentukan jumlah jam yang dibutuhkan RDS untuk mempertahankan log biner, gunakan prosedur tersimpan mysql.rds_set_configuration dan tentukan periode yang cukup agar Anda dapat mengunduh log. Setelah Anda mengatur periode retensi, pantau penggunaan penyimpanan untuk instans DB guna memastikan bahwa log biner yang dipertahankan tidak memakan terlalu banyak ruang penyimpanan.

Contoh berikut menetapkan periode retensi ke 1 hari.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

Untuk menampilkan pengaturan saat ini, gunakan prosedur tersimpan mysql.rds_show_configuration.

```
call mysql.rds_show_configuration;
```

File log basis data Aurora PostgreSQL

Aurora PostgreSQL mencatat aktivitas basis data ke file log PostgreSQL default. Untuk instans DB PostgreSQL on-premise, pesan ini disimpan secara lokal di `log/postgresql.log`. Untuk klaster DB Aurora PostgreSQL, file log tersedia di klaster Aurora. Selain itu, Anda harus menggunakan Konsol Amazon RDS untuk melihat atau mengunduh kontennya. Tingkat penglogan default menangkap kegagalan masuk, kesalahan server fatal, kebuntuan, dan kegagalan kueri.

Untuk informasi selengkapnya tentang cara menampilkan, mengunduh, dan melihat log basis data berbasis file, lihat Memantau file log Amazon Aurora. Untuk mempelajari selengkapnya tentang log PostgreSQL, lihat Menggunakan log Amazon RDS dan Aurora PostgreSQL: Bagian 1 dan Menggunakan log Amazon RDS dan Aurora PostgreSQL: Bagian 2.

Selain log PostgreSQL standar yang dibahas dalam topik ini, Aurora PostgreSQL juga mendukung ekstensi PostgreSQL Audit (`pgAudit`). Sebagian besar industri dan lembaga pemerintah yang teregulasi perlu mempertahankan log audit atau jejak audit perubahan yang dibuat pada data untuk memenuhi persyaratan hukum. Untuk informasi tentang cara menginstal dan menggunakan pgAudit, lihat Menggunakan pgAudit untuk membuat log aktivitas basis data.

Topik

- Parameter yang memengaruhi perilaku pengelogan
- Mengaktifkan pengelogan kueri untuk klaster DB Aurora PostgreSQL

Parameter yang memengaruhi perilaku pengelogan

Anda dapat menyesuaikan perilaku pengelogan untuk klaster DB Aurora PostgreSQL dengan mengubah berbagai parameter. Dalam tabel berikut, Anda dapat menemukan parameter yang memengaruhi durasi penyimpanan log, waktu untuk memutar log, dan apakah akan menampilkan log sebagai format CSV (nilai yang dipisahkan koma). Anda juga dapat menemukan output teks yang dikirim ke STDERR, di antara pengaturan lainnya. Untuk mengubah pengaturan parameter yang dapat dimodifikasi, gunakan grup parameter klaster DB kustom untuk klaster DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Bekerja dengan grup parameter. Seperti disebutkan dalam tabel, `log_line_prefix` tidak dapat diubah.

Parameter	Default	Deskripsi
log_destination	stderr	Menetapkan format output untuk log. Defaultnya adalah `stderr`, tetapi Anda juga dapat

Parameter	Default	Deskripsi
		menentukan nilai dipisahkan koma (CSV) dengan menambahkan `csvlog` ke pengaturan. Lihat informasi yang lebih lengkap di Mengatur tujuan log (`stderr, csvlog`)
log_filename	postgresql.log.%Y-%m-%d-%H%M	Menentukan pola untuk nama file log. Selain default, parameter ini mendukung `postgresql.log.%Y-%m-%d` dan `postgresql.log.%Y-%m-%d-%H` untuk pola nama file.
log_line_prefix	%t:%r:%u@%d:[%p]:	Mendefinisikan awalan untuk setiap baris log yang akan ditulis ke `stderr`, untuk mencatat waktu (%t), host jarak jauh (%r), pengguna (%u), basis data (%d), dan ID proses (%p). Anda tidak dapat mengubah parameter ini.
log_rotation_age	60	Menit setelah file log dirotasi secara otomatis. Anda dapat mengubah nilai ini menjadi antara 1 dan 1.440 menit. Untuk informasi selengkapnya, lihat Mengatur rotasi file log.
log_rotation_size	–	Ukuran (kB) di mana log dirotasi secara otomatis. Secara default, parameter ini tidak digunakan karena log dirotasi berdasarkan parameter `log_rotation_age` . Untuk mempelajari selengkapnya, lihat Mengatur rotasi file log.
rds.log_retention_period	4320	Log PostgreSQL yang lebih lama dari jumlah menit yang ditentukan dihapus. Nilai default 4320 menit menghapus file log setelah 3 hari. Untuk informasi selengkapnya, lihat Mengatur periode retensi log.

Untuk mengidentifikasi masalah aplikasi, Anda dapat mencari kegagalan kueri, kegagalan masuk, kebuntuan, dan kesalahan server fatal di log. Misalnya, anggaplah Anda mengonversi aplikasi

lama dari Oracle ke Aurora PostgreSQL, tetapi tidak semua kueri dikonversi dengan benar. Kueri yang salah format ini menghasilkan pesan kesalahan yang dapat Anda temukan di log untuk membantu mengidentifikasi masalah. Untuk informasi selengkapnya tentang pengelogan kueri, lihat Mengaktifkan pengelogan kueri untuk klaster DB Aurora PostgreSQL.

Dalam topik berikut, Anda dapat menemukan informasi tentang cara mengatur berbagai parameter yang mengontrol detail dasar untuk log PostgreSQL Anda.

Topik

- Mengatur periode retensi log
- Mengatur rotasi file log
- Mengatur tujuan log (stderr, csvlog)
- Memahami parameter log_line_prefix

Mengatur periode retensi log

Parameter `rds.log_retention_period` menentukan berapa lama klaster DB Aurora PostgreSQL menyimpan file log-nya. Pengaturan default-nya adalah 3 hari (4.320 menit), tetapi Anda dapat mengatur nilai ini ke mana saja dari 1 hari (1.440 menit) hingga 7 hari (10.080 menit). Pastikan bahwa klaster DB Aurora PostgreSQL Anda memiliki penyimpanan yang cukup untuk menyimpan file log selama periode waktu tertentu.

Kami menyarankan agar log Anda dipublikasikan secara rutin ke Amazon CloudWatch Logs sehingga Anda dapat melihat dan menganalisis data sistem lama setelah log dihapus dari cluster DB PostgreSQL Aurora Anda. Untuk informasi selengkapnya, lihat Menerbitkan log Aurora PostgreSQL ke Amazon Logs CloudWatch . Setelah Anda mengatur CloudWatch penerbitan, Aurora tidak menghapus log sampai setelah diterbitkan ke CloudWatch Log.

Amazon Aurora mengompresi log PostgreSQL lama ketika penyimpanan untuk instans DB mencapai ambang batas. Aurora mengompresi file menggunakan utilitas kompresi gzip. Untuk informasi selengkapnya, lihat situs web gzip.

Saat penyimpanan instans DB rendah dan semua log yang tersedia dikompresi, Anda akan mendapatkan peringatan seperti berikut:

```
Warning: local storage for PostgreSQL log files is critically low for
this Aurora PostgreSQL instance, and could lead to a database outage.
```

Jika tidak ada cukup penyimpanan, Aurora dapat menghapus log PostgreSQL yang terkompresi sebelum akhir periode retensi tertentu. Jika itu terjadi, Anda akan melihat pesan yang mirip dengan pesan berikut:

```
The oldest PostgreSQL log files were deleted due to local storage constraints.
```

Mengatur rotasi file log

Aurora membuat file log baru setiap jam secara default. Waktu dikendalikan oleh parameter `log_rotation_age`. Parameter ini memiliki nilai default 60 (menit), tetapi Anda dapat mengaturnya ke mana saja dari 1 menit hingga 24 jam (1.440 menit). Ketika tiba waktunya rotasi, file log baru yang berbeda akan dibuat. File ini diberi nama sesuai dengan pola yang ditentukan oleh parameter `log_filename`.

File log juga dapat dirotasi sesuai dengan ukurannya, seperti yang ditentukan dalam parameter `log_rotation_size`. Parameter ini menentukan bahwa log harus dirotasi saat mencapai ukuran yang ditentukan (dalam kilobyte). `log_rotation_size` default-nya adalah 100000 kB (kilobyte) untuk klaster DB Aurora PostgreSQL, tetapi Anda dapat mengatur nilai ini ke mana pun dari 50.000 hingga 1.000.000 kilobyte.

Nama file log didasarkan pada pola nama file yang ditentukan dalam parameter `log_filename`. Pengaturan yang tersedia untuk parameter ini adalah sebagai berikut:

- `postgresql.log.%Y-%m-%d` – Format default untuk nama file log. Termasuk tahun, bulan, dan tanggal dalam nama file log.
- `postgresql.log.%Y-%m-%d-%H` – Termasuk jam dalam format nama file log.
- `postgresql.log.%Y-%m-%d-%H%M` – Termasuk jam:menit dalam format nama file log.

Jika Anda mengatur parameter `log_rotation_age` ke kurang dari 60 menit, atur parameter `log_filename` ke format menit.

Untuk informasi selengkapnya, lihat `log_rotation_age` dan `log_rotation_size` dalam dokumentasi PostgreSQL.

Mengatur tujuan log (**`stderr`**, **`csvlog`**)

Secara default, Aurora PostgreSQL menghasilkan log dalam format kesalahan standar (stderr). Format ini adalah pengaturan default untuk parameter `log_destination`. Setiap pesan diawali

menggunakan pola yang ditentukan dalam parameter `log_line_prefix`. Untuk mengetahui informasi selengkapnya, lihat Memahami parameter log_line_prefix.

Aurora PostgreSQL juga dapat menghasilkan log dalam format `csvlog`. `csvlog` berguna untuk menganalisis data log sebagai data nilai yang dipisahkan koma (CSV). Misalnya, anggaplah Anda menggunakan ekstensi `log_fdw` untuk bekerja dengan log Anda sebagai tabel asing. Tabel asing yang dibuat pada file log `stderr` berisi satu kolom dengan data peristiwa log. Dengan menambahkan `csvlog` ke parameter `log_destination`, Anda mendapatkan file log dalam format CSV dengan demarkasi untuk beberapa kolom tabel asing. Anda kini dapat mengurutkan dan menganalisis log dengan lebih mudah.

Jika Anda menentukan `csvlog` untuk parameter ini, perhatikan bahwa file `stderr` dan `csvlog` dihasilkan. Pastikan untuk memantau penyimpanan yang digunakan oleh log, dengan mempertimbangkan `rds.log_retention_period` dan pengaturan lain yang memengaruhi penyimpanan log dan omset. Menggunakan `stderr` dan `csvlog` lebih dari dua kali lipat penyimpanan yang digunakan oleh log.

Jika Anda menambahkan `csvlog` ke `log_destination` dan ingin kembali ke `stderr` sendiri, Anda perlu mengatur ulang parameter. Untuk melakukannya, buka Konsol Amazon RDS, lalu buka grup parameter klaster DB untuk instans Anda. Pilih parameter `log_destination`, pilih Edit parameter, lalu pilih Atur ulang.

Untuk informasi selengkapnya tentang cara mengonfigurasi pengelogan, lihat Menggunakan log Amazon RDS dan Aurora PostgreSQL: Bagian 1.

Memahami parameter log_line_prefix

Format log `stderr` mengawali setiap pesan log dengan detail yang ditentukan oleh parameter `log_line_prefix`, sebagai berikut.

```
%t:%r:%u@%d:[%p]:t
```

Anda dapat mengubah pengaturan ini: Setiap entri log yang dikirim ke `stderr` berisi informasi berikut.

- `%t` – Waktu entri log
- `%r` – Alamat host jarak jauh
- `%u@%d` – Nama pengguna @ nama basis data

- `[%p]` – ID Proses jika tersedia

Mengaktifkan pengelogan kueri untuk klaster DB Aurora PostgreSQL

Anda dapat mengumpulkan informasi yang lebih mendetail tentang aktivitas basis data, termasuk kueri, kueri yang menunggu kunci, titik pemeriksaan, dan banyak detail lainnya dengan mengatur beberapa parameter yang tercantum dalam tabel berikut. Topik ini berfokus pada kueri pengelogan.

Parameter	Default	Deskripsi
log_connections	–	Mencatat setiap koneksi yang berhasil. Untuk mempelajari cara menggunakan parameter ini dengan `log_disconnections` untuk mendeteksi churn koneksi, lihatMengelola churn koneksi Aurora PostgreSQL dengan pooling .
log_disconnections	–	Mencatat akhir setiap sesi dan durasinya . Untuk mempelajari cara menggunak an parameter ini dengan `log_conne ctions` untuk mendeteksi churn koneksi, lihatMengelola churn koneksi Aurora PostgreSQL dengan pooling .
log_checkpoints	1	Mencatat setiap titik pemeriksaan.
log_lock_waits	–	Mencatat waktu tunggu kunci yang panjang. Secara default, parameter ini tidak diatur.
log_min_duration_s ample	–	(md) Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat sampel pernyataan dicatat. Ukuran sampel diatur menggunakan parameter `log_state ment_sample_rate` .
log_min_duration_s tatement	–	Setiap pernyataan SQL yang berjalan setidakny a selama periode waktu tertentu atau lebih lama akan dicatat. Secara default, parameter

Parameter	Default	Deskripsi
		ini tidak diatur. Mengaktifkan parameter ini dapat membantu Anda menemukan kueri yang belum dioptimalkan.
log_statement	–	Menetapkan jenis pernyataan yang dicatat. Secara default, parameter ini tidak diatur, tetapi Anda dapat mengubahnya ke `all`, `ddl`, atau `mod` untuk menentukan jenis pernyataan SQL yang ingin Anda catat. Jika menentukan apa pun selain `none` untuk parameter ini, Anda juga harus mengambil langkah-langkah tambahan untuk mencegah eksposur kata sandi dalam file log. Untuk informasi selengkapnya, lihat Mengurangi risiko eksposur kata sandi saat menggunakan pengelogan kueri.
log_statement_sample_rate	–	Persentase pernyataan melebihi waktu yang ditentukan dalam `log_min_duration_sample` untuk dicatat, yang dinyatakan sebagai nilai titik mengambang antara 0,0 dan 1,0.
log_statement_stats	–	Menulis statistik performa kumulatif ke log server.

Menggunakan pengelogan untuk menemukan kueri performa lambat

Anda dapat mencatat pernyataan dan kueri SQL untuk membantu menemukan kueri performa lambat. Anda mengaktifkan kemampuan ini dengan memodifikasi pengaturan di `log_statement` dan parameter `log_min_duration` seperti yang diuraikan dalam bagian ini. Sebelum mengaktifkan pengelogan kueri untuk klaster DB Aurora PostgreSQL, Anda harus mengetahui kemungkinan eksposur kata sandi di dalam log dan cara mengurangi risiko ini. Untuk informasi selengkapnya, lihat Mengurangi risiko eksposur kata sandi saat menggunakan pengelogan kueri.

Berikut ini, Anda dapat menemukan informasi referensi tentang parameter `log_statement` dan `log_min_duration`.

log_statement

Parameter ini menentukan jenis pernyataan SQL yang harus dikirim ke log. Nilai default-nya adalah `none`. Jika Anda mengubah parameter ini ke `all`, `ddl`, atau `mod`, pastikan untuk menerapkan tindakan yang disarankan untuk mengurangi risiko eksposur kata sandi di dalam log. Untuk informasi selengkapnya, lihat Mengurangi risiko eksposur kata sandi saat menggunakan pengelogan kueri.

all

Mencatat semua pernyataan. Pengaturan ini direkomendasikan untuk tujuan debugging.

ddl

Mencatat semua pernyataan bahasa definisi data (DDL), seperti CREATE, ALTER, DROP, dan seterusnya.

mod

Mencatat semua pernyataan DDL dan pernyataan bahasa manipulasi data (DML), seperti INSERT, UPDATE, dan DELETE, yang mengubah data.

none

Tidak ada pernyataan SQL yang dicatat. Kami merekomendasikan pengaturan ini untuk menghindari risiko eksposur kata sandi di dalam log.

log_min_duration_statement

Setiap pernyataan SQL yang berjalan setidaknya selama periode waktu tertentu atau lebih lama akan dicatat. Secara default, parameter ini tidak diatur. Mengaktifkan parameter ini dapat membantu Anda menemukan kueri yang belum dioptimalkan.

–1–2147483647

Jumlah milidetik (md) runtime di mana pernyataan dicatat.

Untuk menyiapkan pengelogan kueri

Langkah-langkah ini mengasumsikan bahwa klaster DB Aurora PostgreSQL Anda menggunakan grup parameter klaster kustom.

1. Atur parameter `log_statement` ke `all`. Contoh berikut ini menunjukkan informasi yang ditulis ke file `postgresql.log` dengan pengaturan parameter ini.

```
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: statement:
 SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:LOG: QUERY
 STATISTICS
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:DETAIL: ! system
 usage stats:
! 0.017355 s user, 0.000000 s system, 0.168593 s elapsed
! [0.025146 s user, 0.000000 s system total]
! 36644 kB max resident size
! 0/8 [0/8] filesystem blocks in/out
! 0/733 [0/1364] page faults/reclaims, 0 [0] swaps
! 0 [0] signals rcvd, 0/0 [0/0] messages rcvd/sent
! 19/0 [27/0] voluntary/involuntary context switches
2022-10-05 22:05:52 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: SELECT
 feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:ERROR: syntax error
 at or near "ORDER" at character 1
2022-10-05 22:05:56 UTC:52.95.4.1(11335):postgres@labdb:[3639]:STATEMENT: ORDER BY
 s.confidence DESC;
---------------------- END OF LOG ----------------------
```

2. Atur parameter `log_min_duration_statement`. Contoh berikut ini menunjukkan informasi yang ditulis ke file `postgresql.log` saat pengaturan parameter ini diatur ke 1.

   Kueri yang melebihi durasi yang ditentukan dalam parameter `log_min_duration_statement` dicatat. Bagian berikut menunjukkan satu contoh. Anda dapat melihat file log untuk klaster DB Aurora PostgreSQL di Konsol Amazon RDS.

```
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: statement: DROP
 table comments;
2022-10-05 19:05:19 UTC:52.95.4.1(6461):postgres@labdb:[6144]:LOG: duration:
 167.754 ms
2022-10-05 19:08:07 UTC::@:[355]:LOG: checkpoint starting: time
2022-10-05 19:08:08 UTC::@:[355]:LOG: checkpoint complete: wrote 11 buffers
 (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write=1.013 s, sync=0.006 s,
 total=1.033 s; sync files=8, longest=0.004 s, average=0.001 s; distance=131028 kB,
 estimate=131028 kB
```

```
----------------------- END OF LOG ----------------------
```

Mengurangi risiko eksposur kata sandi saat menggunakan pengelogan kueri

Kami menyarankan agar Anda tetap mengatur `log_statement` ke `none` agar tidak mengekspos kata sandi. Jika Anda mengatur `log_statement` ke `all`, `ddl`, atau `mod`, sebaiknya Anda mengambil satu atau beberapa langkah berikut.

- Untuk klien, enkripsi informasi sensitif. Untuk informasi selengkapnya, lihat Opsi Enkripsi dalam dokumentasi PostgreSQL. Gunakan opsi `ENCRYPTED` (dan `UNENCRYPTED`) dari pernyataan `CREATE` dan `ALTER`. Untuk informasi selengkapnya, lihat CREATE USER dalam dokumentasi PostgreSQL.
- Untuk klaster DB Aurora PostgreSQL, siapkan dan gunakan ekstensi PostgreSQL Audit (pgAudit). Ekstensi ini menyunting informasi sensitif dalam pernyataan CREATE dan ALTER yang dikirim ke log. Untuk informasi selengkapnya, lihat Menggunakan pgAudit untuk membuat log aktivitas basis data.
- Batasi akses ke CloudWatch log.
- Gunakan mekanisme autentikasi yang lebih kuat seperti IAM.

Memantau panggilan API Amazon Aurora di AWS CloudTrail

AWS CloudTrail adalah layanan AWS yang membantu Anda mengaudit akun AWS Anda. AWS CloudTrail diaktifkan untuk akun AWS Anda saat Anda membuatnya. Untuk informasi selengkapnya tentang CloudTrail, lihat Panduan Pengguna AWS CloudTrail.

Topik

- Integrasi CloudTrail dengan Amazon Aurora
- Entri file log Amazon Aurora

Integrasi CloudTrail dengan Amazon Aurora

Semua tindakan Amazon Aurora di-log oleh CloudTrail. CloudTrail menyediakan rekam tindakan yang diambil oleh pengguna, peran, atau layanan AWS di Amazon Aurora.

Peristiwa CloudTrail

CloudTrail menangkap panggilan API untuk Amazon Aurora sebagai peristiwa. Peristiwa merepresentasikan satu permintaan dari sumber apa pun dan menyertakan informasi tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya. Peristiwa mencakup panggilan dari konsol Amazon RDS dan dari panggilan kode ke operasi API Amazon RDS.

Aktivitas Amazon Aurora direkam dalam peristiwa CloudTrail di Riwayat peristiwa. Anda dapat menggunakan konsol CloudTrail untuk melihat aktivitas dan peristiwa API yang direkam selama 90 hari terakhir di Wilayah AWS. Untuk informasi selengkapnya, lihat Melihat peristiwa dengan riwayat peristiwa CloudTrail.

Jejak CloudTrail

Untuk catatan berkelanjutan peristiwa di akun AWS Anda, yang meliputi peristiwa untuk Amazon Aurora, buatlah jejak. Sebuah jejak adalah konfigurasi yang memungkinkan pengiriman peristiwa ke bucket Amazon S3 yang ditentukan. Biasanya, CloudTrail mengirimkan file log dalam waktu 15 menit dari aktivitas akun.

> **Note**
>
> Jika Anda tidak mengonfigurasi jejak, Anda masih dapat melihat peristiwa terbaru dalam konsol CloudTrail di Riwayat peristiwa.

Anda dapat membuat dua jenis jejak untuk akun AWS: jejak yang berlaku untuk semua Wilayah, atau jejak yang berlaku untuk satu Wilayah. Secara default, ketika Anda membuat jejak di konsol tersebut, jejak diterapkan ke semua Wilayah.

Selain itu, Anda dapat mengonfigurasi layanan AWS lainnya untuk menganalisis lebih lanjut dan bertindak berdasarkan data peristiwa yang dikumpulkan di log CloudTrail. Untuk informasi selengkapnya, lihat:

- Gambaran umum untuk membuat jejak
- Layanan yang didukung dan integrasi CloudTrail
- Mengonfigurasi notifikasi Amazon SNS untuk CloudTrail
- Menerima file log CloudTrail dari banyak Wilayah dan Menerima file log CloudTrail dari banyak akun

Entri file log Amazon Aurora

File log CloudTrail berisi satu atau beberapa entri log. File log CloudTrail bukan merupakan jejak tumpukan panggilan API publik yang berurutan, sehingga file tersebut tidak muncul dalam urutan tertentu.

Contoh berikut menunjukkan entri log CloudTrail yang menunjukkan tindakan `CreateDBInstance`.

```
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
```

```
    "eventTime": "2018-07-30T22:14:06Z",
    "eventSource": "rds.amazonaws.com",
    "eventName": "CreateDBInstance",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.15.42 Python/3.6.1 Darwin/17.7.0 botocore/1.10.42",
    "requestParameters": {
        "enableCloudwatchLogsExports": [
            "audit",
            "error",
            "general",
            "slowquery"
        ],
        "dBInstanceIdentifier": "test-instance",
        "engine": "mysql",
        "masterUsername": "myawsuser",
        "allocatedStorage": 20,
        "dBInstanceClass": "db.m1.small",
        "masterUserPassword": "****"
    },
    "responseElements": {
        "dBInstanceArn": "arn:aws:rds:us-east-1:123456789012:db:test-instance",
        "storageEncrypted": false,
        "preferredBackupWindow": "10:27-10:57",
        "preferredMaintenanceWindow": "sat:05:47-sat:06:17",
        "backupRetentionPeriod": 1,
        "allocatedStorage": 20,
        "storageType": "standard",
        "engineVersion": "8.0.28",
        "dbInstancePort": 0,
        "optionGroupMemberships": [
            {
                "status": "in-sync",
                "optionGroupName": "default:mysql-8-0"
            }
        ],
        "dBParameterGroups": [
            {
                "dBParameterGroupName": "default.mysql8.0",
                "parameterApplyStatus": "in-sync"
            }
        ],
        "monitoringInterval": 0,
        "dBInstanceClass": "db.m1.small",
```

```
        "readReplicaDBInstanceIdentifiers": [],
        "dBSubnetGroup": {
            "dBSubnetGroupName": "default",
            "dBSubnetGroupDescription": "default",
            "subnets": [
                {
                    "subnetAvailabilityZone": {"name": "us-east-1b"},
                    "subnetIdentifier": "subnet-cbfff283",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1e"},
                    "subnetIdentifier": "subnet-d7c825e8",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1f"},
                    "subnetIdentifier": "subnet-6746046b",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1c"},
                    "subnetIdentifier": "subnet-bac383e0",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1d"},
                    "subnetIdentifier": "subnet-42599426",
                    "subnetStatus": "Active"
                },
                {
                    "subnetAvailabilityZone": {"name": "us-east-1a"},
                    "subnetIdentifier": "subnet-da327bf6",
                    "subnetStatus": "Active"
                }
            ],
            "vpcId": "vpc-136a4c6a",
            "subnetGroupStatus": "Complete"
        },
        "masterUsername": "myawsuser",
        "multiAZ": false,
        "autoMinorVersionUpgrade": true,
        "engine": "mysql",
        "cACertificateIdentifier": "rds-ca-2015",
```

```
        "dbiResourceId": "db-ETDZIIXHEWY5N7GXVC4SH7H5IA",
        "dBSecurityGroups": [],
        "pendingModifiedValues": {
            "masterUserPassword": "****",
            "pendingCloudwatchLogsExports": {
                "logTypesToEnable": [
                    "audit",
                    "error",
                    "general",
                    "slowquery"
                ]
            }
        },
        "dBInstanceStatus": "creating",
        "publiclyAccessible": true,
        "domainMemberships": [],
        "copyTagsToSnapshot": false,
        "dBInstanceIdentifier": "test-instance",
        "licenseModel": "general-public-license",
        "iAMDatabaseAuthenticationEnabled": false,
        "performanceInsightsEnabled": false,
        "vpcSecurityGroups": [
            {
                "status": "active",
                "vpcSecurityGroupId": "sg-f839b688"
            }
        ]
    },
    "requestID": "daf2e3f5-96a3-4df7-a026-863f96db793e",
    "eventID": "797163d3-5726-441d-80a7-6eeb7464acd4",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


Seperti ditunjukkan pada elemen `userIdentity` dalam contoh sebelumnya, setiap entri peristiwa atau log berisi informasi tentang siapa yang membuat permintaan. Informasi identitas membantu Anda menentukan berikut ini:

- Apakah permintaan tersebut dibuat dengan kredensial root atau pengguna IAM.
- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna gabungan.
- Apakah permintaan tersebut dibuat oleh layanan AWS lain.

Untuk informasi selengkapnya tentang `userIdentity`, lihat CloudTrail userIdentity Element. Untuk informasi selengkapnya tentang `CreateDBInstance` dan tindakan Amazon Aurora lainnya, lihat Referensi API Amazon RDS.

Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data

Dengan menggunakan Aliran Aktivitas Basis Data, Anda dapat memantau aliran aktivitas basis data secara waktu nyaris nyata.

Topik

- Ikhtisar Aliran Aktivitas Basis Data
- Prasyarat jaringan untuk aliran aktivitas basis data Aurora MySQL
- Memulai aliran aktivitas basis data
- Mendapatkan status aliran aktivitas basis data
- Menghentikan aliran aktivitas basis data
- Memantau aliran aktivitas basis data
- Mengelola akses ke aliran aktivitas basis data

Ikhtisar Aliran Aktivitas Basis Data

Sebagai seorang administrator basis data Amazon Aurora, Anda perlu melindungi basis data Anda dan memenuhi persyaratan kepatuhan dan peraturan. Satu strateginya adalah mengintegrasikan aliran aktivitas basis data dengan alat pemantauan. Dengan cara ini, Anda memantau dan mengatur alarm untuk aktivitas pengauditan basis data klaster Amazon Aurora .

Ancaman keamanan bersifat eksternal dan internal. Untuk melindungi terhadap ancaman internal, Anda dapat mengendalikan akses administrator ke aliran data dengan mengonfigurasikan fitur Aliran Aktivitas Basis Data. DBA tidak memiliki akses ke pengumpulan, pengiriman, penyimpanan, dan pengolahan aliran.

Topik

- Cara kerja aliran aktivitas basis data
- Mode asinkron dan sinkron untuk aliran aktivitas basis data
- Persyaratan dan keterbatasan untuk aliran aktivitas basis data
- Kawasan dan ketersediaan versi
- Kelas-kelas instans basis data yang didukung untuk aliran aktivitas basis data

Cara kerja aliran aktivitas basis data

Di Amazon Aurora, Anda memulai aliran aktivitas basis data di tingkat klaster. Semua instans basis data dalam klaster Anda mengaktifkan aliran aktivitas basis data.

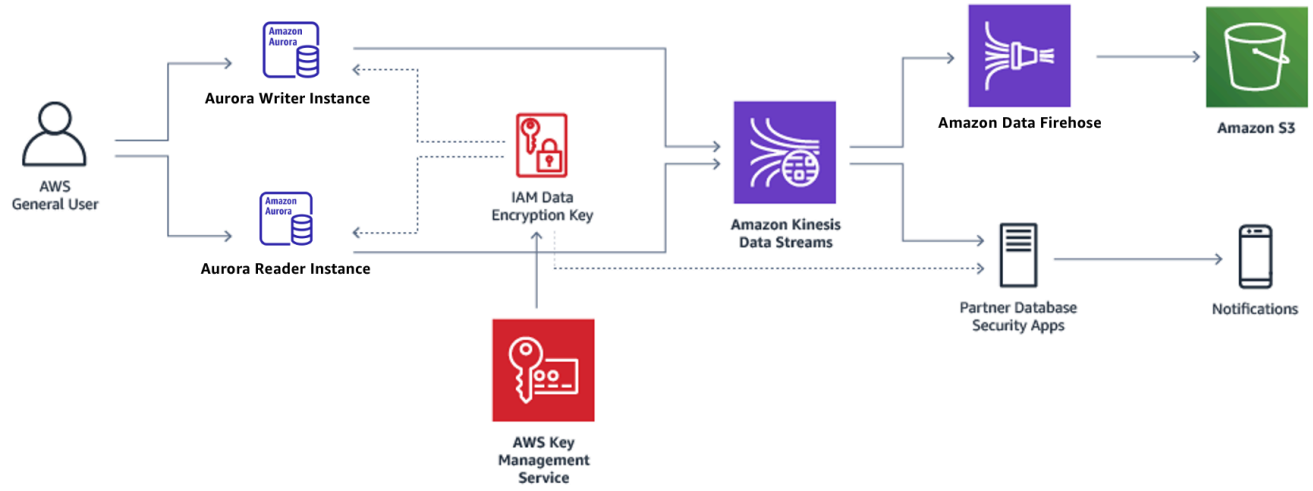
Klaster basis data Aurora Anda mendorong aktivitas ke aliran data Amazon Kinesis dalam waktu nyaris nyata. Aliran Kinesis dibuat secara otomatis. Dari Kinesis, Anda dapat mengonfigurasi AWS layanan seperti Amazon Data Firehose dan AWS Lambda menggunakan aliran dan menyimpan data.

> **Important**
>
> Penggunaan fitur aliran aktivitas basis data di Amazon RDS adalah gratis, tetapi Amazon Kinesis mengenakan biaya untuk aliran data. Lihat informasi yang lebih lengkap di struktur harga Amazon Kinesis Data Streams.

Jika Anda menggunakan basis data global Aurora, mulai secara terpisah aliran aktivitas basis data pada setiap klaster basis data. Setiap klaster mengirimkan data audit ke aliran Kinesisnya sendiri dalam Wilayah AWS-nya sendiri. Aliran aktivitas tidak beroperasi secara berbeda selama pindah saat gagal/failover. Aliran akan terus mengaudit basis data global Anda seperti biasa.

Anda dapat mengonfigurasikan aplikasi untuk pengelolaan kepatuhan agar menggunakan aliran aktivitas basis data. Untuk Aurora PostgreSQL, aplikasi kepatuhan mencakup IBM Security Guardium dan Audit dan Perlindungan Database Imperva. SecureSphere Aplikasi-aplikasi ini dapat menggunakan aliran untuk menghasilkan peringatan dan aktivitas audit pada klaster basis data Aurora.

Grafik berikut menunjukkan cluster Aurora DB yang dikonfigurasi dengan Amazon Data Firehose.



Mode asinkron dan sinkron untuk aliran aktivitas basis data

Anda dapat memilih agar sesi basis data menangani peristiwa aktivitas basis data dalam salah satu mode berikut:

- Mode asinkron – Apabila sesi basis data menghasilkan peristiwa aliran aktivitas, sesi akan kembali dengan seketika ke aktivitas normal. Di latar belakang, peristiwa aliran aktivitas dijadikan rekam yang awet. Jika kesalahan terjadi dalam tugas latar belakang, peristiwa RDS akan dikirim. Peristiwa ini menunjukkan awal dan akhir segala jendela waktu ketika rekam peristiwa aliran aktivitas mungkin telah hilang.

  Mode asinkron lebih memprioritaskan kinerja basis data daripada akurasi aliran aktivitas.

  > **Note**
  >
  > Mode asinkron tersedia untuk baik Aurora PostgreSQL maupun Aurora MySQL.

- Mode sinkron – Ketika sesi basis data menghasilkan peristiwa aliran aktivitas, sesi ini memblokir aktivitas-aktivitas lain sampai peristiwa itu dijadikan awet. Jika peristiwa tidak dapat dijadikan awet karena suatu alasan, sesi basis data akan kembali ke aktivitas normal. Namun, peristiwa RDS dikirim yang menunjukkan bahwa rekam aliran aktivitas mungkin hilang selama beberapa waktu. Peristiwa RDS kedua dikirim setelah sistem kembali ke keadaan sehat.

  Mode sinkron lebih memprioritaskan akurasi aliran aktivitas daripada kinerja basis data.

> **Note**
>
> Mode sinkron tersedia untuk Aurora PostgreSQL. Anda tidak dapat menggunakan mode sinkron dengan Aurora MySQL.

Persyaratan dan keterbatasan untuk aliran aktivitas basis data

Dalam Aurora, aliran aktivitas basis data memiliki persyaratan dan keterbatasan berikut:

- Amazon Kinesis diharuskan untuk aliran aktivitas basis data.
- AWS Key Management Service (AWS KMS) diperlukan untuk aliran aktivitas database karena selalu dienkripsi.
- Menerapkan enkripsi tambahan ke aliran data Amazon Kinesis Anda tidak kompatibel dengan aliran aktivitas database, yang sudah dienkripsi dengan kunci Anda. AWS KMS
- Mulai aliran aktivitas basis data Anda di tingkat klaster basis data. Jika Anda menambahkan instans basis data ke klaster, Anda tidak perlu memulai aliran aktivitas pada instans: instans: aliran diaudit secara otomatis.
- Di basis data global Aurora, pastikan untuk memulai secara terpisah aliran aktivitas pada setiap klaster basis data. Setiap klaster mengirimkan data audit ke aliran Kinesisnya sendiri dalam Wilayah AWS-nya sendiri.
- Di Aurora PostgreSQL, pastikan untuk menghentikan aliran aktivitas basis data sebelum pemutakhiran. Anda dapat memulai aliran aktivitas basis data setelah pemutakhiran selesai.

Kawasan dan ketersediaan versi

Ketersediaan fitur dan dukungan bervariasi di seluruh versi khusus dari setiap mesin basis data Aurora, dan di seluruh Wilayah AWS. Lihat informasi yang lebih lengkap tentang ketersediaan versi dan Kawasan dengan Aurora dan aliran aktivitas basis data di Aliran aktivitas basis data di Aurora.

Kelas-kelas instans basis data yang didukung untuk aliran aktivitas basis data

Untuk Aurora MySQL, Anda dapat menggunakan aliran aktivitas basis data dengan kelas-kelas instans basis data berikut:

- db.r7g.*large

- db.r6g.*large
- db.r6i.*large
- db.r5.*large
- db.x2g.*

Untuk Aurora PostgreSQL, Anda dapat menggunakan aliran aktivitas basis data dengan kelas-kelas instans basis data berikut:

- db.r7g.*large
- db.r6g.*large
- db.r6i.*large
- db.r6id.*large
- db.r5.*large
- db.r4.*large
- db.x2g.*

Prasyarat jaringan untuk aliran aktivitas basis data Aurora MySQL

Di bagian berikut, Anda dapat menemukan cara mengonfigurasikan cloud privat virtual (VPC) untuk digunakan dengan aliran aktivitas basis data.

Topik

- Prasyarat untuk titik akhir AWS KMS
- Prasyarat untuk ketersediaan publik
- Prasyarat untuk ketersediaan privat

Prasyarat untuk titik akhir AWS KMS

Instans dalam klaster Aurora MySQL yang menggunakan aliran aktivitas harus dapat mengakses titik akhir AWS KMS. Pastikan bahwa persyaratan ini terpenuhi sebelum Anda mengaktifkan aliran aktivitas basis data untuk klaster Aurora MySQL. Jika klaster Aurora tersedia untuk umum, persyaratan ini terpenuhi secara otomatis.

> **Important**
>
> Jika klaster DB Aurora MySQL tidak dapat mengakses titik akhir AWS KMS, aliran aktivitas berhenti. Dalam hal itu, Aurora memberi tahu Anda tentang masalah lewat Peristiwa RDS.

Prasyarat untuk ketersediaan publik

Agar menjadi publik, klaster DB Aurora harus memenuhi persyaratan berikut:

- Dapat Diakses secara Publik adalah Ya di halaman detail klaster AWS Management Console.
- Klaster DB berada di subnet publik Amazon VPC. Untuk informasi selengkapnya tentang instans DB yang dapat diakses publik, lihat Bekerja dengan instans DB dalam VPC. Untuk informasi selengkapnya tentang subnet Amazon VPC publik, lihat VPC dan Subnet Anda.

Prasyarat untuk ketersediaan privat

Jika klaster DB Aurora Anda berada di subnet publik VPC dan tidak dapat diakses publik, klaster itu bersifat privat. Untuk menjaga klaster tetap privat dan menggunakannya dengan aliran aktivitas basis data, Anda memiliki opsi-opsi berikut:

- Konfigurasikan Network Address Translation (NAT) di VPC Anda. Untuk informasi selengkapnya, lihat Gateway NAT.
- Buat titik akhir AWS KMS di VPC Anda. Opsi ini disarankan karena lebih mudah dikonfigurasikan.

Untuk membuat titik akhir AWS KMS di VPC Anda

1. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
2. Pada panel navigasi, silakan pilih Titik akhir.
3. Pilih Buat Titik Akhir.

   Halaman Buat Titik Akhir muncul.
4. Lakukan hal berikut:

   - Dalam Kategori layanan, pilih Layanan AWS.
   - Di Nama Layanan, pilih com.amazonaws.***wilayah***.kms, dengan *wilayah* adalah Wilayah AWS tempat klaster Anda berada.

- Untuk VPC, pilih VPC tempat klaster Anda berada.

5. Pilih Buat Titik Akhir.

Untuk informasi selengkapnya tentang cara mengonfigurasikan titik akhir VPC, lihat Titik Akhir VPC.

Memulai aliran aktivitas basis data

Untuk memantau aktivitas basis data bagi semua instans dalam klaster basis data Aurora, mulai aliran aktivitas pada tingkat klaster. Setiap instans basis data yang Anda tambahkan ke klaster turut dipantau secara otomatis. Jika Anda menggunakan basis data global Aurora, mulai secara terpisah aliran aktivitas basis data pada setiap klaster basis data. Setiap klaster mengirimkan data audit ke aliran Kinesisnya sendiri dalam Wilayah AWS-nya sendiri.

Ketika Anda memulai aliran aktivitas , setiap peristiwa aktivitas basis data yang Anda konfigurasikan dalam kebijakan audit akan menghasilkan peristiwa aliran aktivitas. Perintah-perintah SQL seperti `CONNECT` dan `SELECT` menghasilkan peristiwa akses. Perintah-perintah SQL seperti `CREATE` dan `INSERT` menghasilkan peristiwa perubahan.

Konsol

Untuk memulai aliran aktivitas basis data

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster basis data tempat Anda ingin memulai aliran aktivitas.
4. Untuk Tindakan, pilih Mulai aliran aktivitas.

   Jendela Mulai aliran aktivitas basis data activity: *nama* muncul, dengan *nama* adalah klaster basis data Anda.
5. Masukkan setelan berikut:

   - Untuk AWS KMS key, pilih sebuah kunci dari daftar AWS KMS keys.

> **Note**
>
> Jika klaster Aurora MySQL Anda tidak dapat mengakses kunci KMS, ikuti petunjuk dalam Prasyarat jaringan untuk aliran aktivitas basis data Aurora MySQL untuk mengaktifkan dahulu akses itu.

Aurora menggunakan kunci KMS untuk mengenkripsi kunci yang pada gilirannya mengenkripsi aktivitas basis data. Pilih kunci KMS selain kunci bawaan. Lihat informasi yang lebih lengkap tentang kunci enkripsi dan AWS KMS di Apakah AWS Key Management Service? dalam Panduan Pengembang AWS Key Management Service.

- Untuk Mode aliran aktivitas basis data, pilih Asinkron atau Sinkron.

> **Note**
>
> Pilihan ini hanya berlaku untuk Aurora PostgreSQL. Untuk Aurora MySQL, Anda hanya bisa menggunakan mode asinkron.

- Pilih Seketika.

  Bila Anda memilih Segera, klaster basis data memulai ulang dengan seketika. Jika Anda memilih Selama jendela pemeliharaan berikutnya, klaster basis data tidak seketika memulai ulang. Dalam hal ini, aliran aktivitas basis data tidak dimulai hingga jendela pemeliharaan berikutnya.

6. Pilih Mulai aliran aktivitas basis data.

   Status untuk klaster basis data menunjukkan bahwa aliran aktivitas dimulai.

> **Note**
>
> Jika Anda mendapatkan kesalahan `You can't start a database activity stream in this configuration`, periksa Kelas-kelas instans basis data yang didukung untuk aliran aktivitas basis data untuk melihat apakah klaster basis data Anda menggunakan kelas instans yang didukung.

AWS CLI

Untuk memulai aliran aktivitas database untuk cluster DB , konfigurasikan DB cluster menggunakan start-activity-streamAWS CLIperintah.

- `--resource-arn` *`arn`* – Menentukan Amazon Resource Name (ARN) klaster basis data.
- `--mode` *`sync-or-async`* – Menentukan mode sinkron (`sync`) atau asinkron (`async`). Untuk Aurora PostgreSQL, Anda dapat memilih salah satu nilai. Untuk Aurora MySQL, pilih `async`.
- `--kms-key-id` *`key`* – Menentukan pengidentifikasi kunci KMS untuk mengenkripsi pesan dalam aliran aktivitas basis data. Pengidentifikasi kunci KMS AWS adalah ARN kunci, ID kunci, ARN alias, atau nama alias bagi AWS KMS key.

Contoh berikut memulai aliran aktivitas basis data untuk sebuah klaster basis data dalam mode asinkron.

Untuk Linux, macOS, atau Unix:

```
aws rds start-activity-stream \
    --mode async \
    --kms-key-id my-kms-key-arn \
    --resource-arn my-cluster-arn \
    --apply-immediately
```

Untuk Windows:

```
aws rds start-activity-stream ^
    --mode async ^
    --kms-key-id my-kms-key-arn ^
    --resource-arn my-cluster-arn ^
    --apply-immediately
```

API RDS

Untuk memulai aliran aktivitas database untuk cluster DB, , konfigurasikan cluster menggunakan StartActivityStreamoperasi.

Panggil tindakan dengan parameter-parameter di bawah:

- `Region`

- `KmsKeyId`
- `ResourceArn`
- `Mode`

Mendapatkan status aliran aktivitas basis data

Anda bisa mendapatkan status aliran aktivitas menggunakan konsol atau AWS CLI.

Konsol

Untuk mendapatkan status aliran aktivitas basis data

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih tautan klaster DB.
3. Pilih tab Konfigurasi, dan periksa Aliran aktivitas basis data untuk statusnya.

AWS CLI

Anda bisa mendapatkan konfigurasi aliran aktivitas untuk klaster DB sebagai respons terhadap permintaan CLI describe-db-clusters .

Contoh berikut menjelaskan *`my-cluster`*.

```
aws rds --region my-region describe-db-clusters --db-cluster-identifier my-cluster
```

Contoh berikut menunjukkan respons JSON. Bidang-bidang berikut ditampilkan:

- `ActivityStreamKinesisStreamName`
- `ActivityStreamKmsKeyId`
- `ActivityStreamStatus`
- `ActivityStreamMode`
-

Bidang-bidang ini sama untuk Aurora PostgreSQL dan Aurora MySQL, kecuali bahwa `ActivityStreamMode` selalu `async` untuk Aurora MySQL, sedangkan untuk Aurora PostgreSQL mungkin `sync` atau `async`.

```
{
    "DBClusters": [
        {
      "DBClusterIdentifier": "my-cluster",
            ...
            "ActivityStreamKinesisStreamName": "aws-rds-das-cluster-
A6TSYXITZCZXJHIRVFUBZ5LTWY",
            "ActivityStreamStatus": "starting",
            "ActivityStreamKmsKeyId": "12345678-abcd-efgh-ijkl-bd041f170262",
            "ActivityStreamMode": "async",
            "DbClusterResourceId": "cluster-ABCD123456"
            ...
        }
    ]
}
```

API RDS

Anda bisa mendapatkan konfigurasi aliran aktivitas untuk klaster DB sebagai respons operasi DescribeDBClusters .

Menghentikan aliran aktivitas basis data

Anda dapat menghentikan aliran aktivitas menggunakan konsol atau AWS CLI.

Jika Anda menghapus klaster basis data, aliran aktivitas dihentikan dan aliran Amazon Kinesis yang mendasari dihapus secara otomatis.

Konsol

Untuk mematikan aliran aktivitas

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Database.
3. Pilih klaster basis data yang ingin Anda hentikan aliran aktivitas basis datanya.
4. Untuk Tindakan, pilih Hentikan aliran aktivitas. Jendela Aliran Aktivitas Basis Data akan muncul.
    a. Pilih Seketika.

        Bila Anda memilih Segera, klaster basis data memulai ulang dengan seketika. Jika Anda memilih Selama jendela pemeliharaan berikutnya, klaster basis data tidak seketika memulai

ulang. Dalam hal ini, aliran aktivitas basis data tidak berhenti hingga jendela pemeliharaan berikutnya.

b. Pilih Lanjutkan.

AWS CLI

Untuk menghentikan aliran aktivitas database untuk DB cluster , konfigurasikan menggunakan AWS CLI perintah stop-activity-stream. Tandai Kawasan AWS untuk klaster basis data dengan menggunakan parameter `--region`. Parameter `--apply-immediately` bersifat opsional.

Untuk Linux, macOS, atau Unix:

```
aws rds --region MY_REGION \
    stop-activity-stream \
    --resource-arn MY_CLUSTER_ARN \
    --apply-immediately
```

Untuk Windows:

```
aws rds --region MY_REGION ^
    stop-activity-stream ^
    --resource-arn MY_CLUSTER_ARN ^
    --apply-immediately
```

API RDS

Untuk menghentikan aliran aktivitas database untuk DB cluster , konfigurasikan cluster menggunakan StopActivityStreamoperasi. Tandai Kawasan AWS untuk klaster basis data dengan menggunakan parameter `Region`. Parameter `ApplyImmediately` bersifat opsional.

Memantau aliran aktivitas basis data

Aliran aktivitas basis data memantau dan melaporkan aktivitas. Aliran aktivitas dikumpulkan dan dikirim ke Amazon Kinesis. Dari Kinesis, Anda dapat memantau aliran aktivitas, atau layanan dan aplikasi lain dapat menggunakan aliran aktivitas untuk analisis lebih lanjut. Anda dapat menemukan nama aliran Kinesis yang mendasarinya dengan menggunakan AWS CLI perintah `describe-db-clusters`atau operasi API RDS. `DescribeDBClusters`

Aurora mengelola aliran Kinesis untuk Anda sebagai berikut:

- Aurora membuat aliran Kinesis secara otomatis dengan periode retensi 24 jam.
- Aurora menskalakan aliran Kinesis jika perlu.
- Jika Anda menghentikan aliran aktivitas basis data atau menghapus klaster basis data, Aurora menghapus aliran Kinesis.

Kategori-kategori aktivitas berikut dipantau dan dimasukkan ke dalam log audit aliran aktivitas:

- Perintah SQL – Semua perintah SQL diaudit, begitu pula dengan pernyataan yang disiapkan, fungsi bawaan, dan fungsi dalam PL/SQL. Panggilan ke prosedur tersimpan akan diaudit. Setiap pernyataan SQL yang diterbitkan di dalam prosedur atau fungsi tersimpan juga diaudit.
- Informasi basis data lainnya – Aktivitas yang dipantau mencakup pernyataan SQL lengkap, hitungan baris yang terpengaruh dari perintah DML, objek yang diakses, dan nama basis data unik. Untuk Aurora PostgreSQL, aliran aktivitas basis data juga memantau variabel pengikatan dan parameter prosedur tersimpan.

  > **Important**
  >
  > Teks SQL lengkap setiap pernyataan, yang meliputi semua data sensitif, dapat dilihat di log audit aliran aktivitas. Namun, kata sandi pengguna basis data disensor jika Aurora dapat memastikannya dari konteks, seperti dalam pernyataan SQL berikut.
  >
  > ```
  > ALTER ROLE role-name WITH password
  > ```

- Informasi koneksi – Aktivitas yang dipantau mencakup sesi dan informasi jaringan, ID proses server, dan kode keluar.

Jika aliran aktivitas mengalami kegagalan saat memantau instans basis data, Anda akan diberi tahu melalui peristiwa RDS.

Topik

- Mengakses aliran aktivitas dari Kinesis
- Isi dan contoh log Audit
- databaseActivityEventDaftar array JSON
- Memproses aliran aktivitas database menggunakan AWS SDK

Mengakses aliran aktivitas dari Kinesis

Saat Anda mengaktifkan aliran aktivitas untuk klaster basis data, aliran Kinesis dibuat untuk Anda. Dari Kinesis, Anda dapat memantau aktivitas basis data Anda secara waktu nyata. Untuk menganalisis lebih lanjut aktivitas basis data, Anda dapat menghubungkan aliran Kinesis dengan aplikasi konsumen. Anda juga dapat menghubungkan aliran ke aplikasi manajemen kepatuhan seperti IBM Security Guardium atau Audit SecureSphere Database Imperva dan Perlindungan IBM Security Guardium atau Audit dan Perlindungan . SecureSphere

Anda dapat mengakses aliran Kinesis dari konsol RDS atau konsol Kinesis.

Untuk mengakses aliran aktivitas dari Kinesis dengan menggunakan konsol RDS

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster basis data tempat Anda memulai aliran aktivitas.
4. Pilih Konfigurasi.
5. Di bawah Aliran aktivitas basis data, pilih penaut di bawah Aliran Kinesis.
6. Di konsol Kinesis, pilih Pemantauan untuk mulai mengamati aktivitas basis data.

Untuk mengakses aliran aktivitas dari Kinesis dengan menggunakan konsol Kinesis

1. Buka konsol Kinesis di https://console.aws.amazon.com/kinesis.
2. Pilih aliran aktivitas Anda dari daftar aliran Kinesis.

   Nama aliran aktivitas mencakup awalan `aws-rds-das-cluster-` diikuti dengan ID sumber daya klaster basis data. Berikut sebuah contohnya.

   ```
   aws-rds-das-cluster-NHVOV4PCLWHGF52NP
   ```

   Untuk memakai konsol Amazon RDS guna menemukan ID sumber daya bagi klaster basis data, pilih klaster basis data Anda dari daftar basis data, lalu pilih tab Konfigurasi.

   Untuk menggunakan AWS CLI untuk menemukan nama aliran Kinesis lengkap untuk aliran aktivitas, gunakan permintaan describe-db-clustersCLI dan catat nilai dalam respons. `ActivityStreamKinesisStreamName`
3. Pilih Pemantauan untuk mulai mengamati aktivitas basis data.

Lihat informasi yang lebih lengkap tentang penggunaan Amazon Kinesis di Apakah Amazon Kinesis Data Streams?.

Isi dan contoh log Audit

Peristiwa-peristiwa yang dipantau disajikan dalam aliran aktivitas basis data berupa string JSON. Strukturnya terdiri atas objek JSON yang berisi `DatabaseActivityMonitoringRecord`, yang selanjutnya berisi sebuah larik peristiwa aktivitas `databaseActivityEventList`.

Topik

- Contoh-contoh log audit untuk aliran aktivitas
- Objek JSON DatabaseActivityMonitoringRecords
- databaseActivityEvents Objek JSON

Contoh-contoh log audit untuk aliran aktivitas

Berikut adalah contoh log audit JSON terdekripsi dari rekam peristiwa aktivitas.

Example Rekam peristiwa aktivitas dari Aurora PostgreSQL

Rekam peristiwa aktivitas berikut menunjukkan upaya masuk dengan penggunaan pernyataan SQL `CONNECT` (`command`) oleh klien psql (`clientApplication`).

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-10-30 00:39:49.940668+00",
          "logTime": "2019-10-30 00:39:49.990579+00",
          "statementId": 1,
          "substatementId": 1,
          "objectType": null,
          "command": "CONNECT",
          "objectName": null,
          "databaseName": "postgres",
```

```
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "49804",
          "sessionId": "5ce5f7f0.474b",
          "rowCount": null,
          "commandText": null,
          "paramList": [],
          "pid": 18251,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "MISC",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

Example Rekam peristiwa aktivitas pernyataan SQL CONNECT Aurora MySQL

Rekam peristiwa aktivitas berikut menunjukkan upaya masuk dengan penggunaan pernyataan SQL `CONNECT` (`command`) oleh klien mysql (`clientApplication`).

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:13.267214+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"rdsadmin",
      "databaseName":"",
      "remoteHost":"localhost",
```

```
      "remotePort":"11053",
      "command":"CONNECT",
      "commandText":"",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"",
      "statementId":0,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"725121",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:13.267207+00",
      "endTime":"2020-05-22 18:07:13.267213+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```


Example Rekam peristiwa aktivitas pernyataan CREATE TABLE Aurora PostgreSQL

Contoh berikut menunjukkan peristiwa `CREATE TABLE` untuk Aurora PostgreSQL.


```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-05-24 00:36:54.403455+00",
          "logTime": "2019-05-24 00:36:54.494235+00",
          "statementId": 2,
          "substatementId": 1,
```

```
          "objectType": null,
          "command": "CREATE TABLE",
          "objectName": null,
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
          "rowCount": null,
          "commandText": "create table my_table (id serial primary key, name
 varchar(32));",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "DDL",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

Example Rekam peristiwa aktivitas pernyataan CREATE TABLE Aurora MySQL

Contoh berikut menunjukkan pernyataan `CREATE TABLE` untuk Aurora MySQL. Operasi ini disajikan sebagai dua rekam peristiwa terpisah. Satu peristiwa memiliki `"class":"MAIN"`. Peristiwa yang lain memiliki `"class":"AUX"`. Pesan-pesan mungkin tiba dengan sebarang urutan. Bidang `logTime` peristiwa `MAIN` selalu lebih awal dari bidang-bidang `logTime` sabarang peristiwa `AUX` yang terkait.

Contoh berikut menunjukkan peristiwa dengan nilai `class` berupa `MAIN`.

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
```

```
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:07:12.250221+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
      "commandText":"CREATE TABLE test1 (id INT)",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.250222+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```


Contoh berikut menunjukkan peristiwa yang bersangkutan dengan nilai `class` berupa AUX.


```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
```

```
      "logTime":"2020-05-22 18:07:12.247182+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"CREATE",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65459278,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"725118",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:07:12.226384+00",
      "endTime":"2020-05-22 18:07:12.247182+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}
```

Example Rekam peristiwa aktivitas Aurora PostgreSQL

Contoh berikut menunjukkan peristiwa SELECT .

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":
    {
      "type":"DatabaseActivityMonitoringRecord",
      "clusterId":"cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
```

```
      "instanceId":"db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
      "databaseActivityEventList":[
        {
          "startTime": "2019-05-24 00:39:49.920564+00",
          "logTime": "2019-05-24 00:39:49.940668+00",
          "statementId": 6,
          "substatementId": 1,
          "objectType": "TABLE",
          "command": "SELECT",
          "objectName": "public.my_table",
          "databaseName": "postgres",
          "dbUserName": "rdsadmin",
          "remoteHost": "172.31.3.195",
          "remotePort": "34534",
          "sessionId": "5ce73c6f.7e64",
          "rowCount": 10,
          "commandText": "select * from my_table;",
          "paramList": [],
          "pid": 32356,
          "clientApplication": "psql",
          "exitCode": null,
          "class": "READ",
          "serverVersion": "2.3.1",
          "serverType": "PostgreSQL",
          "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
          "serverHost": "172.31.3.192",
          "netProtocol": "TCP",
          "dbProtocol": "Postgres 3.0",
          "type": "record",
          "errorMessage": null
        }
      ]
    },
   "key":"decryption-key"
}
```

```
{
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "",
    "instanceId": "db-4JCWQLUZVFYP7DIWP6JVQ77O3Q",
    "databaseActivityEventList": [
        {
            "class": "TABLE",
```

```
"clientApplication": "Microsoft SQL Server Management Studio - Query",
"command": "SELECT",
"commandText": "select * from [testDB].[dbo].[TestTable]",
"databaseName": "testDB",
"dbProtocol": "SQLSERVER",
"dbUserName": "test",
"endTime": null,
"errorMessage": null,
"exitCode": 1,
"logTime": "2022-10-06 21:24:59.9422268+00",
"netProtocol": null,
"objectName": "TestTable",
"objectType": "TABLE",
"paramList": null,
"pid": null,
"remoteHost": "local machine",
"remotePort": null,
"rowCount": 0,
"serverHost": "172.31.30.159",
"serverType": "SQLSERVER",
"serverVersion": "15.00.4073.23.v1.R1",
"serviceName": "sqlserver-ee",
"sessionId": 62,
"startTime": null,
"statementId": "0x03baed90412f564fad640ebe51f89b99",
"substatementId": 1,
"transactionId": "4532935",
"type": "record",
"engineNativeAuditFields": {
    "target_database_principal_id": 0,
    "target_server_principal_id": 0,
    "target_database_principal_name": "",
    "server_principal_id": 2,
    "user_defined_information": "",
    "response_rows": 0,
    "database_principal_name": "dbo",
    "target_server_principal_name": "",
    "schema_name": "dbo",
    "is_column_permission": true,
    "object_id": 581577110,
    "server_instance_name": "EC2AMAZ-NFUJJNO",
    "target_server_principal_sid": null,
    "additional_information": "",
    "duration_milliseconds": 0,
```

```
                "permission_bitmask": "0x00000000000000000000000000000001",
                "data_sensitivity_information": "",
                "session_server_principal_name": "test",
                "connection_id": "AD3A5084-FB83-45C1-8334-E923459A8109",
                "audit_schema_version": 1,
                "database_principal_id": 1,
                "server_principal_sid":
 "0x0105000000000005150000000bdc2795e2d0717901ba6998cf4010000",
                "user_defined_event_id": 0,
                "host_name": "EC2AMAZ-NFUJJNO"
            }
        }
    ]
}
```

Example Rekam peristiwa aktivitas pernyataan SELECT Aurora MySQL

Contoh berikut menunjukkan peristiwa `SELECT`.

Contoh berikut menunjukkan peristiwa dengan nilai `class` berupa `MAIN`.

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "clusterId":"cluster-some_id",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986467+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"QUERY",
      "commandText":"SELECT * FROM test1 WHERE id < 28",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65469218,
      "substatementId":1,
      "exitCode":"0",
      "sessionId":"726571",
```

```
      "rowCount":2,
      "serverHost":"master",
      "serverType":"MySQL",
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
      "endTime":"2020-05-22 18:29:57.986467+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"MAIN"
    }
  ]
}
```

Contoh berikut menunjukkan peristiwa yang bersangkutan dengan nilai `class` berupa `AUX`.

```
{
  "type":"DatabaseActivityMonitoringRecord",
  "instanceId":"db-some_id",
  "databaseActivityEventList":[
    {
      "logTime":"2020-05-22 18:29:57.986399+00",
      "type":"record",
      "clientApplication":null,
      "pid":2830,
      "dbUserName":"master",
      "databaseName":"test",
      "remoteHost":"localhost",
      "remotePort":"11054",
      "command":"READ",
      "commandText":"test1",
      "paramList":null,
      "objectType":"TABLE",
      "objectName":"test1",
      "statementId":65469218,
      "substatementId":2,
      "exitCode":"",
      "sessionId":"726571",
      "rowCount":0,
      "serverHost":"master",
      "serverType":"MySQL",
```

```
      "serviceName":"Amazon Aurora MySQL",
      "serverVersion":"MySQL 5.7.12",
      "startTime":"2020-05-22 18:29:57.986364+00",
      "endTime":"2020-05-22 18:29:57.986399+00",
      "transactionId":"0",
      "dbProtocol":"MySQL",
      "netProtocol":"TCP",
      "errorMessage":"",
      "class":"AUX"
    }
  ]
}
```

Objek JSON DatabaseActivityMonitoringRecords

Rekam peristiwa aktivitas basis data berada dalam objek JSON yang berisi informasi berikut.

Bidang JSON	Tipe data	Deskripsi
`type`	string	Jenis rekam JSON. Nilainya adalah `DatabaseActivityMonitoringRecords` .
`version`	string	Versi rekam pemantauan aktivitas basis data. Versi rekam aktivitas basis data yang dihasilkan bergantung pada versi mesin klaster basis data: • Rekam aktivitas basis data versi 1.1 dihasilkan untuk klaster basis data Aurora PostgreSQL yang menjalankan versi mesin 10.10 dan versi kecil yang lebih baru serta versi mesin 11.5 dan yang lebih baru. • Rekam aktivitas basis data versi 1.0 dihasilkan untuk klaster basis data Aurora

Bidang JSON	Tipe data	Deskripsi
		PostgreSQL yang menjalankan versi mesin 10.7 dan 11.4. Semua bidang berikut ada dalam versi 1.0 dan versi 1.1 kecuali jika diberi catatan khusus.
databaseActivityEvents	string	Objek JSON yang berisi peristiwa aktivitas.
kunci	string	Kunci enkripsi yang Anda gunakan untuk mendekripsi databaseActivityEventDaftar

databaseActivityEvents Objek JSON

Objek JSON `databaseActivityEvents` berisi informasi berikut.

Bidang-bidang tingkat atas dalam rekam JSON

Setiap peristiwa dalam log audit dibungkus dalam sebuah rekam dalam format JSON. Rekam ini berisi bidang-bidang berikut.

tipe

Bidang ini selalu memiliki nilai `DatabaseActivityMonitoringRecords`.

versi

Bidang ini mewakili versi protokol atau kontrak data aliran aktivitas basis data. Versi menentukan bidang-bidang yang tersedia.

Versi 1.0 mewakili dukungan aliran aktivitas data asli untuk Aurora PostgreSQL versi 10.7 dan 11.4. Versi 1.1 mewakili dukungan aliran aktivitas data untuk Aurora PostgreSQL versi 10.10 dan yang lebih tinggi serta Aurora PostgreSQL versi 11.5 dan yang lebih tinggi. Versi 1.1 mencakup bidang-bidang tambahan `errorMessage` dan `startTime`. Versi 1.2 mewakili dukungan aliran aktivitas data untuk Aurora MySQL 2.08 dan lebih tinggi. Versi 1.2 mencakup bidang-bidang tambahan `endTime` dan `transactionId`.

databaseActivityEvents

String terenkripsi yang mewakili satu atau beberapa peristiwa aktivitas. String disajikan berupa larik byte base64. Saat Anda mendekripsi string, hasilnya adalah rekam dalam format JSON dengan bidang-bidang seperti ditunjukkan dalam contoh di bagian ini.

kunci

Kunci data terenkripsi yang digunakan untuk mengenkripsi string `databaseActivityEvents`. Ini sama dengan AWS KMS key yang Anda berikan saat memulai aliran aktivitas database.

Contoh berikut menunjukkan format rekam ini.

```
{
  "type":"DatabaseActivityMonitoringRecords",
  "version":"1.1",
  "databaseActivityEvents":"encrypted audit records",
  "key":"encrypted key"
}
```

Lakukan langkah-langkah berikut untuk mendekripsi isi bidang `databaseActivityEvents`:

1. Lakukan dekripsi nilai dalam bidang JSON `key` dengan menggunakan kunci KMS yang Anda sediakan ketika memulai aliran aktivitas basis data. Melakukan hal itu akan menghasilkan kunci enkripsi data berupa teks jelas.
2. Base64 mendekode nilai dalam bidang JSON `databaseActivityEvents` untuk mendapatkan teks sandi, dalam format biner, dari muatan audit.
3. Lakukan dekripsi teks sandi biner dengan kunci enkripsi data yang Anda dekode pada langkah pertama.
4. Lakukan dekompresi muatan yang terdekripsi.
   - Muatan terenkripsi ada di bidang `databaseActivityEvents`.
   - Bidang `databaseActivityEventList` berisi larik rekam audit. Bidang `type` dalam larik dapat berupa `record` atau `heartbeat`.

Rekam peristiwa aktivitas log audit adalah objek JSON yang berisi informasi berikut.

Bidang JSON	Tipe data	Deskripsi
`type`	string	Jenis rekam JSON. Nilainya adalah `DatabaseActivityMonitoringRecord` .
`clusterId`	string	Pengidentifikasi sumber daya klaster basis data. Pengidentifikasi ini berkaitan dengan atribut klaster basis data `DbClusterResourceId` .
`instanceId`	string	Pengidentifikasi sumber daya instans basis data. Pengidentifikasi ini berkaitan dengan atribut instans basis data `DbiResourceId` .
databaseActivityEventDaftar	string	Larik rekam audit aktivitas atau pesan denyut jantung.

databaseActivityEventDaftar array JSON

Muatan log audit adalah larik JSON `databaseActivityEventList` terenkripsi. Tabel-tabel berikut memerinci secara alfabetis bidang-bidang untuk setiap peristiwa aktivitas dalam larik `DatabaseActivityEventList` terdekripsi sebuah log audit. Bidang-bidang akan berbeda tergantung pada apakah Anda menggunakan Aurora PostgreSQL atau Aurora MySQL. Rujuk ke tabel yang relevan dengan mesin basis data Anda.

> **Important**
>
> Struktur peristiwa dapat berubah sewaktu-waktu. Aurora mungkin menambahkan bidang-bidang baru ke peristiwa aktivitas di masa mendatang. Dalam aplikasi yang menguraikan data JSON, pastikan bahwa kode Anda dapat mengabaikan atau mengambil tindakan yang tepat untuk nama-nama bidang yang tidak dikenal.

databaseActivityEventKolom daftar untuk Aurora PostgreSQL

Field	Tipe data	Deskripsi
`class`	string	Kelas peristiwa aktivitas. Nilai-nilai yang valid untuk Aurora PostgreSQL adalah:

Field	Tipe data	Deskripsi
		• ALL • CONNECT – Peristiwa koneksi atau pemutusan. • DDL – Pernyataan DDL yang tidak termasuk dalam daftar pernyataan untuk kelas ROLE. • FUNCTION – Panggilan fungsi atau blok DO. • MISC – Perintah lain-lain, seperti DISCARD, FETCH, CHECKPOINT , atau VACUUM. • NONE • READ – Pernyataan SELECT atau COPY apabila sumbernya sebuah relasi atau kueri. • ROLE – Pernyataan seputar peran dan privilese, yang meliputi GRANT, REVOKE, dan CREATE/ALTER/DROP ROLE. • WRITE – Pernyataan INSERT, UPDATE, DELETE, TRUNCATE, atau COPY ketika tujuan adalah relasi.
clientApplication	string	Aplikasi yang digunakan klien untuk menghubungi seperti dilaporkan oleh klien. Klien tidak wajib memberikan informasi ini, sehingga nilainya dapat null.
command	string	Nama perintah SQL tanpa perincian perintah sama sekali.

Field	Tipe data	Deskripsi
`commandText`	string	Pernyataan SQL sebenarnya yang diberikan oleh pengguna. Untuk Aurora PostgreSQL, nilainya identik dengan pernyataan SQL asli. Bidang ini digunakan untuk semua jenis rekam kecuali untuk menghubungkan atau memutus rekam, yang dalam hal ini nilainya null. **Important** Teks SQL lengkap setiap pernyataan, yang meliputi semua data sensitif, dapat dilihat di log audit aliran aktivitas. Namun, kata sandi pengguna basis data disensor jika Aurora dapat menentukannya dari konteks, seperti dalam pernyataan SQL berikut. `ALTER ROLE role-name WITH password`
`databaseName`	string	Basis data yang terhubung dengan pengguna.
`dbProtocol`	string	Protokol basis data, misalnya `Postgres 3.0`.
`dbUserName`	string	Pengguna basis data yang diautentikasi oleh klien.

Field	Tipe data	Deskripsi
`errorMessage` (khusus rekam aktivitas basis data 1.1)	string	Jika ada kesalahan, bidang ini diisi dengan pesan kesalahan yang akan dihasilkan oleh server basis data. Nilai `errorMessage` adalah null untuk pernyataan normal yang tidak mengakibatkan kesalahan. Kesalahan didefinisikan sebagai sebarang aktivitas yang akan menghasilkan peristiwa log kesalahan PostgreSQL yang dapat dilihat klien pada tingkat keparahan `ERROR` atau lebih tinggi. Lihat informasi yang lebih lengkap di Tingkat Keparahan Pesan PostgreSQL. Misalnya, kesalahan sintaks dan pembatalan kueri menghasilkan pesan kesalahan. Kesalahan-kesalahan internal server PostgreSQL, seperti kesalahan proses penunjuk pemeriksaan latar belakang tidak menghasilkan pesan kesalahan. Namun, rekam untuk peristiwa seperti itu masih dipancarkan terlepas dari setelan level keparahan log. Hal ini mencegah penyerang mematikan pengelogan untuk mencoba menghindari deteksi. Lihat juga bidang `exitCode`.
`exitCode`	int	Nilai yang digunakan untuk rekam keluar sesi. Pada keluar yang mulus, ini berisi kode keluar. Kode keluar tidak selalu dapat diperoleh dalam beberapa skenario kegagalan. Contoh-contoh adalah jika PostgreSQL melakukan `exit()` atau jika operator melakukan perintah seperti `kill -9`. Jika ada kesalahan apa pun, bidang `exitCode` menampilk an kode kesalahan SQL, `SQLSTATE`, sebagaimana tercantum dalam Kode Kesalahan PostgreSQL. Lihat juga bidang `errorMessage` .
`logTime`	string	Stempel waktu seperti tercatat di jalur kode audit. Ini mewakili waktu akhir eksekusi pernyataan SQL. Lihat juga bidang `startTime` .

Field	Tipe data	Deskripsi
`netProtocol`	string	Protokol komunikasi jaringan.
`objectName`	string	Nama objek basis data jika pernyataan SQL beroperasi pada objek itu. Bidang ini hanya digunakan apabila pernyataan SQL beroperasi pada objek basis data. Jika pernyataan SQL tidak beroperasi pada sebuah objek, nilai ini null.
`objectType`	string	Jenis objek basis data, seperti tabel, indeks, tampilan, dan sebagainya. Bidang ini hanya digunakan apabila pernyataan SQL beroperasi pada objek basis data. Jika pernyataan SQL tidak beroperasi pada sebuah objek, nilai ini null. Nilai-nilai yang valid meliputi: • `COMPOSITE TYPE` • `FOREIGN TABLE` • `FUNCTION` • `INDEX` • `MATERIALIZED VIEW` • `SEQUENCE` • `TABLE` • `TOAST TABLE` • `VIEW` • `UNKNOWN`
`paramList`	string	Larik parameter terpisah koma yang disampaikan ke pernyataan SQL. Jika pernyataan SQL tidak memiliki parameter, nilai ini adalah larik kosong.
`pid`	int	ID proses proses sisi belakang yang dialokasikan untuk melayani koneksi klien.
`remoteHost`	string	Alamat IP atau nama host klien. Untuk Aurora PostgreSQL, informasi yang digunakan bergantung pada setelan parameter `log_hostname` basis data.

Field	Tipe data	Deskripsi
`remotePort`	string	Nomor porta klien.
`rowCount`	int	Jumlah baris yang dihasilkan oleh pernyataan SQL. Misalnya, jika pernyataan SELECT menghasilkan 10 baris, rowCount adalah 10. Untuk pernyataan INSERT atau UPDATE, rowCount adalah 0.
`serverHost`	string	Alamat IP host server basis data.
`serverType`	string	Jenis server basis data, misalnya `PostgreSQL` .
`serverVersion`	string	Versi server basis data, misalnya `2.3.1` untuk Aurora PostgreSQL.
`serviceName`	string	Nama layanan, misalnya `Amazon Aurora PostgreSQL-Compatible edition` .
`sessionId`	int	Pengidentifikasi sesi unik semu.
`sessionId`	int	Pengidentifikasi sesi unik semu.
`startTime` (khusus rekam aktivitas basis data 1.1)	string	Waktu ketika eksekusi dimulai untuk pernyataan SQL. Untuk menghitung waktu eksekusi kira-kira pernyataan SQL, gunakan `logTime - startTime` . Lihat juga bidang `logTime`.
`statementId`	int	Pengidentifikasi untuk pernyataan SQL klien. Penghitung berada di tingkat sesi dan bertambah dengan setiap pernyataan SQL yang dimasukkan oleh klien.
`substatementId`	int	Pengidentifikasi untuk subpernyataan SQL. Nilai ini menghitung subpernyataan terkandung untuk setiap pernyataan SQL yang diidentifikasi oleh bidang `statementId` .
`type`	string	Jenis peristiwa. Nilai-nilai yang valid adalah `record` atau `heartbeat` .

databaseActivityEventDaftar bidang untuk Aurora MySQL

Field	Tipe data	Deskripsi
`class`	string	Kelas peristiwa aktivitas. Nilai-nilai yang valid untuk Aurora MySQL adalah: • `MAIN` – Peristiwa utama yang mewakili pernyataan SQL. • `AUX` – Peristiwa tambahan yang berisi detail tambahan. Misalnya, pernyataan yang mengganti nama objek mungkin memiliki peristiwa dengan kelas `AUX` yang mencerminkan nama baru. Untuk menemukan peristiwa-peristiwa `MAIN` dan `AUX` yang berkaitan dengan pernyataan yang sama, periksa berbagai peristiwa yang memiliki nilai yang sama untuk bidang `pid` dan untuk bidang `statementId` .
`clientApplication`	string	Aplikasi yang digunakan klien untuk menghubungi seperti dilaporkan oleh klien. Klien tidak wajib memberikan informasi ini, sehingga nilainya dapat null.
`command`	string	Kategori umum pernyataan SQL. Nilai untuk bidang ini bergantung pada nilai `class`. Nilai-nilai ketika `class` adalah `MAIN` meliputi: • `CONNECT` – Ketika sesi klien terhubung. • `QUERY` – Pernyataan SQL. Disertai oleh satu atau beberapa peristiwa dengan nilai `class` adalah `AUX`. • `DISCONNECT` – Ketika sesi klien terputus. • `FAILED_CONNECT` – Ketika klien mencoba menghubung, tetapi tidak dapat. • `CHANGEUSER` – Perubahan status yang merupakan bagian dari protokol jaringan MySQL, bukan dari pernyataan yang Anda terbitkan.

Field	Tipe data	Deskripsi
		Nilai-nilai ketika `class` adalah `AUX` meliputi: • `READ` – Pernyataan `SELECT` atau `COPY` apabila sumbernya sebuah relasi atau kueri. • `WRITE` – Pernyataan `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, atau `COPY` apabila tujuannya sebuah relasi. • `DROP` – Menghapus objek. • `CREATE` – Membuat objek. • `RENAME` – Mengganti nama objek. • `ALTER` – Mengubah properti-properti suatu objek.

Field	Tipe data	Deskripsi
`commandText`	string	Untuk peristiwa dengan nilai `class` adalah `MAIN`, bidang ini mewakili pernyataan SQL sebenarnya yang diberikan oleh pengguna. Bidang ini digunakan untuk semua jenis rekam kecuali untuk menghubungkan atau memutus rekam, yang dalam hal ini nilainya null. Untuk peristiwa dengan nilai `class` adalah `AUX`, bidang ini memuat informasi tambahan tentang objek yang terlibat dalam peristiwa. Untuk Aurora MySQL, karakter-karakter seperti tanda kutip didahului oleh garis miring kiri, yang mewakili karakter pengelak/escape. **Important** Teks SQL lengkap setiap pernyataan, yang meliputi semua data sensitif, dapat dilihat di log audit. Namun, kata sandi pengguna basis data disensor jika Aurora dapat menentukannya dari konteks, seperti dalam pernyataan SQL berikut. `mysql> SET PASSWORD = 'my-password ';` **Note** Tetapkan kata sandi selain penggugah/prompt yang ditampilkan di sini sebagai praktik terbaik keamanan.
`databaseName`	string	Basis data yang terhubung dengan pengguna.
`dbProtocol`	string	Protokol basis data. Saat ini, nilai ini selalu `MySQL` untuk Aurora MySQL.

Field	Tipe data	Deskripsi
`dbUserName`	string	Pengguna basis data yang diautentikasi oleh klien.
`endTime` (khusus rekam aktivitas basis data 1.2)	string	Waktu ketika eksekusi diakhiri untuk pernyataan SQL. Disajikan dalam format Waktu Universal Terkoordinasi (UTC). Untuk menghitung waktu eksekusi pernyataan SQL, gunakan `endTime - startTime` . Lihat juga bidang `startTime` .
`errorMessage` (khusus rekam aktivitas basis data 1.1)	string	Jika ada kesalahan, bidang ini diisi dengan pesan kesalahan yang akan dihasilkan oleh server basis data. Nilai `errorMessage` adalah null untuk pernyataan normal yang tidak mengakibatkan kesalahan. Kesalahan didefinisikan sebagai sebarang aktivitas yang akan menghasilkan peristiwa log kesalahan MySQL yang dapat dilihat klien pada tingkat keparahan `ERROR` atau lebih tinggi. Lihat informasi yang lebih lengkap di Log Kesalahan dalam Manual Referensi MySQL. Misalnya, kesalahan sintaks dan pembatalan kueri menghasilkan pesan kesalahan . Kesalahan-kesalahan internal server MySQL, seperti kesalahan proses penunjuk pemeriksaan latar belakang tidak menghasilkan pesan kesalahan. Namun, rekam untuk peristiwa seperti itu masih dipancarkan terlepas dari setelan level keparahan log. Hal ini mencegah penyerang mematikan pengelogan untuk mencoba menghindari deteksi. Lihat juga bidang `exitCode`.
`exitCode`	int	Nilai yang digunakan untuk rekam keluar sesi. Pada keluar yang mulus, ini berisi kode keluar. Kode keluar tidak selalu dapat diperoleh dalam beberapa skenario kegagalan. Dalam kasus-kasus tersebut, nilai ini mungkin nol atau mungkin kosong.

Field	Tipe data	Deskripsi
`logTime`	string	Stempel waktu seperti tercatat di jalur kode audit. Disajikan dalam format Waktu Universal Terkoordinasi (UTC). Lihat cara paling akurat menghitung durasi pernyataan di bidang-bidang `startTime` dan `endTime`.
`netProtocol`	string	Protokol komunikasi jaringan. Saat ini, nilai ini selalu `TCP` untuk Aurora MySQL.
`objectName`	string	Nama objek basis data jika pernyataan SQL beroperasi pada objek itu. Bidang ini hanya digunakan apabila pernyataan SQL beroperasi pada objek basis data. Jika pernyataan SQL tidak beroperasi pada sebuah objek, nilai ini null. Untuk membangun nama objek berkualifikasi penuh, gabungkan `databaseName` dan `objectName`. Jika kueri melibatkan beberapa objek, bidang ini dapat berupa daftar nama terpisah koma.
`objectType`	string	Jenis objek basis data, seperti tabel, indeks, dan sebagainya. Bidang ini hanya digunakan apabila pernyataan SQL beroperasi pada objek basis data. Jika pernyataan SQL tidak beroperasi pada sebuah objek, nilai ini null. Nilai-nilai yang valid untuk Aurora MySQL meliputi: • `INDEX` • `TABLE` • `UNKNOWN`
`paramList`	string	Bidang ini tidak digunakan untuk Aurora MySQL dan selalu null.
`pid`	int	ID proses proses sisi belakang yang dialokasikan untuk melayani koneksi klien. Ketika server basis data dimulai ulang, `pid` berubah dan penghitung untuk bidang `statementId` dimulai ulang dari awal.

Field	Tipe data	Deskripsi
`remoteHost`	string	Alamat IP atau nama host klien yang menerbitkan pernyataan SQL. Untuk Aurora MySQL, informasi yang digunakan bergantung pada setelan parameter `skip_name_resolve` basis data. Nilai `localhost` menunjukkan aktivitas dari pengguna khusus `rdsadmin`.
`remotePort`	string	Nomor porta klien.
`rowCount`	int	Jumlah baris tabel yang terpengaruh atau diambil oleh pernyataan SQL. Bidang ini hanya digunakan untuk pernyataan SQL yang merupakan pernyataan bahasa manipulasi data (DML). Jika pernyataan SQL bukan pernyataan DML, nilai ini null.
`serverHost`	string	Pengidentifikasi instans server basis data. Nilai ini diwakili secara berbeda untuk Aurora MySQL dengan untuk Aurora PostgreSQL. Aurora PostgreSQL menggunakan alamat IP, bukan pengidentifikasi.
`serverType`	string	Jenis server basis data, misalnya `MySQL`.
`serverVersion`	string	Versi server basis data. Saat ini, nilai ini selalu `MySQL 5.7.12` untuk Aurora MySQL.
`serviceName`	string	Nama layanan. Saat ini, nilai ini selalu `Amazon Aurora MySQL` untuk Aurora MySQL.
`sessionId`	int	Pengidentifikasi sesi unik semu.
`startTime` (khusus rekam aktivitas basis data 1.1)	string	Waktu ketika eksekusi dimulai untuk pernyataan SQL. Disajikan dalam format Waktu Universal Terkoordinasi (UTC). Untuk menghitung waktu eksekusi pernyataan SQL, gunakan `endTime - startTime` . Lihat juga bidang `endTime`.

Field	Tipe data	Deskripsi
`statementId`	int	Pengidentifikasi untuk pernyataan SQL klien. Penghitung bertambah dengan setiap pernyataan SQL yang dimasukkan oleh klien. Penghitung dinolkan saat instans basis data dimulai ulang.
`substatementId`	int	Pengidentifikasi untuk subpernyataan SQL. Nilai ini adalah 1 untuk peristiwa dengan kelas `MAIN` dan 2 untuk peristiwa dengan kelas `AUX`. Gunakan bidang `statementId` untuk mengidentifikasi semua peristiwa yang dihasilkan oleh pernyataan yang sama.
`transactionId` (khusus rekam aktivitas basis data 1.2)	int	Pengidentifikasi untuk transaksi.
`type`	string	Jenis peristiwa. Nilai-nilai yang valid adalah `record` atau `heartbeat` .

Memproses aliran aktivitas database menggunakan AWS SDK

Anda dapat memproses aliran aktivitas secara terprogram menggunakan SDK AWS . Berikut adalah contoh Java dan Python yang berfungsi penuh tentang cara Anda dapat memproses aliran data Kinesis.

Java

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.net.InetAddress;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
```

```
import java.security.Security;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.zip.GZIPInputStream;

import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.spec.SecretKeySpec;

import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.encryptionsdk.AwsCrypto;
import com.amazonaws.encryptionsdk.CryptoInputStream;
import com.amazonaws.encryptionsdk.jce.JceMasterKey;
import
 com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ThrottlingException;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorFactory;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.InitialPositionInStream;
import
 com.amazonaws.services.kinesis.clientlibrary.lib.worker.KinesisClientLibConfiguration;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker.Builder;
import com.amazonaws.services.kinesis.model.Record;
import com.amazonaws.services.kms.AWSKMS;
import com.amazonaws.services.kms.AWSKMSClientBuilder;
import com.amazonaws.services.kms.model.DecryptRequest;
import com.amazonaws.services.kms.model.DecryptResult;
import com.amazonaws.util.Base64;
import com.amazonaws.util.IOUtils;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.annotations.SerializedName;
import org.bouncycastle.jce.provider.BouncyCastleProvider;
```

```
public class DemoConsumer {

    private static final String STREAM_NAME = "aws-rds-das-[cluster-external-
resource-id]";
    private static final String APPLICATION_NAME = "AnyApplication"; //unique
 application name for dynamo table generation that holds kinesis shard tracking
    private static final String AWS_ACCESS_KEY =
 "[AWS_ACCESS_KEY_TO_ACCESS_KINESIS]";
    private static final String AWS_SECRET_KEY =
 "[AWS_SECRET_KEY_TO_ACCESS_KINESIS]";
    private static final String DBC_RESOURCE_ID = "[cluster-external-resource-id]";
    private static final String REGION_NAME = "[region-name]"; //us-east-1, us-
east-2...
    private static final BasicAWSCredentials CREDENTIALS = new
 BasicAWSCredentials(AWS_ACCESS_KEY, AWS_SECRET_KEY);
    private static final AWSStaticCredentialsProvider CREDENTIALS_PROVIDER = new
 AWSStaticCredentialsProvider(CREDENTIALS);

    private static final AwsCrypto CRYPTO = new AwsCrypto();
    private static final AWSKMS KMS = AWSKMSClientBuilder.standard()
            .withRegion(REGION_NAME)
            .withCredentials(CREDENTIALS_PROVIDER).build();

    class Activity {
        String type;
        String version;
        String databaseActivityEvents;
        String key;
    }

    class ActivityEvent {
        @SerializedName("class") String _class;
        String clientApplication;
        String command;
        String commandText;
        String databaseName;
        String dbProtocol;
        String dbUserName;
        String endTime;
        String errorMessage;
        String exitCode;
        String logTime;
        String netProtocol;
        String objectName;
```

```
        String objectType;
        List<String> paramList;
        String pid;
        String remoteHost;
        String remotePort;
        String rowCount;
        String serverHost;
        String serverType;
        String serverVersion;
        String serviceName;
        String sessionId;
        String startTime;
        String statementId;
        String substatementId;
        String transactionId;
        String type;
    }

    class ActivityRecords {
        String type;
        String clusterId;
        String instanceId;
        List<ActivityEvent> databaseActivityEventList;
    }

    static class RecordProcessorFactory implements IRecordProcessorFactory {
        @Override
        public IRecordProcessor createProcessor() {
            return new RecordProcessor();
        }
    }

    static class RecordProcessor implements IRecordProcessor {

        private static final long BACKOFF_TIME_IN_MILLIS = 3000L;
        private static final int PROCESSING_RETRIES_MAX = 10;
        private static final long CHECKPOINT_INTERVAL_MILLIS = 60000L;
        private static final Gson GSON = new
GsonBuilder().serializeNulls().create();

        private static final Cipher CIPHER;
        static {
            Security.insertProviderAt(new BouncyCastleProvider(), 1);
            try {
```

```
                CIPHER = Cipher.getInstance("AES/GCM/NoPadding", "BC");
            } catch (NoSuchAlgorithmException | NoSuchPaddingException |
NoSuchProviderException e) {
                throw new ExceptionInInitializerError(e);
            }
        }

        private long nextCheckpointTimeInMillis;

        @Override
        public void initialize(String shardId) {
        }

        @Override
        public void processRecords(final List<Record> records, final
IRecordProcessorCheckpointer checkpointer) {
            for (final Record record : records) {
                processSingleBlob(record.getData());
            }

            if (System.currentTimeMillis() > nextCheckpointTimeInMillis) {
                checkpoint(checkpointer);
                nextCheckpointTimeInMillis = System.currentTimeMillis() +
CHECKPOINT_INTERVAL_MILLIS;
            }
        }

        @Override
        public void shutdown(IRecordProcessorCheckpointer checkpointer,
ShutdownReason reason) {
            if (reason == ShutdownReason.TERMINATE) {
                checkpoint(checkpointer);
            }
        }

        private void processSingleBlob(final ByteBuffer bytes) {
            try {
                // JSON $Activity
                final Activity activity = GSON.fromJson(new String(bytes.array(),
StandardCharsets.UTF_8), Activity.class);

                // Base64.Decode
                final byte[] decoded =
Base64.decode(activity.databaseActivityEvents);
```

```
                final byte[] decodedDataKey = Base64.decode(activity.key);

                Map<String, String> context = new HashMap<>();
                context.put("aws:rds:dbc-id", DBC_RESOURCE_ID);

                // Decrypt
                final DecryptRequest decryptRequest = new DecryptRequest()

.withCiphertextBlob(ByteBuffer.wrap(decodedDataKey)).withEncryptionContext(context);
                final DecryptResult decryptResult = KMS.decrypt(decryptRequest);
                final byte[] decrypted = decrypt(decoded,
getByteArray(decryptResult.getPlaintext()));

                // GZip Decompress
                final byte[] decompressed = decompress(decrypted);
                // JSON $ActivityRecords
                final ActivityRecords activityRecords = GSON.fromJson(new
String(decompressed, StandardCharsets.UTF_8), ActivityRecords.class);

                // Iterate throught $ActivityEvents
                for (final ActivityEvent event :
activityRecords.databaseActivityEventList) {
                    System.out.println(GSON.toJson(event));
                }
            } catch (Exception e) {
                // Handle error.
                e.printStackTrace();
            }
        }

        private static byte[] decompress(final byte[] src) throws IOException {
            ByteArrayInputStream byteArrayInputStream = new
ByteArrayInputStream(src);
            GZIPInputStream gzipInputStream = new
GZIPInputStream(byteArrayInputStream);
            return IOUtils.toByteArray(gzipInputStream);
        }

        private void checkpoint(IRecordProcessorCheckpointer checkpointer) {
            for (int i = 0; i < PROCESSING_RETRIES_MAX; i++) {
                try {
                    checkpointer.checkpoint();
                    break;
                } catch (ShutdownException se) {
```

```
                    // Ignore checkpoint if the processor instance has been shutdown
(fail over).
                    System.out.println("Caught shutdown exception, skipping
checkpoint." + se);
                    break;
                } catch (ThrottlingException e) {
                    // Backoff and re-attempt checkpoint upon transient failures
                    if (i >= (PROCESSING_RETRIES_MAX - 1)) {
                        System.out.println("Checkpoint failed after " + (i + 1) +
"attempts." + e);
                        break;
                    } else {
                        System.out.println("Transient issue when checkpointing -
attempt " + (i + 1) + " of " + PROCESSING_RETRIES_MAX + e);
                    }
                } catch (InvalidStateException e) {
                    // This indicates an issue with the DynamoDB table (check for
table, provisioned IOPS).
                    System.out.println("Cannot save checkpoint to the DynamoDB table
used by the Amazon Kinesis Client Library." + e);
                    break;
                }
                try {
                    Thread.sleep(BACKOFF_TIME_IN_MILLIS);
                } catch (InterruptedException e) {
                    System.out.println("Interrupted sleep" + e);
                }
            }
        }
    }

    private static byte[] decrypt(final byte[] decoded, final byte[] decodedDataKey)
throws IOException {
        // Create a JCE master key provider using the random key and an AES-GCM
encryption algorithm
        final JceMasterKey masterKey = JceMasterKey.getInstance(new
SecretKeySpec(decodedDataKey, "AES"),
                "BC", "DataKey", "AES/GCM/NoPadding");
        try (final CryptoInputStream<JceMasterKey> decryptingStream =
CRYPTO.createDecryptingStream(masterKey, new ByteArrayInputStream(decoded));
             final ByteArrayOutputStream out = new ByteArrayOutputStream()) {
            IOUtils.copy(decryptingStream, out);
            return out.toByteArray();
        }
```

```
    }

    public static void main(String[] args) throws Exception {
        final String workerId = InetAddress.getLocalHost().getCanonicalHostName() +
 ":" + UUID.randomUUID();
        final KinesisClientLibConfiguration kinesisClientLibConfiguration =
                new KinesisClientLibConfiguration(APPLICATION_NAME, STREAM_NAME,
 CREDENTIALS_PROVIDER, workerId);

 kinesisClientLibConfiguration.withInitialPositionInStream(InitialPositionInStream.LATEST);
        kinesisClientLibConfiguration.withRegionName(REGION_NAME);
        final Worker worker = new Builder()
                .recordProcessorFactory(new RecordProcessorFactory())
                .config(kinesisClientLibConfiguration)
                .build();

        System.out.printf("Running %s to process stream %s as worker %s...\n",
 APPLICATION_NAME, STREAM_NAME, workerId);

        try {
            worker.run();
        } catch (Throwable t) {
            System.err.println("Caught throwable while processing data.");
            t.printStackTrace();
            System.exit(1);
        }
        System.exit(0);
    }

    private static byte[] getByteArray(final ByteBuffer b) {
        byte[] byteArray = new byte[b.remaining()];
        b.get(byteArray);
        return byteArray;
    }
}
```

Python

```
import base64
import json
import zlib
import aws_encryption_sdk
from aws_encryption_sdk import CommitmentPolicy
```

```
from aws_encryption_sdk.internal.crypto import WrappingKey
from aws_encryption_sdk.key_providers.raw import RawMasterKeyProvider
from aws_encryption_sdk.identifiers import WrappingAlgorithm, EncryptionKeyType
import boto3

REGION_NAME = '<region>'                    # us-east-1
RESOURCE_ID = '<external-resource-id>'      # cluster-ABCD123456
STREAM_NAME = 'aws-rds-das-' + RESOURCE_ID  # aws-rds-das-cluster-ABCD123456

enc_client =
 aws_encryption_sdk.EncryptionSDKClient(commitment_policy=CommitmentPolicy.FORBID_ENCRYPT_AL

class MyRawMasterKeyProvider(RawMasterKeyProvider):
    provider_id = "BC"

    def __new__(cls, *args, **kwargs):
        obj = super(RawMasterKeyProvider, cls).__new__(cls)
        return obj

    def __init__(self, plain_key):
        RawMasterKeyProvider.__init__(self)
        self.wrapping_key =
 WrappingKey(wrapping_algorithm=WrappingAlgorithm.AES_256_GCM_IV12_TAG16_NO_PADDING,
                                        wrapping_key=plain_key,
 wrapping_key_type=EncryptionKeyType.SYMMETRIC)

    def _get_raw_key(self, key_id):
        return self.wrapping_key


def decrypt_payload(payload, data_key):
    my_key_provider = MyRawMasterKeyProvider(data_key)
    my_key_provider.add_master_key("DataKey")
    decrypted_plaintext, header = enc_client.decrypt(
        source=payload,

 materials_manager=aws_encryption_sdk.materials_managers.default.DefaultCryptoMaterialsManag
    return decrypted_plaintext


def decrypt_decompress(payload, key):
    decrypted = decrypt_payload(payload, key)
    return zlib.decompress(decrypted, zlib.MAX_WBITS + 16)
```

```
def main():
    session = boto3.session.Session()
    kms = session.client('kms', region_name=REGION_NAME)
    kinesis = session.client('kinesis', region_name=REGION_NAME)

    response = kinesis.describe_stream(StreamName=STREAM_NAME)
    shard_iters = []
    for shard in response['StreamDescription']['Shards']:
        shard_iter_response = kinesis.get_shard_iterator(StreamName=STREAM_NAME,
 ShardId=shard['ShardId'],

 ShardIteratorType='LATEST')
        shard_iters.append(shard_iter_response['ShardIterator'])

    while len(shard_iters) > 0:
        next_shard_iters = []
        for shard_iter in shard_iters:
            response = kinesis.get_records(ShardIterator=shard_iter, Limit=10000)
            for record in response['Records']:
                record_data = record['Data']
                record_data = json.loads(record_data)
                payload_decoded =
 base64.b64decode(record_data['databaseActivityEvents'])
                data_key_decoded = base64.b64decode(record_data['key'])
                data_key_decrypt_result =
 kms.decrypt(CiphertextBlob=data_key_decoded,

 EncryptionContext={'aws:rds:dbc-id': RESOURCE_ID})
                print (decrypt_decompress(payload_decoded,
 data_key_decrypt_result['Plaintext']))
            if 'NextShardIterator' in response:
                next_shard_iters.append(response['NextShardIterator'])
        shard_iters = next_shard_iters


if __name__ == '__main__':
    main()
```

Mengelola akses ke aliran aktivitas basis data

Setiap pengguna dengan hak akses peran AWS Identity and Access Management (IAM) yang sesuai untuk aliran aktivitas basis data dapat membuat, memulai, menghentikan, dan mengubah pengaturan aliran aktivitas untuk klaster basis data. Semua tindakan ini dimasukkan ke dalam log audit aliran. Untuk praktik kepatuhan terbaik, sebaiknya jangan berikan hak akses ini kepada DBA.

Anda mengatur akses ke aliran aktivitas basis data dengan menggunakan kebijakan IAM. Lihat informasi selengkapnya tentang autentikasi Aurora di Manajemen identitas dan akses untuk Amazon Aurora. Lihat informasi selengkapnya tentang pembuatan kebijakan IAM di Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM.

Example Kebijakan untuk memungkinkan konfigurasi aliran aktivitas basis data

Untuk memberi pengguna akses terperinci untuk mengubah aliran aktivitas, gunakan kunci-kunci konteks operasi khusus layanan `rds:StartActivityStream` dan `rds:StopActivityStream` dalam kebijakan IAM. Contoh kebijakan IAM berikut memungkinkan pengguna atau peran untuk mengonfigurasi aliran aktivitas.

```
{
   "Version":"2012-10-17",
   "Statement":[
        {
            "Sid":"ConfigureActivityStreams",
            "Effect":"Allow",
            "Action": [
                "rds:StartActivityStream",
                "rds:StopActivityStream"
            ],
            "Resource":"*",
          }
     ]
}
```

Example Kebijakan untuk memungkinkan dimulainya aliran aktivitas basis data

Contoh kebijakan IAM berikut memungkinkan pengguna atau peran untuk memulai aliran aktivitas.

```
{
    "Version":"2012-10-17",
```

```
    "Statement":[
        {
            "Sid":"AllowStartActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
    ]
}
```

Example Kebijakan untuk memungkinkan penghentian aliran aktivitas basis data

Contoh kebijakan IAM berikut memungkinkan pengguna atau peran untuk menghentikan aliran aktivitas.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowStopActivityStreams",
            "Effect":"Allow",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example Kebijakan untuk menolak dimulainya aliran aktivitas basis data

Contoh kebijakan IAM berikut mencegah pengguna atau peran memulai aliran aktivitas.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStartActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StartActivityStream",
            "Resource":"*"
        }
     ]
}
```

Example Kebijakan untuk menolak penghentian aliran aktivitas basis data

Contoh kebijakan IAM berikut mencegah pengguna atau peran menghentikan aliran aktivitas.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"DenyStopActivityStreams",
            "Effect":"Deny",
            "Action":"rds:StopActivityStream",
            "Resource":"*"
        }
    ]
}
```

Menggunakan Amazon Aurora MySQL

Amazon Aurora MySQL adalah mesin basis data relasional yang terkelola sepenuhnya dan kompatibel dengan MySQL. Mesin basis data ini menggabungkan kecepatan dan keandalan basis data komersial kelas atas dengan kesederhanaan dan efektivitas biaya basis data sumber terbuka. Aurora MySQL adalah pengganti "drop-in" untuk MySQL serta memudahkan dan menghemat biaya untuk menyiapkan, mengoperasikan, dan menskalakan deployment MySQL baru dan yang sudah ada, sehingga Anda bebas untuk fokus pada bisnis dan aplikasi Anda. Amazon RDS menyediakan administrasi untuk Aurora dengan menangani tugas basis data rutin seperti penyediaan, patching, pencadangan, pemulihan, deteksi kegagalan, dan perbaikan. Amazon RDS juga menyediakan alat migrasi yang memerlukan hanya beberapa klik untuk mengonversi aplikasi Amazon RDS for MySQL Anda yang ada menjadi aplikasi Aurora MySQL.

Topik

- Gambaran umum Amazon Aurora MySQL
- Keamanan dengan Amazon Aurora MySQL
- Memperbarui aplikasi untuk terhubung ke klaster DB Aurora MySQL menggunakan sertifikat TLS baru
- Menggunakan autentikasi Kerberos untuk Aurora MySQL
- Memigrasikan data ke klaster DB Amazon Aurora MySQL
- Mengelola Amazon Aurora MySQL
- Menyesuaikan Aurora MySQL
- Bekerja dengan kueri paralel untuk Amazon Aurora MySQL
- Menggunakan Audit Lanjutan dengan klaster DB Amazon Aurora MySQL
- Replikasi dengan Amazon Aurora MySQL
- Mengintegrasikan Amazon Aurora MySQL dengan layanan AWS lainnya
- Mode lab Amazon Aurora MySQL
- Praktik terbaik dengan Amazon Aurora MySQL
- Memecahkan masalah kinerja database MySQL Amazon Aurora
- Referensi Amazon Aurora MySQL
- Pembaruan mesin basis data untuk Amazon Aurora MySQL

Gambaran umum Amazon Aurora MySQL

Bagian berikut menyediakan gambaran umum Amazon Aurora MySQL.

Topik

- Peningkatan performa Amazon Aurora MySQL
- Amazon Aurora MySQL dan data spasial
- Aurora MySQL versi 3 yang kompatibel dengan MySQL 8.0
- Aurora MySQL versi 2 yang kompatibel dengan MySQL 5.7

Peningkatan performa Amazon Aurora MySQL

Amazon Aurora menyediakan peningkatan performa untuk mendukung beragam kebutuhan basis data komersial kelas atas.

Penyisipan cepat

Penyisipan cepat mempercepat penyisipan paralel yang diurutkan berdasarkan kunci primer dan berlaku khusus untuk pernyataan `LOAD DATA` dan `INSERT INTO ... SELECT ...`. Sisipkan cache dengan cepat ke posisi kursor dalam sebuah traversal indeks sambil mengeksekusi pernyataan. Hal ini membantu agar indeks tidak perlu di-traversing lagi.

Penyisipan cepat diaktifkan hanya untuk tabel InnoDB biasa di Aurora MySQL versi 3.03.2 dan lebih tinggi. Pengoptimalan ini tidak berfungsi untuk tabel sementara InnoDB. Ini dinonaktifkan di Aurora MySQL versi 2 untuk semua versi 2.11 dan 2.12. Pengoptimalan sisipan cepat hanya berfungsi jika optimasi Indeks Hash Adaptif dinonaktifkan.

Anda dapat memantau metrik berikut untuk menentukan efektivitas penyisipan cepat untuk klaster DB Anda:

- `aurora_fast_insert_cache_hits`: Penghitung yang bertambah saat kursor yang di-cache berhasil diambil dan diverifikasi.
- `aurora_fast_insert_cache_misses`: Penghitung yang bertambah ketika kursor yang di-cache tidak lagi valid dan Aurora melakukan traversal indeks normal.

Anda dapat mengambil nilai saat ini dari metrik penyisipan cepat dengan menggunakan perintah berikut:

```
mysql> show global status like 'Aurora_fast_insert%';
```

Anda akan mendapatkan output seperti yang berikut ini:

```
+--------------------------------+-----------+
| Variable_name                  | Value     |
+--------------------------------+-----------+
| Aurora_fast_insert_cache_hits  | 3598300   |
| Aurora_fast_insert_cache_misses | 436401336 |
+--------------------------------+-----------+
```

Amazon Aurora MySQL dan data spasial

Daftar berikut merangkum fitur spasial Aurora MySQL utama dan menjelaskan bagaimana fitur tersebut terkait dengan fitur spasial di MySQL:

- Aurora MySQL versi 2 mendukung jenis data spasial dan fungsi relasi spasial yang sama dengan MySQL 5.7. Untuk informasi selengkapnya tentang jenis dan fungsi data ini, lihat Spatial Data Types dan Spatial Relation Functions dalam dokumentasi MySQL 5.7.
- Aurora MySQL versi 3 mendukung jenis data spasial dan fungsi relasi spasial yang sama dengan MySQL 8.0. Untuk informasi selengkapnya tentang jenis dan fungsi data ini, lihat Spatial Data Types dan Spatial Relation Functions dalam dokumentasi MySQL 8.0.
- Aurora MySQL mendukung pengindeksan spasial pada tabel InnoDB. Pengindeksan spasial meningkatkan performa kueri pada set data besar untuk kueri pada data spasial. Di MySQL, pengindeksan spasial untuk tabel InnoDB tersedia di MySQL 5.7 dan 8.0.

  Aurora MySQL menggunakan strategi pengindeksan spasial yang berbeda dari MySQL untuk performa tinggi dengan kueri spasial. Implementasi indeks spasial Aurora menggunakan kurva pengisian ruang pada B-tree, yang ditujukan untuk memberikan performa yang lebih tinggi untuk pemindaian rentang spasial daripada R-tree.

  > **Note**
  >
  > Di Aurora MySQL, transaksi pada tabel dengan indeks spasial yang ditentukan pada kolom dengan pengidentifikasi referensi spasial (SRID) tidak dapat melakukan penyisipan ke area yang dipilih untuk diperbarui oleh transaksi lain.

Pernyataan bahasa definisi data (DDL) berikut didukung untuk membuat indeks pada kolom yang menggunakan jenis data spasial.

CREATE TABLE

Anda dapat menggunakan kata kunci `SPATIAL INDEX` dalam pernyataan `CREATE TABLE` untuk menambahkan indeks spasial ke kolom di tabel baru. Berikut adalah contohnya.

```
CREATE TABLE test (shape POLYGON NOT NULL, SPATIAL INDEX(shape));
```

ALTER TABLE

Anda dapat menggunakan kata kunci `SPATIAL INDEX` dalam pernyataan `ALTER TABLE` untuk menambahkan indeks spasial ke kolom dalam tabel yang ada. Berikut adalah contohnya.

```
ALTER TABLE test ADD SPATIAL INDEX(shape);
```

CREATE INDEX

Anda dapat menggunakan kata kunci `SPATIAL` dalam pernyataan `CREATE INDEX` untuk menambahkan indeks spasial ke kolom dalam tabel yang ada. Berikut adalah contohnya.

```
CREATE SPATIAL INDEX shape_index ON test (shape);
```

Aurora MySQL versi 3 yang kompatibel dengan MySQL 8.0

Anda dapat menggunakan Aurora MySQL versi 3 untuk mendapatkan fitur terbaru yang kompatibel dengan MySQL, peningkatan performa, dan perbaikan bug. Di bagian berikut ini, Anda dapat mempelajari tentang Aurora MySQL versi 3, dengan kompatibilitas MySQL 8.0. Anda dapat mempelajari cara meningkatkan klaster dan aplikasi Anda ke Aurora MySQL versi 3.

Beberapa fitur Aurora, seperti Aurora Serverless v2, memerlukan Aurora MySQL versi 3.

Topik

- Fitur dari MySQL 8.0 Community Edition
- Prasyarat Aurora MySQL versi 3 untuk Aurora MySQL Nirserver v2
- Catatan rilis untuk Aurora MySQL versi 3

- Optimisasi kueri paralel baru
- Optimisasi untuk mengurangi waktu pengaktifan ulang basis data.
- Perilaku tabel sementara baru di Aurora MySQL versi 3
- Perbandingan Aurora MySQL versi 2 dan Aurora MySQL versi 3
- Perbandingan Aurora MySQL versi 3 dan MySQL 8.0 Community Edition
- Meningkatkan ke Aurora MySQL versi 3

Fitur dari MySQL 8.0 Community Edition

Rilis awal Aurora MySQL versi 3 kompatibel dengan MySQL 8.0.23 Community Edition. MySQL 8.0 memperkenalkan beberapa fitur baru, termasuk yang berikut:

- Fungsi JSON. Untuk informasi penggunaan, lihat JSON Functions dalam Panduan Referensi MySQL.
- Fungsi Jendela. Untuk informasi penggunaan, lihat Window Functions dalam Panduan Referensi MySQL.
- Ekspresi tabel umum (CTE), yang menggunakan klausa `WITH`. Untuk informasi penggunaan, lihat WITH (Common Table Expressions) dalam Panduan Referensi MySQL.
- Klausa `ADD COLUMN` dan `RENAME COLUMN` yang dioptimalkan untuk pernyataan `ALTER TABLE`. Optimisasi ini disebut “DDL instan”. Aurora MySQL versi 3 kompatibel dengan fitur DDL instan MySQL komunitas. Fitur DDL cepat Aurora sebelumnya tidak digunakan. Untuk informasi penggunaan untuk DDL instan, lihat DDL instan (Aurora MySQL versi 3).
- Indeks menurun, fungsional, dan tidak terlihat. Untuk informasi penggunaan, lihat Invisible Indexes, Descending Indexes, dan CREATE INDEX Statement dalam Panduan Referensi MySQL.
- Hak akses berbasis peran yang dikontrol melalui pernyataan SQL. Untuk informasi selengkapnya tentang perubahan pada model hak akses, lihat Model hak akses berbasis peran.
- Klausa `NOWAIT` dan `SKIP LOCKED` dengan pernyataan `SELECT ... FOR SHARE`. Klausa ini menghindari tindakan menunggu transaksi lain untuk membuka kunci baris. Untuk informasi penggunaan, lihat Locking Reads dalam Panduan Referensi MySQL.
- Peningkatan pada replikasi log biner (binlog). Untuk detail Aurora MySQL, lihat Replikasi log biner. Khususnya, Anda dapat melakukan replikasi yang difilter. Untuk informasi penggunaan tentang replikasi yang difilter, lihat How Servers Evaluate Replication Filtering Rules dalam Panduan Referensi MySQL.

- Petunjuk. Beberapa petunjuk yang kompatibel dengan MySQL 8.0 sudah di-backport ke Aurora MySQL versi 2. Untuk informasi tentang menggunakan petunjuk dengan Aurora MySQL, lihat Petunjuk Aurora MySQL. Untuk daftar lengkap petunjuk di MySQL 8.0 komunitas, lihat Optimizer Hints dalam Panduan Referensi MySQL.

Untuk daftar lengkap fitur yang ditambahkan ke MySQL 8.0 edisi komunitas, lihat postingan blog Daftar lengkap fitur baru di MySQL 8.0.

Aurora MySQL versi 3 juga mencakup perubahan kata kunci untuk bahasa inklusif, yang di-backport dari MySQL 8.0.26 komunitas. Untuk detail tentang perubahan tersebut, lihat Perubahan bahasa inklusif untuk Aurora MySQL versi 3.

Prasyarat Aurora MySQL versi 3 untuk Aurora MySQL Nirserver v2

Aurora MySQL versi 3 adalah prasyarat untuk semua instans DB di klaster Aurora MySQL Nirserver v2. Aurora MySQL Nirserver v2 mencakup dukungan untuk instans pembaca dalam klaster DB, dan fitur Aurora lainnya yang tidak tersedia untuk Aurora MySQL Nirserver v1. Layanan ini juga memiliki penskalaan yang lebih cepat dan lebih granular daripada Aurora MySQL Nirserver v1.

Catatan rilis untuk Aurora MySQL versi 3

Untuk catatan rilis untuk semua rilis Aurora MySQL versi 3, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 3 dalam Catatan Rilis untuk Aurora MySQL.

Optimisasi kueri paralel baru

Optimisasi kueri pararel Aurora sekarang berlaku untuk lebih banyak operasi SQL:

- Kueri pararel sekarang berlaku untuk tabel yang berisi jenis data `TEXT`, `BLOB`, `JSON`, `GEOMETRY`, serta `VARCHAR` dan `CHAR` yang lebih panjang dari 768 byte.
- Kueri paralel dapat mengoptimalkan kueri yang memerlukan tabel yang dipartisi.
- Kueri paralel dapat mengoptimalkan kueri yang memerlukan panggilan fungsi agregat dalam daftar pilih dan klausa `HAVING`.

Untuk informasi selengkapnya tentang peningkatan ini, lihat Meningkatkan klaster kueri paralel ke Aurora MySQL versi 3. Untuk informasi umum tentang kueri paralel Aurora, lihat Bekerja dengan kueri paralel untuk Amazon Aurora MySQL.

Optimisasi untuk mengurangi waktu pengaktifan ulang basis data.

Klaster DB Aurora MySQL Anda harus memiliki ketersediaan tinggi selama pemadaman yang direncanakan dan tidak direncanakan.

Administrator basis data perlu melakukan pemeliharaan basis data sesekali. Pemeliharaan ini mencakup patching basis data, tingkatkan, modifikasi parameter basis data yang memerlukan boot ulang manual, pelaksanaan failover untuk mengurangi waktu yang diperlukan dalam mengubah kelas instans, dan sebagainya. Tindakan yang direncanakan ini membutuhkan waktu henti.

Namun, waktu henti juga dapat disebabkan oleh tindakan yang tidak direncanakan, seperti failover yang tidak terduga karena kesalahan perangkat keras yang mendasarinya atau throttling sumber daya basis data. Semua tindakan yang direncanakan dan tidak direncanakan ini mengakibatkan pengaktifan ulang basis data.

Di Aurora MySQL versi 3.05 dan lebih tinggi, kami telah memperkenalkan optimisasi yang mengurangi waktu pengaktifan ulang basis data. Optimisasi ini memberikan waktu henti hingga 65% lebih sedikit daripada tanpa optimisasi, dan lebih sedikit gangguan pada beban kerja basis data Anda, setelah pengaktifan ulang.

Selama pengaktifan basis data, banyak komponen memori internal yang diinisialisasi. Yang terbesar adalah pool buffer InnoDB, yang di Aurora MySQL adalah 75% dari ukuran memori instans secara default. Pengujian kami telah menemukan bahwa waktu inisialisasinya sebanding dengan ukuran pool buffer InnoDB, dan oleh karena itu, akan diskalakan dengan ukuran kelas instans DB. Selama tahap inisialisasi ini, basis data tidak dapat menerima koneksi, sehingga memperpanjang waktu henti selama pengaktifan ulang. Tahap pertama dari pengaktifan ulang cepat Aurora MySQL akan mengoptimalkan inisialisasi pool buffer, yang mengurangi waktu untuk inisialisasi basis data dan dengan demikian akan mengurangi waktu pengaktifan ulang secara keseluruhan.

Untuk detail selengkapnya, lihat blog Kurangi waktu henti dengan optimisasi waktu pengaktifan ulang basis data Amazon Aurora MySQL.

Perilaku tabel sementara baru di Aurora MySQL versi 3

Aurora MySQL versi 3 menangani tabel sementara secara berbeda dari Aurora MySQL versi sebelumnya. Perilaku baru ini diwarisi dari MySQL 8.0 Community Edition. Berikut dua jenis tabel sementara yang dapat dibuat dengan Aurora MySQL versi 3:

- Tabel sementara internal (atau implisit) — Dibuat oleh mesin Aurora MySQL untuk menangani operasi seperti pengurutan agregasi, tabel turunan, atau ekspresi tabel umum (CTE).

- Tabel sementara yang dibuat pengguna (atau eksplisit) — Dibuat oleh mesin Aurora MySQL saat Anda menggunakan pernyataan `CREATE TEMPORARY TABLE`.

Terdapat pertimbangan tambahan untuk tabel sementara internal dan yang dibuat pengguna pada instans DB pembaca Aurora. Kita akan membahas perubahan ini di bagian berikut.

Topik

- Mesin penyimpanan untuk tabel sementara internal (implisit)
- Membatasi ukuran tabel sementara internal dalam memori
- Mengurangi masalah kepenuhan untuk tabel sementara internal di Aurora Replicas
- Tabel sementara yang dibuat pengguna (eksplisit) pada instans DB pembaca
- Kesalahan dan mitigasi pembuatan tabel sementara

Mesin penyimpanan untuk tabel sementara internal (implisit)

Saat membuat set hasil perantara, Aurora MySQL awalnya mencoba menulis ke tabel sementara dalam memori. Langkah ini mungkin tidak berhasil karena tipe data yang tidak kompatibel atau batas yang dikonfigurasi. Jika demikian, tabel sementara dikonversi ke tabel sementara di disk, bukan disimpan di memori. Informasi selengkapnya tentang hal ini dapat ditemukan di Penggunaan Tabel Sementara Internal di MySQL dalam dokumentasi MySQL.

Di Aurora MySQL versi 3, cara kerja tabel sementara internal berbeda dengan Aurora MySQL versi sebelumnya. Alih-alih memilih antara mesin penyimpanan InnoDB dan MyISAM untuk tabel sementara tersebut, sekarang Anda memilih antara mesin penyimpanan `TempTable` dan InnoDB.

Dengan mesin penyimpanan `TempTable`, Anda dapat membuat pilihan tambahan untuk cara menangani data tertentu. Data yang terpengaruh memenuhi kumpulan memori yang menampung semua tabel sementara internal untuk instans DB.

Pilihan tersebut dapat memengaruhi performa kueri yang menghasilkan volume data sementara yang tinggi, misalnya saat melakukan agregasi seperti `GROUP BY` pada tabel besar.

> **Tip**
>
> Jika beban kerja Anda mencakup kueri yang menghasilkan tabel sementara internal, konfirmasikan performa aplikasi Anda dengan perubahan ini melalui upaya menjalankan tolok ukur dan memantau metrik terkait performa.

Dalam kasus tertentu, jumlah data sementara sesuai dengan kumpulan memori `TempTable` atau hanya memenuhi kumpulan memori dalam jumlah kecil. Jika demikian, sebaiknya gunakan pengaturan `TempTable` untuk tabel sementara internal dan file yang dipetakan memori untuk menyimpan data overflow apa pun. Ini adalah pengaturan default.

Mesin penyimpanan `TempTable` adalah pengaturan default. `TempTable` menggunakan kumpulan memori umum untuk semua tabel sementara yang menggunakan mesin ini, bukan batas memori maksimum per tabel. Ukuran kumpulan memori ini ditentukan oleh parameter temptable_max_ram. Nilai default adalah 1 GiB pada instans DB dengan memori 16 GiB atau lebih, dan 16 MB pada instans DB dengan memori kurang dari 16 GiB. Ukuran kumpulan memori memengaruhi konsumsi memori tingkat sesi.

Dalam kasus tertentu, bila Anda menggunakan mesin penyimpanan `TempTable`, data sementara mungkin melebihi ukuran kumpulan memori. Jika demikian, Aurora MySQL akan menyimpan data overflow tersebut menggunakan mekanisme sekunder.

Anda dapat mengatur parameter temptable_max_mmap untuk memilih apakah data akan memenuhi file sementara yang dipetakan memori atau tabel sementara internal InnoDB di disk. Format data yang berbeda dan kriteria overflow dari mekanisme overflow ini dapat memengaruhi performa kueri. Hal ini dilakukan dengan memengaruhi jumlah data yang ditulis ke disk dan permintaan atas throughput penyimpanan disk.

Aurora MySQL menyimpan data overflow secara berbeda tergantung pada pilihan tujuan overflow data Anda dan apakah kueri berjalan pada instans DB penulis atau pembaca:

- Pada instans penulis, data yang memenuhi tabel sementara internal InnoDB disimpan dalam volume klaster Aurora.
- Pada instans penulis, data yang memenuhi file sementara yang dipetakan memori disimpan di penyimpanan lokal pada instans Aurora MySQL versi 3.
- Pada instans pembaca, data overflow selalu berada pada file sementara yang dipetakan memori di penyimpanan lokal. Hal ini karena instans hanya baca tidak dapat menyimpan data apa pun pada volume klaster Aurora.

Parameter konfigurasi terkait tabel sementara internal berlaku untuk instans penulis dan pembaca dengan cara yang berbeda di klaster Anda:

- Pada instans pembaca, Aurora MySQL selalu menggunakan mesin penyimpanan `TempTable`.

- Ukuran default `temptable_max_mmap` adalah 1 GiB untuk instans penulis dan pembaca, berapa pun ukuran memori instans DB-nya. Anda dapat menyesuaikan nilai ini pada instans penulis dan pembaca.
- Mengatur `temptable_max_mmap` ke `0` akan menonaktifkan penggunaan file sementara yang dipetakan memori pada instans penulis.
- Anda tidak dapat mengatur `temptable_max_mmap` ke `0` pada instans pembaca.

> **Note**
>
> Kami tidak menyarankan penggunaan parameter temptable_use_mmap. Parameter tersebut telah dihentikan, dan dukungannya direncanakan akan dihapus dalam rilis MySQL mendatang.

Membatasi ukuran tabel sementara internal dalam memori

Sebagaimana dibahas dalam Mesin penyimpanan untuk tabel sementara internal (implisit), Anda dapat mengontrol sumber daya tabel sementara secara global dengan menggunakan pengaturan temptable_max_ram dan temptable_max_mmap.

Anda juga dapat membatasi ukuran setiap tabel sementara internal dalam memori menggunakan parameter DB tmp_table_size. Batas ini dimaksudkan untuk mencegah setiap kueri mengonsumsi sumber daya tabel sementara global dalam jumlah besar, yang dapat memengaruhi performa kueri bersamaan yang memerlukan sumber daya ini.

Parameter `tmp_table_size` menetapkan ukuran maksimum tabel sementara yang dibuat oleh mesin penyimpanan `MEMORY` di Aurora MySQL versi 3.

Di Aurora MySQL versi 3.04 dan yang lebih tinggi, `tmp_table_size` juga menetapkan ukuran maksimum tabel sementara yang dibuat oleh mesin penyimpanan `TempTable` saat parameter DB `aurora_tmptable_enable_per_table_limit` diatur ke `ON`. Perilaku ini dinonaktifkan secara default (`OFF`), yang merupakan perilaku serupa seperti di Aurora MySQL versi 3.03 dan yang lebih rendah.

- Bila `aurora_tmptable_enable_per_table_limit` diatur ke `OFF`, `tmp_table_size` tidak dipertimbangkan untuk tabel sementara internal dalam memori yang dibuat oleh mesin penyimpanan `TempTable`.

Namun, batas sumber daya `TempTable` global masih berlaku. Bila batas sumber daya `TempTable` global tercapai, Aurora MySQL memiliki perilaku berikut:

- Instans DB penulis – Aurora MySQL secara otomatis mengonversi tabel sementara dalam memori menjadi tabel sementara di disk InnoDB.
- Instans DB pembaca – Kueri berakhir dengan kesalahan.

```
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
```

- Bila `aurora_tmptable_enable_per_table_limit` diatur ke `ON`, Aurora MySQL memiliki perilaku berikut saat batas `tmp_table_size` tercapai:
  - Instans DB penulis – Aurora MySQL secara otomatis mengonversi tabel sementara dalam memori menjadi tabel sementara di disk InnoDB.
  - Instans DB pembaca – Kueri berakhir dengan kesalahan.

    ```
    ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlxx_xxx' is full
    ```

    Dalam kasus ini, batas sumber daya `TempTable` global dan batas per tabel berlaku.

> **Note**
>
> Parameter `aurora_tmptable_enable_per_table_limit` tidak berpengaruh saat internal_tmp_mem_storage_engine diatur ke `MEMORY`. Dalam hal ini, ukuran maksimum tabel sementara dalam memori ditentukan oleh nilai tmp_table_size atau max_heap_table_size, mana pun yang lebih kecil.

Contoh berikut menunjukkan perilaku parameter `aurora_tmptable_enable_per_table_limit` untuk instans DB penulis dan pembaca.

Example instans DB penulis dengan **`aurora_tmptable_enable_per_table_limit`** diatur ke **`OFF`**

Tabel sementara dalam memori tidak dikonversi ke tabel sementara di disk InnoDB.

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
|                  0 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
+----------+
| max(n)   |
+----------+
| 60000000 |
+----------+
1 row in set (13.99 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Example instans DB penulis dengan **aurora_tmptable_enable_per_table_limit** diatur ke **ON**

Tabel sementara dalam memori dikonversi ke tabel sementara di disk InnoDB.

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
+--------------------+-----------------+-----------------------------------------
+------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+--------------------+-----------------+-----------------------------------------
+------------------+
|                  0 | 3.04.0          |                                        1 |
      16777216 |
+--------------------+-----------------+-----------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
+-------------------------+-------+
| Created_tmp_disk_tables | 0     |
+-------------------------+-------+
1 row in set (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
+---------+
| max(n)  |
+---------+
| 6000000 |
+---------+
1 row in set (4.10 sec)

mysql> show status like '%created_tmp_disk%';
+-------------------------+-------+
| Variable_name           | Value |
```

```
+-------------------------+-------+
| Created_tmp_disk_tables | 1     |
+-------------------------+-------+
1 row in set (0.00 sec)
```

Example instans DB pembaca dengan **aurora_tmptable_enable_per_table_limit** diatur ke **OFF**

Kueri selesai tanpa kesalahan karena `tmp_table_size` tidak berlaku dan batas sumber daya `TempTable` global belum tercapai.

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
|                  1 | 3.04.0           |                                       0 |
       1073741824 |           1073741824 |
+--------------------+------------------+-----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 60000000) SELECT max(n) FROM cte;
+----------+
| max(n)   |
+----------+
| 60000000 |
+----------+
1 row in set (14.05 sec)
```

Example instans DB pembaca dengan **aurora_tmptable_enable_per_table_limit** diatur ke **OFF**

Kueri ini mencapai batas sumber daya TempTable global dengan `aurora_tmptable_enable_per_table_limit` diatur ke OFF. Kueri berakhir dengan kesalahan pada instans pembaca.

```
mysql> set aurora_tmptable_enable_per_table_limit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@temptable_max_r
+--------------------+------------------+----------------------------------------
+---------------------+----------------------+
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@temptable_max_ram | @@temptable_max_mmap |
+--------------------+------------------+----------------------------------------
+---------------------+----------------------+
|                  1 | 3.04.0           |                                        0 |
       1073741824 |           1073741824 |
+--------------------+------------------+----------------------------------------
+---------------------+----------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.01 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 120000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_1586_2' is full
```

Example instans DB pembaca dengan **aurora_tmptable_enable_per_table_limit** diatur ke **ON**

Kueri berakhir dengan kesalahan saat batas `tmp_table_size` tercapai.

```
mysql> set aurora_tmptable_enable_per_table_limit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> select
 @@innodb_read_only,@@aurora_version,@@aurora_tmptable_enable_per_table_limit,@@tmp_table_size;
+--------------------+------------------+----------------------------------------
+------------------+
```

```
| @@innodb_read_only | @@aurora_version | @@aurora_tmptable_enable_per_table_limit |
 @@tmp_table_size |
+--------------------+------------------+------------------------------------------
+------------------+
|                  1 | 3.04.0           |                                        1 |
      16777216 |
+--------------------+------------------+------------------------------------------
+------------------+
1 row in set (0.00 sec)

mysql> set cte_max_recursion_depth=4294967295;
Query OK, 0 rows affected (0.00 sec)

mysql> WITH RECURSIVE cte (n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM cte WHERE n <
 6000000) SELECT max(n) FROM cte;
ERROR 1114 (HY000): The table '/rdsdbdata/tmp/#sqlfd_8_2' is full
```

Mengurangi masalah kepenuhan untuk tabel sementara internal di Aurora Replicas

Untuk mencegah masalah pembatasan ukuran pada tabel sementara, atur parameter `temptable_max_ram` dan `temptable_max_mmap` ke nilai gabungan yang dapat memenuhi persyaratan beban kerja Anda.

Berhati-hatilah saat mengatur nilai parameter `temptable_max_ram`. Mengatur nilai terlalu tinggi akan mengurangi memori yang tersedia pada instans basis data, yang dapat menyebabkan kondisi kehabisan memori. Pantau memori rata-rata yang dapat dikosongkan pada instans DB. Lalu, tentukan nilai yang sesuai untuk `temptable_max_ram`, sehingga Anda akan tetap memiliki memori bebas dalam jumlah wajar yang tersisa pada instans. Untuk informasi selengkapnya, lihat Masalah memori yang dapat dikosongkan di Amazon Aurora.

Penting juga bagi Anda untuk memantau ukuran penyimpanan lokal dan konsumsi ruang tabel sementara. Untuk informasi selengkapnya tentang pemantauan penyimpanan lokal pada instans, lihat artikel Pusat Pengetahuan AWS, Apa yang disimpan di penyimpanan lokal yang kompatibel dengan Aurora MySQL, dan bagaimana saya dapat memecahkan masalah penyimpanan lokal?.

> **Note**
>
> Prosedur ini tidak berfungsi bila parameter `aurora_tmptable_enable_per_table_limit` diatur ke `ON`. Untuk informasi selengkapnya, lihat Membatasi ukuran tabel sementara internal dalam memori.

Example 1

Anda tahu bahwa tabel sementara Anda berkembang menjadi berukuran kumulatif 20 GiB. Anda ingin mengatur tabel sementara dalam memori ke 2 GiB dan mengembangkannya menjadi maksimum 20 GiB pada disk.

Atur `temptable_max_ram` ke **`2,147,483,648`** dan `temptable_max_mmap` ke **`21,474,836,480`**. Nilai ini dihitung dalam byte.

Pengaturan parameter ini memastikan bahwa tabel sementara Anda dapat berkembang menjadi total kumulatif 22 GiB.

Example 2

Ukuran instans Anda saat ini adalah 16xlarge atau lebih besar. Anda tidak mengetahui ukuran total tabel sementara yang mungkin Anda perlukan. Anda ingin dapat menggunakan hingga 4 GiB dalam memori dan hingga ukuran penyimpanan maksimum yang tersedia pada disk.

Atur `temptable_max_ram` ke **`4,294,967,296`** dan `temptable_max_mmap` ke **`1,099,511,627,776`**. Nilai ini dihitung dalam byte.

Di sini, Anda mengatur `temptable_max_mmap` ke 1 TiB, yang lebih kecil dari penyimpanan lokal maksimum sebesar 1,2 TiB pada instans DB Aurora 16xlarge.

Pada ukuran instans yang lebih kecil, sesuaikan nilai `temptable_max_mmap` agar tidak mengisi penyimpanan lokal yang tersedia. Misalnya, instans 2xlarge hanya memiliki penyimpanan lokal yang tersedia sebesar 160 GiB. Karena itu, sebaiknya atur nilainya menjadi kurang dari 160 GiB. Untuk informasi selengkapnya tentang penyimpanan lokal yang tersedia untuk ukuran instans DB, lihat Batas penyimpanan sementara untuk Aurora MySQL.

Tabel sementara yang dibuat pengguna (eksplisit) pada instans DB pembaca

Anda dapat membuat tabel sementara eksplisit menggunakan kata kunci `TEMPORARY` dalam pernyataan `CREATE TABLE`. Tabel sementara eksplisit didukung pada instans DB penulis di klaster DB Aurora. Anda juga dapat menggunakan tabel sementara eksplisit pada instans DB pembaca, tetapi tabel tidak dapat menerapkan penggunaan mesin penyimpanan InnoDB.

Untuk mencegah timbulnya kesalahan saat membuat tabel sementara eksplisit pada instans DB pembaca Aurora MySQL, pastikan Anda menjalankan semua pernyataan `CREATE TEMPORARY TABLE` dengan salah satu atau kedua cara berikut:

- Jangan tentukan klausa `ENGINE=InnoDB`.

- Jangan atur mode SQL ke NO_ENGINE_SUBSTITUTION.

Kesalahan dan mitigasi pembuatan tabel sementara

Kesalahan yang Anda terima berbeda tergantung pada apakah Anda menggunakan pernyataan CREATE TEMPORARY TABLE biasa atau variasi CREATE TEMPORARY TABLE AS SELECT. Contoh berikut menunjukkan berbagai jenis kesalahan.

Perilaku tabel sementara ini hanya berlaku untuk instans hanya-baca. Contoh pertama ini mengonfirmasi jenis instans yang terhubung dengan sesi.

```
mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
```

Untuk pernyataan CREATE TEMPORARY TABLE biasa, pernyataan akan gagal saat mode SQL NO_ENGINE_SUBSTITUTION diaktifkan. Ketika NO_ENGINE_SUBSTITUTION dinonaktifkan (default), substitusi mesin yang sesuai dibuat, dan pembuatan tabel sementara berhasil.

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql>  CREATE TEMPORARY TABLE tt2 (id int) ENGINE=InnoDB;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> CREATE TEMPORARY TABLE tt4 (id int) ENGINE=InnoDB;

mysql> SHOW CREATE TABLE tt4\G
*************************** 1. row ***************************
       Table: tt4
Create Table: CREATE TEMPORARY TABLE `tt4` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Untuk pernyataan CREATE TEMPORARY TABLE AS SELECT, pernyataan akan gagal saat mode SQL NO_ENGINE_SUBSTITUTION diaktifkan. Ketika NO_ENGINE_SUBSTITUTION dinonaktifkan (default), substitusi mesin yang sesuai dibuat, dan pembuatan tabel sementara berhasil.

```
mysql> set sql_mode = 'NO_ENGINE_SUBSTITUTION';

mysql> CREATE TEMPORARY TABLE tt1 ENGINE=InnoDB AS SELECT * FROM t1;
ERROR 3161 (HY000): Storage engine InnoDB is disabled (Table creation is disallowed).

mysql> SET sql_mode = '';

mysql> show create table tt3;
+-------+-----------------------------------------------------------------+
| Table | Create Table                                                    |
+-------+-----------------------------------------------------------------+
| tt3   | CREATE TEMPORARY TABLE `tt3` (
  `id` int DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci |
+-------+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

Untuk informasi selengkapnya tentang aspek penyimpanan dan implikasi performa tabel sementara di Aurora MySQL versi 3, lihat postingan blog Menggunakan mesin penyimpanan TempTable di Amazon RDS for MySQL dan Amazon Aurora MySQL.

Perbandingan Aurora MySQL versi 2 dan Aurora MySQL versi 3

Gunakan yang berikut ini untuk mempelajari tentang perubahan yang harus diperhatikan saat Anda meningkatkan klaster Aurora MySQL versi 2 ke versi 3.

Topik

- Perbedaan fitur antara Aurora MySQL versi 2 dan 3
- Dukungan kelas instans
- Perubahan parameter untuk Aurora MySQL versi 3
- Variabel status
- Perubahan bahasa inklusif untuk Aurora MySQL versi 3
- Nilai AUTO_INCREMENT
- Replikasi log biner

Perbedaan fitur antara Aurora MySQL versi 2 dan 3

Fitur Amazon Aurora MySQL berikut didukung di Aurora MySQL for MySQL 5.7, tetapi fitur ini tidak didukung di Aurora MySQL for MySQL 8.0:

- Anda tidak dapat menggunakan Aurora MySQL versi 3 untuk klaster Aurora Serverless v1. Aurora MySQL versi 3 berfungsi dengan Aurora Serverless v2.
- Mode lab tidak berlaku untuk Aurora MySQL versi 3. Tidak ada fitur mode lab apa pun di Aurora MySQL versi 3. DDL instan menggantikan fitur DDL online cepat yang sebelumnya tersedia dalam mode lab. Sebagai contoh, lihat DDL instan (Aurora MySQL versi 3).
- Cache kueri dihapus dari MySQL 8.0 komunitas dan juga dari Aurora MySQL versi 3.
- Aurora MySQL versi 3 kompatibel dengan fitur hash join MySQL komunitas. Implementasi khusus Aurora untuk hash join di Aurora MySQL versi 2 tidak digunakan. Untuk informasi tentang menggunakan hash join dengan kueri paralel Aurora, lihat Mengaktifkan hash join untuk klaster kueri paralel dan Petunjuk Aurora MySQL. Untuk informasi penggunaan umum tentang hash join, lihat Hash Join Optimization dalam Panduan Referensi MySQL.
- Prosedur tersimpan `mysql.lambda_async` yang sudah dihentikan di Aurora MySQL versi 2 dihapus di versi 3. Untuk versi 3, gunakan fungsi asinkron `lambda_async` sebagai gantinya.
- Set karakter default di Aurora MySQL versi 3 adalah `utf8mb4`. Di Aurora MySQL versi 2, set karakter default adalah `latin1`. Untuk informasi tentang set karakter ini, lihat The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding) dalam Panduan Referensi MySQL.

Beberapa fitur Aurora MySQL tersedia untuk kombinasi tertentu dari AWS Region dan versi mesin DB. Untuk detailnya, lihat Fitur yang didukung di Amazon Aurora oleh Wilayah AWS dan mesin DB Aurora.

Dukungan kelas instans

Aurora MySQL versi 3 mendukung set kelas instans yang berbeda dari Aurora MySQL versi 2:

- Untuk instans yang lebih besar, Anda dapat menggunakan kelas instans modern seperti `db.r5`, `db.r6g`, dan `db.x2g`.
- Untuk instans yang lebih kecil, Anda dapat menggunakan kelas instans modern seperti `db.t3` dan `db.t4g`.

> **Note**
>
> Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Menggunakan kelas instans T untuk pengembangan dan pengujian.

Kelas instans berikut dari Aurora MySQL versi 2 tidak tersedia untuk Aurora MySQL versi 3:

- `db.r4`
- `db.r3`
- `db.t3.small`
- `db.t2`

Periksa skrip administrasi Anda untuk menemukan pernyataan CLI yang membuat instans DB Aurora MySQL. Lakukan hard-coding terhadap nama kelas instans yang tidak tersedia untuk Aurora MySQL versi 3. Jika perlu, ubah nama kelas instans menjadi nama yang didukung Aurora MySQL versi 3.

> **Tip**
>
> Untuk memeriksa kelas instance yang dapat Anda gunakan untuk kombinasi tertentu dari versi Aurora MySQL dan AWS Region, gunakan perintah. `describe-orderable-db-instance-options` AWS CLI

Untuk detail selengkapnya tentang kelas instans Aurora, lihat Kelas instans DB Aurora.

Perubahan parameter untuk Aurora MySQL versi 3

Aurora MySQL versi 3 mencakup parameter konfigurasi tingkat klaster dan tingkat instans baru. Aurora MySQL versi 3 juga menghapus beberapa parameter yang ada di Aurora MySQL versi 2. Beberapa nama parameter diubah sebagai hasil dari inisiatif untuk bahasa inklusif. Untuk kompatibilitas mundur, Anda masih dapat mengambil nilai parameter menggunakan nama lama atau nama baru. Namun, Anda harus menggunakan nama baru untuk menentukan nilai parameter dalam grup parameter kustom.

Di Aurora MySQL versi 3, nilai parameter `lower_case_table_names` diatur secara permanen pada saat klaster dibuat. Jika Anda menggunakan nilai nondefault untuk opsi ini, siapkan grup parameter kustom Aurora MySQL versi 3 Anda sebelum meningkatkan. Kemudian, tentukan grup parameter selama operasi pembuatan klaster atau pemulihan snapshot.

> **Note**
>
> Dengan basis data global Aurora berdasarkan Aurora MySQL, Anda tidak dapat melakukan peningkatan di tempat dari Aurora MySQL versi 2 ke versi 3 jika parameter

`lower_case_table_names` diaktifkan. Gunakan metode pemulihan snapshot sebagai gantinya.

Di Aurora MySQL versi 3, parameter `init_connect` dan `read_only` tidak berlaku untuk pengguna yang memiliki hak akses `CONNECTION_ADMIN`. Ini termasuk pengguna master Aurora. Untuk informasi selengkapnya, lihat Model hak akses berbasis peran.

Untuk daftar lengkap parameter klaster Aurora MySQL, lihat Parameter tingkat klaster. Tabel ini mencakup semua parameter dari Aurora MySQL versi 2 dan 3. Tabel ini mencakup catatan yang menunjukkan parameter mana yang baru di Aurora MySQL versi 3 atau yang telah dihapus dari Aurora MySQL versi 3.

Untuk daftar lengkap parameter instans MySQL Aurora, lihat Parameter tingkat instans. Tabel ini mencakup semua parameter dari Aurora MySQL versi 2 dan 3. Tabel ini mencakup catatan yang menunjukkan parameter mana yang baru di Aurora MySQL versi 3 dan parameter mana yang dihapus dari Aurora MySQL versi 3. Ini juga mencakup catatan yang menunjukkan parameter mana yang dapat dimodifikasi di versi sebelumnya, tetapi bukan Aurora MySQL versi 3.

Untuk informasi tentang nama parameter yang berubah, lihat Perubahan bahasa inklusif untuk Aurora MySQL versi 3.

Variabel status

Untuk informasi tentang variabel status yang tidak berlaku untuk Aurora MySQL, lihat Variabel status MySQL yang tidak berlaku untuk Aurora MySQL.

Perubahan bahasa inklusif untuk Aurora MySQL versi 3

Aurora MySQL versi 3 kompatibel dengan versi 8.0.23 MySQL edisi komunitas. Aurora MySQL versi 3 juga mencakup perubahan dari MySQL 8.0.26 terkait dengan kata kunci dan skema sistem untuk bahasa inklusif. Misalnya, perintah `SHOW REPLICA STATUS` sekarang lebih disarankan daripada `SHOW SLAVE STATUS`.

CloudWatch Metrik Amazon berikut memiliki nama baru di Aurora MySQL versi 3.

Di Aurora MySQL versi 3, hanya nama metrik baru yang tersedia. Pastikan untuk memperbarui alarm atau otomatisasi lain yang bergantung pada nama metrik saat Anda meningkatkan ke Aurora MySQL versi 3.

Nama lama	Nama baru	
ForwardingMasterDMLLatency	ForwardingWriterDMLLatency	
ForwardingMasterOpenSessions	ForwardingWriterOpenSessions	
AuroraDMLRejectedMasterFull	AuroraDMLRejectedWriterFull	
ForwardingMasterDMLThroughput	ForwardingWriterDMLThroughput	

Variabel status berikut memiliki nama baru di Aurora MySQL versi 3.

Untuk kompatibilitas, Anda dapat menggunakan salah satu nama ini dalam rilis awal Aurora MySQL versi 3. Nama variabel status lama akan dihapus dalam rilis mendatang.

Nama yang akan dihapus	Nama yang baru atau direkomendasikan	
Aurora_fwd_master_dml_stmt_duration	Aurora_fwd_writer_dml_stmt_duration	
Aurora_fwd_master_dml_stmt_count	Aurora_fwd_writer_dml_stmt_count	
Aurora_fwd_master_select_stmt_duration	Aurora_fwd_writer_select_stmt_duration	
Aurora_fwd_master_select_stmt_count	Aurora_fwd_writer_select_stmt_count	
Aurora_fwd_master_errors_session_timeout	Aurora_fwd_writer_errors_session_timeout	

Nama yang akan dihapus	Nama yang baru atau direkomendasikan	
`Aurora_fwd_master_open_sessions`	`Aurora_fwd_writer_open_sessions`	
`Aurora_fwd_master_errors_session_limit`	`Aurora_fwd_writer_errors_session_limit`	
`Aurora_fwd_master_errors_rpc_timeout`	`Aurora_fwd_writer_errors_rpc_timeout`	

Parameter konfigurasi berikut memiliki nama baru di Aurora MySQL versi 3.

Untuk kompatibilitas, Anda dapat memeriksa nilai parameter di klien `mysql` dengan menggunakan salah satu nama dalam rilis awal Aurora MySQL versi 3. Anda hanya dapat menggunakan nama baru saat memodifikasi nilai dalam grup parameter kustom. Nama parameter lama akan dihapus dalam rilis mendatang.

Nama yang akan dihapus	Nama yang baru atau direkomendasikan	
`aurora_fwd_master_idle_timeout`	`aurora_fwd_writer_idle_timeout`	
`aurora_fwd_master_max_connections_pct`	`aurora_fwd_writer_max_connections_pct`	
`master_verify_checksum`	`source_verify_checksum`	
`sync_master_info`	`sync_source_info`	
`init_slave`	`init_replica`	
`rpl_stop_slave_timeout`	`rpl_stop_replica_timeout`	

Nama yang akan dihapus	Nama yang baru atau direkomendasikan	
log_slow_slave_statements	log_slow_replica_statements	
slave_max_allowed_packet	replica_max_allowed_packet	
slave_compressed_protocol	replica_compressed_protocol	
slave_exec_mode	replica_exec_mode	
slave_type_conversions	replica_type_conversions	
slave_sql_verify_checksum	replica_sql_verify_checksum	
slave_parallel_type	replica_parallel_type	
slave_preserve_commit_order	replica_preserve_commit_order	
log_slave_updates	log_replica_updates	
slave_allow_batching	replica_allow_batching	
slave_load_tmpdir	replica_load_tmpdir	
slave_net_timeout	replica_net_timeout	
sql_slave_skip_counter	sql_replica_skip_counter	
slave_skip_errors	replica_skip_errors	

Nama yang akan dihapus	Nama yang baru atau direkomendasikan	
slave_checkpoint_period	replica_checkpoint_period	
slave_checkpoint_group	replica_checkpoint_group	
slave_transaction_retries	replica_transaction_retries	
slave_parallel_workers	replica_parallel_workers	
slave_pending_jobs_size_max	replica_pending_jobs_size_max	
pseudo_slave_mode	pseudo_replica_mode	

Prosedur tersimpan berikut memiliki nama baru di Aurora MySQL versi 3.

Untuk kompatibilitas, Anda dapat menggunakan salah satu nama ini dalam rilis awal Aurora MySQL versi 3. Nama prosedur lama akan dihapus dalam rilis mendatang.

Nama yang akan dihapus	Nama yang baru atau direkomendasikan	
mysql.rds_set_master_auto_position	mysql.rds_set_source_auto_position	
mysql.rds_set_external_master	mysql.rds_set_external_source	
mysql.rds_set_external_master_with_auto_position	mysql.rds_set_external_source_with_auto_position	

Nama yang akan dihapus	Nama yang baru atau direkomendasikan	
`mysql.rds_reset_external_master`	`mysql.rds_reset_external_source`	
`mysql.rds_next_master_log`	`mysql.rds_next_source_log`	

Nilai AUTO_INCREMENT

Di Aurora MySQL versi 3, Aurora mempertahankan nilai `AUTO_INCREMENT` untuk setiap tabel saat mengaktifkan ulang setiap instans DB. Di Aurora MySQL versi 2, nilai `AUTO_INCREMENT` tidak dipertahankan setelah pengaktifan ulang.

`AUTO_INCREMENT`Nilai tidak dipertahankan saat Anda menyiapkan klaster baru dengan memulihkan dari snapshot, melakukan point-in-time pemulihan, dan mengkloning kloning klaster. Dalam kasus ini, nilai `AUTO_INCREMENT` diinisialisasi ke nilai berdasarkan nilai kolom terbesar dalam tabel pada saat snapshot dibuat. Perilaku ini berbeda dari di RDS for MySQL 8.0, yang mempertahankan nilai selama operasi `AUTO_INCREMENT` ini.

Replikasi log biner

Dalam MySQL 8.0 edisi komunitas, replikasi log biner diaktifkan secara default. Di Aurora MySQL versi 3, replikasi log biner dinonaktifkan secara default.

> **Tip**
>
> Jika persyaratan ketersediaan tinggi Anda dipenuhi oleh fitur replikasi default Aurora, Anda dapat menonaktifkan replikasi log biner. Dengan begitu, Anda dapat menghindari overhead performa replikasi log biner. Anda juga dapat menghindari pemantauan dan pemecahan masalah terkait yang diperlukan untuk mengelola replikasi log biner.

Aurora mendukung replikasi log biner dari sumber yang kompatibel dengan MySQL 5.7 ke Aurora MySQL versi 3. Sistem sumber dapat berupa klaster DB Aurora MySQL, instans DB RDS for MySQL, atau instans MySQL on-premise.

Seperti halnya MySQL komunitas, Aurora MySQL mendukung replikasi dari sumber yang menjalankan versi tertentu ke target yang menjalankan versi mayor yang sama atau satu versi mayor yang lebih tinggi. Misalnya, replikasi dari sistem yang kompatibel dengan MySQL 5.6 ke Aurora MySQL versi 3 tidak didukung. Replikasi dari Aurora MySQL versi 3 ke sistem yang kompatibel dengan MySQL 5.7 atau MySQL 5.6 tidak didukung. Untuk detail tentang menggunakan replikasi log biner, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner).

Aurora MySQL versi 3 mencakup perbaikan replikasi log biner di MySQL 8.0, komunitas seperti replikasi yang difilter. Untuk detail tentang peningkatan MySQL 8.0 komunitas, lihat How Servers Evaluate Replication Filtering Rules dalam Panduan Referensi MySQL.

Replikasi multithread

Dengan Aurora MySQL versi 3, Aurora MySQL mendukung replikasi multithread. Untuk informasi penggunaan, lihat Replikasi log biner multithreaded (Aurora MySQL versi 3).

> **Note**
>
> Kami tetap menyarankan untuk tidak menggunakan replikasi multithread dengan Aurora MySQL versi 2.

Kompresi transaksi untuk replikasi log biner

Untuk informasi penggunaan tentang kompresi log biner, lihat Binary Log Transaction Compression dalam Panduan Referensi MySQL.

Batasan berikut berlaku untuk kompresi log biner di Aurora MySQL versi 3:

- Transaksi yang data log binernya lebih besar dari ukuran paket maksimum yang diizinkan tidak akan dikompresi. Hal ini berlaku terlepas dari apakah pengaturan kompresi log biner Aurora MySQL diaktifkan atau tidak. Transaksi semacam itu direplikasi tanpa dikompresi.
- Jika Anda menggunakan konektor untuk pengambilan data perubahan (CDC) yang belum mendukung MySQL 8.0, Anda tidak dapat menggunakan fitur ini. Kami menyarankan Anda menguji konektor pihak ketiga secara menyeluruh dengan kompresi log biner. Selain itu, kami menyarankan Anda melakukannya sebelum mengaktifkan kompresi binlog pada sistem yang menggunakan replikasi binlog untuk CDC.

Perbandingan Aurora MySQL versi 3 dan MySQL 8.0 Community Edition

Anda dapat menggunakan informasi berikut untuk mempelajari tentang perubahan yang harus diperhatikan ketika Anda mengonversi dari sistem yang kompatibel dengan MySQL 8.0 yang berbeda ke Aurora MySQL versi 3.

Secara umum, Aurora MySQL versi 3 mendukung set fitur MySQL 8.0.23 komunitas. Beberapa fitur baru dari MySQL 8.0 edisi komunitas tidak berlaku untuk Aurora MySQL. Beberapa fitur tersebut tidak kompatibel dengan beberapa aspek Aurora, seperti arsitektur penyimpanan Aurora. Fitur lain tidak diperlukan karena layanan manajemen Amazon RDS menyediakan fungsionalitas yang setara. Fitur berikut di MySQL 8.0 komunitas tidak didukung atau berfungsi secara berbeda di Aurora MySQL versi 3.

Untuk catatan rilis untuk semua rilis Aurora MySQL versi 3, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 3 dalam Catatan Rilis untuk Aurora MySQL.

Topik

- Fitur MySQL 8.0 tidak tersedia di Aurora MySQL versi 3
- Model hak akses berbasis peran
- Autentikasi

Fitur MySQL 8.0 tidak tersedia di Aurora MySQL versi 3

Fitur berikut dari MySQL 8.0 komunitas tidak tersedia atau berfungsi secara berbeda di Aurora MySQL versi 3.

- Grup sumber daya dan pernyataan SQL terkait tidak didukung di Aurora MySQL.
- Aurora MySQL tidak mendukung ruang tabel undo yang ditentukan pengguna dan pernyataan SQL terkait, seperti, dan. `CREATE UNDO TABLESPACE` `ALTER UNDO TABLESPACE ... SET INACTIVE` `DROP UNDO TABLESPACE`
- Aurora MySQL tidak mendukung pembatalan pemotongan tablespace untuk versi Aurora MySQL yang lebih rendah dari 3.06. Di Aurora MySQL versi 3.06 dan yang lebih tinggi, pemotongan tablespace undo otomatis didukung.
- Anda tidak dapat memodifikasi pengaturan plugin MySQL apa pun.
- Plugin X tidak didukung.
- Replikasi multisumber tidak didukung.

Model hak akses berbasis peran

Dengan Aurora MySQL versi 3, Anda tidak dapat memodifikasi tabel dalam basis data `mysql` secara langsung. Secara khusus, Anda tidak dapat mengatur pengguna dengan melakukan penyisipan ke dalam tabel `mysql.user`. Sebagai gantinya, Anda menggunakan pernyataan SQL untuk memberikan hak akses berbasis peran. Anda juga tidak dapat membuat jenis objek lain seperti prosedur tersimpan dalam basis data `mysql`. Anda masih dapat mengueri tabel `mysql`. Jika Anda menggunakan replikasi log biner, perubahan yang dilakukan langsung ke tabel `mysql` di klaster sumber tidak direplikasi ke klaster target.

Dalam beberapa kasus, aplikasi Anda mungkin menggunakan pintasan untuk membuat pengguna atau objek lain dengan melakukan penyisipan ke dalam tabel `mysql`. Jika demikian, ubah kode aplikasi Anda untuk menggunakan pernyataan yang sesuai seperti `CREATE USER`. Jika aplikasi Anda membuat prosedur tersimpan atau objek lain dalam basis data `mysql`, gunakan basis data yang berbeda sebagai gantinya.

Untuk mengekspor metadata bagi pengguna database selama migrasi dari database MySQL eksternal, Anda dapat menggunakan perintah MySQL Shell sebagai gantinya. `mysqldump` Untuk informasi selengkapnya, lihat Instance Dump Utility, Schema Dump Utility, dan Table Dump Utility.

Untuk menyederhanakan pengelolaan izin bagi banyak pengguna atau aplikasi, Anda dapat menggunakan pernyataan `CREATE ROLE` untuk membuat peran yang memiliki serangkaian izin. Kemudian, Anda dapat menggunakan pernyataan `GRANT` dan `SET ROLE` serta fungsi `current_role` untuk menetapkan peran ke pengguna atau aplikasi, mengganti peran saat ini, dan memeriksa peran mana yang berlaku. Untuk informasi selengkapnya tentang sistem izin berbasis peran di MySQL 8.0, lihat Using Roles dalam Panduan Referensi MySQL.

> **Important**
>
> Kami sangat menyarankan agar Anda tidak menggunakan pengguna master secara langsung di aplikasi Anda. Sebagai gantinya, ikuti praktik terbaik menggunakan pengguna basis data yang dibuat dengan hak akses minimal yang diperlukan untuk aplikasi Anda.

Aurora MySQL versi 3 mencakup peran khusus yang memiliki semua hak akses berikut. Peran ini bernama `rds_superuser_role`. Pengguna administratif primer untuk setiap klaster sudah diberi peran ini. Peran `rds_superuser_role` mencakup hak akses berikut untuk semua objek basis data:

- `ALTER`

- APPLICATION_PASSWORD_ADMIN
- ALTER ROUTINE
- CONNECTION_ADMIN
- CREATE
- CREATE ROLE
- CREATE ROUTINE
- CREATE TEMPORARY TABLES
- CREATE USER
- CREATE VIEW
- DELETE
- DROP
- DROP ROLE
- EVENT
- EXECUTE
- INDEX
- INSERT
- LOCK TABLES
- PROCESS
- REFERENCES
- RELOAD
- REPLICATION CLIENT
- REPLICATION SLAVE
- ROLE_ADMIN
- SET_USER_ID
- SELECT
- SHOW DATABASES
- SHOW_ROUTINE (Aurora MySQL versi 3.04 dan lebih tinggi)
- SHOW VIEW
- TRIGGER
- UPDATE

- `XA_RECOVER_ADMIN`

Definisi peran juga mencakup `WITH GRANT OPTION` sehingga pengguna administratif dapat memberikan peran tersebut kepada pengguna lain. Secara khusus, administrator harus memberikan hak akses yang diperlukan untuk melakukan replikasi log biner dengan klaster Aurora MySQL sebagai target.

> **Tip**
>
> Untuk melihat detail lengkap izin, masukkan pernyataan berikut.
>
> ```
> SHOW GRANTS FOR rds_superuser_role@'%';
> SHOW GRANTS FOR name_of_administrative_user_for_your_cluster@'%';
> ```

Aurora MySQL versi 3 juga mencakup peran yang dapat Anda gunakan untuk mengakses layanan lain. AWS Anda dapat mengatur peran ini sebagai alternatif untuk pernyataan `GRANT`. Misalnya, Anda menentukan `GRANT AWS_LAMBDA_ACCESS TO user`, bukan `GRANT INVOKE LAMBDA ON *.* TO user`. Untuk prosedur untuk mengakses AWS layanan lain, lihatMengintegrasikan Amazon Aurora MySQL dengan layanan AWS lainnya. Aurora MySQL versi 3 mencakup peran berikut yang terkait dengan mengakses layanan lain: AWS

- Peran `AWS_LAMBDA_ACCESS`, sebagai alternatif untuk hak akses `INVOKE LAMBDA`. Untuk informasi penggunaan, lihat Menginvokasi fungsi Lambda dari klaster DB Amazon Aurora MySQL.
- Peran `AWS_LOAD_S3_ACCESS`, sebagai alternatif untuk hak akses `LOAD FROM S3`. Untuk informasi penggunaan, lihat Memuat data ke klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3.
- Peran `AWS_SELECT_S3_ACCESS`, sebagai alternatif untuk hak akses `SELECT INTO S3`. Untuk informasi penggunaan, lihat Menyimpan data dari klaster DB Amazon Aurora MySQL ke dalam file teks di bucket Amazon S3.
- Peran `AWS_SAGEMAKER_ACCESS`, sebagai alternatif untuk hak akses `INVOKE SAGEMAKER`. Untuk informasi penggunaan, lihat Menggunakan machine learning Amazon Aurora dengan Aurora MySQL.
- Peran `AWS_COMPREHEND_ACCESS`, sebagai alternatif untuk hak akses `INVOKE COMPREHEND`. Untuk informasi penggunaan, lihat Menggunakan machine learning Amazon Aurora dengan Aurora MySQL.

Ketika Anda memberikan akses dengan menggunakan peran di Aurora MySQL versi 3, Anda juga perlu mengaktifkan peran ini dengan menggunakan pernyataan `SET ROLE role_name` atau `SET ROLE ALL`. Contoh berikut menunjukkan cara melakukannya. Ganti nama peran yang sesuai untuk `AWS_SELECT_S3_ACCESS`.

```
# Grant role to user
mysql> GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'

# Check the current roles for your user. In this case, the AWS_SELECT_S3_ACCESS role
 has not been activated.
# Only the rds_superuser_role is currently in effect.
mysql> SELECT CURRENT_ROLE();
+--------------------------+
| CURRENT_ROLE()           |
+--------------------------+
| `rds_superuser_role`@`%` |
+--------------------------+
1 row in set (0.00 sec)

# Activate all roles associated with this user using SET ROLE.
# You can activate specific roles or all roles.
# In this case, the user only has 2 roles, so we specify ALL.
mysql> SET ROLE ALL;
Query OK, 0 rows affected (0.00 sec)

# Verify role is now active
mysql> SELECT CURRENT_ROLE();
+-------------------------------------------------+
| CURRENT_ROLE()                                  |
+-------------------------------------------------+
| `AWS_LAMBDA_ACCESS`@`%`,`rds_superuser_role`@`%` |
+-------------------------------------------------+
```

Autentikasi

Di MySQL 8.0 komunitas, plugin autentikasi default adalah `caching_sha2_password`. Aurora MySQL versi 3 masih menggunakan plugin `mysql_native_password`. Anda tidak dapat mengubah pengaturan `default_authentication_plugin`.

Meningkatkan ke Aurora MySQL versi 3

Untuk jalur peningkatan khusus untuk meningkatkan basis data Anda dari Aurora MySQL versi 2 ke versi 3, Anda dapat menggunakan salah satu pendekatan berikut:

- Lakukan peningkatan di tempat. Ikuti prosedur dalam Cara melakukan upgrade di tempat.
- Buat snapshot dari klaster versi 2 dan pulihkan snapshot ini untuk membuat klaster versi 3 baru. Ikuti prosedur dalam Memulihkan dari snapshot klaster DB.

> **Tip**
>
> Dalam beberapa kasus, Anda mungkin menentukan opsi untuk mengunggah log basis data CloudWatch saat Anda meng-upgrade. Jika demikian, periksa log in CloudWatch untuk mendiagnosis masalah apa pun yang terjadi selama operasi pemutakhiran. Contoh CLI di bagian ini menunjukkan cara melakukannya dengan menggunakan opsi `--enable-cloudwatch-logs-exports`.

Topik

- Perencanaan peningkatan untuk Aurora MySQL versi 3
- Contoh peningkatan dari Aurora MySQL versi 2 ke versi 3 menggunakan metode pemulihan snapshot
- Memecahkan masalah peningkatan dengan Aurora MySQL versi 3
- Pembersihan pasca-peningkatan untuk Aurora MySQL versi 3

Perencanaan peningkatan untuk Aurora MySQL versi 3

Untuk membantu Anda memutuskan waktu dan pendekatan yang tepat untuk meningkatkan, Anda dapat mempelajari perbedaan antara Aurora MySQL versi 3 dan lingkungan Aurora dan MySQL Anda saat ini:

- Jika Anda mengonversi dari RDS for MySQL 8.0 atau MySQL 8.0 Community Edition, lihat Perbandingan Aurora MySQL versi 3 dan MySQL 8.0 Community Edition.
- Jika Anda meningkatkan dari Aurora MySQL versi 2, RDS for MySQL 5.7, atau MySQL 5.7 Community Edition, lihat Perbandingan Aurora MySQL versi 2 dan Aurora MySQL versi 3.

- Buat versi baru yang kompatibel dengan MySQL 8.0 untuk setiap grup parameter kustom. Terapkan nilai parameter kustom yang diperlukan ke grup parameter baru. Baca Perubahan parameter untuk Aurora MySQL versi 3 untuk mempelajari tentang perubahan parameter.

  > **Note**
  >
  > Untuk sebagian besar pengaturan parameter, Anda dapat memilih grup parameter kustom di dua titik. Titik ini adalah saat Anda membuat klaster atau mengaitkan grup parameter dengan klaster nanti.
  > Namun, jika Anda menggunakan pengaturan nondefault untuk parameter `lower_case_table_names`, Anda harus mengatur grup parameter kustom dengan pengaturan ini terlebih dahulu. Kemudian, tentukan grup parameter saat Anda melakukan pemulihan snapshot untuk membuat klaster. Setiap perubahan pada parameter `lower_case_table_names` tidak akan berpengaruh setelah klaster dibuat.
  > Kami menyarankan Anda menggunakan pengaturan yang sama untuk `lower_case_table_names` ketika Anda meningkatkan dari Aurora MySQL versi 2 ke versi 3.
  > Dengan basis data global Aurora berdasarkan Aurora MySQL, Anda tidak dapat melakukan peningkatan di tempat dari Aurora MySQL versi 2 ke versi 3 jika parameter `lower_case_table_names` diaktifkan. Untuk informasi selengkapnya tentang metode yang dapat Anda gunakan, lihat Peningkatan versi utama.

- Tinjau definisi skema dan objek basis data Aurora MySQL versi 2 untuk penggunaan kata kunci baru yang dicadangkan, yang diperkenalkan di MySQL 8.0 Community Edition. Lakukan hal ini sebelum Anda meningkatkan. Untuk informasi selengkapnya, lihat MySQL 8.0 New Keywords and Reserved Words dalam dokumentasi MySQL.

Anda juga dapat menemukan lebih banyak pertimbangan dan tips peningkatan khusus MySQL dalam Changes in MySQL 8.0 dalam Panduan Referensi MySQL. Misalnya, Anda dapat menggunakan perintah `mysqlcheck --check-upgrade` untuk menganalisis basis data Aurora MySQL yang ada dan mengidentifikasi potensi masalah peningkatan.

> **Note**
>
> Sebaiknya gunakan kelas instans DB yang lebih besar saat meningkatkan ke Aurora MySQL versi 3 menggunakan teknik peningkatan atau pemulihan snapshot di tempat. Contohnya adalah db.r5.24xlarge dan db.r6g.16xlarge. Hal ini membantu menyelesaikan proses

peningkatan lebih cepat dengan menggunakan sebagian besar kapasitas CPU yang tersedia pada instans DB. Anda dapat mengubah ke kelas instans DB yang Anda inginkan setelah peningkatan versi mayor selesai.

Setelah Anda menyelesaikan peningkatan itu sendiri, Anda dapat mengikuti prosedur pasca-peningkatan dalam Pembersihan pasca-peningkatan untuk Aurora MySQL versi 3. Terakhir, uji fungsionalitas dan performa aplikasi Anda.

Jika Anda mengonversi dari RDS for MySQL atau MySQL komunitas, ikuti prosedur migrasi yang dijelaskan dalam Memigrasikan data ke klaster DB Amazon Aurora MySQL. Dalam beberapa kasus, Anda mungkin menggunakan replikasi log biner untuk menyinkronkan data Anda dengan klaster Aurora MySQL versi 3 sebagai bagian dari migrasi. Jika demikian, sistem sumber harus menjalankan versi yang kompatibel dengan MySQL 5.7, atau versi yang kompatibel dengan MySQL 8.0, yaitu 8.0.23 atau lebih rendah.

Contoh peningkatan dari Aurora MySQL versi 2 ke versi 3 menggunakan metode pemulihan snapshot

AWS CLI Contoh berikut menunjukkan langkah-langkah untuk meng-upgrade klaster Aurora MySQL versi 2 ke Aurora MySQL versi 3.

Langkah pertama adalah menentukan versi klaster yang ingin Anda tingkatkan. AWS CLI Perintah berikut menunjukkan caranya. Output-nya mengonfirmasi bahwa klaster asli adalah klaster yang kompatibel dengan MySQL 5.7 yang menjalankan Aurora MySQL versi 2.09.2.

Klaster ini memiliki setidaknya satu instans DB. Agar proses peningkatan berfungsi dengan baik, klaster asli ini memerlukan instans penulis.

```
$ aws rds describe-db-clusters --db-cluster-id cluster-57-upgrade-ok \
  --query '*[].EngineVersion' --output text

5.7.mysql_aurora.2.09.2
```

Perintah berikut menunjukkan cara memeriksa jalur peningkatan mana yang tersedia dari versi tertentu. Dalam hal ini, klaster asli menjalankan versi `5.7.mysql_aurora.2.09.2`. Output-nya menunjukkan bahwa versi ini dapat ditingkatkan ke Aurora MySQL versi 3.

Jika klaster asli menggunakan nomor versi yang terlalu rendah untuk ditingkatkan ke Aurora MySQL versi 3, tingkatkan ini memerlukan langkah tambahan. Pertama, pulihkan snapshot untuk membuat

klaster baru yang dapat ditingkatkan ke Aurora MySQL versi 3. Kemudian, ambil snapshot dari klaster perantara tersebut. Terakhir, pulihkan snapshot klaster perantara tersebut untuk membuat klaster Aurora MySQL versi 3 yang baru.

```
$ aws rds describe-db-engine-versions --engine aurora-mysql \
  --engine-version 5.7.mysql_aurora.2.09.2 \
  --query 'DBEngineVersions[].ValidUpgradeTarget[].EngineVersion'
[
    "5.7.mysql_aurora.2.09.3",
    "5.7.mysql_aurora.2.10.0",
    "5.7.mysql_aurora.2.10.1",
    "5.7.mysql_aurora.2.10.2",
    "8.0.mysql_aurora.3.01.1",
    "8.0.mysql_aurora.3.02.0"
]
```

Perintah berikut membuat snapshot dari klaster untuk ditingkatkan ke Aurora MySQL versi 3. Klaster asli tetap utuh. Anda kemudian membuat klaster Aurora MySQL versi 3 yang baru dari snapshot.

```
aws rds create-db-cluster-snapshot --db-cluster-id cluster-57-upgrade-ok \
  --db-cluster-snapshot-id cluster-57-upgrade-ok-snapshot
{
  "DBClusterSnapshotIdentifier": "cluster-57-upgrade-ok-snapshot",
  "DBClusterIdentifier": "cluster-57-upgrade-ok",
  "SnapshotCreateTime": "2021-10-06T23:20:18.087000+00:00"
}
```

Perintah berikut memulihkan snapshot ke klaster baru yang menjalankan Aurora MySQL versi 3.

```
$ aws rds restore-db-cluster-from-snapshot \
  --snapshot-id cluster-57-upgrade-ok-snapshot \
  --db-cluster-id cluster-80-restored --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.02.0 \
  --enable-cloudwatch-logs-exports '["error","general","slowquery","audit"]'
{
  "DBClusterIdentifier": "cluster-80-restored",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "Status": "creating"
}
```

Pemulihan snapshot akan menyiapkan penyimpanan untuk klaster dan menetapkan versi basis data yang dapat digunakan klaster. Kapasitas komputasi klaster terpisah dari penyimpanan. Dengan demikian, Anda mengatur instans DB apa pun untuk klaster setelah klaster itu sendiri dibuat. Contoh berikut membuat instans DB penulis menggunakan salah satu kelas instans db.r5.

> **Tip**
>
> Anda mungkin memiliki skrip administrasi yang membuat instans DB menggunakan kelas instans lama seperti db.r3, db.r4, db.t2, atau db.t3. Jika demikian, ubah skrip Anda untuk menggunakan salah satu kelas instans yang didukung dengan Aurora MySQL versi 3. Untuk informasi tentang kelas instans yang dapat Anda gunakan dengan Aurora MySQL versi 3, lihat Dukungan kelas instans.

```
$ aws rds create-db-instance --db-instance-identifier instance-running-version-3 \
  --db-cluster-identifier cluster-80-restored \
  --db-instance-class db.r5.xlarge --engine aurora-mysql
{
  "DBInstanceIdentifier": "instance-running-version-3",
  "DBClusterIdentifier": "cluster-80-restored",
  "DBInstanceClass": "db.r5.xlarge",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceStatus": "creating"
}
```

Setelah peningkatan selesai, Anda dapat memverifikasi nomor versi khusus Aurora milik klaster dengan menggunakan AWS CLI.

```
$ aws rds describe-db-clusters --db-cluster-id cluster-80-restored \
  --query '*[].EngineVersion' --output text
8.0.mysql_aurora.3.02.0
```

Anda juga dapat memverifikasi versi khusus MySQL milik mesin basis data dengan memanggil fungsi `version`.

```
mysql> select version();
+-----------+
| version() |
+-----------+
```

```
| 8.0.23    |
+-----------+
```

Memecahkan masalah peningkatan dengan Aurora MySQL versi 3

Jika peningkatan Anda ke Aurora MySQL versi 3 tidak berhasil diselesaikan, Anda dapat mendiagnosis penyebab masalahnya. Kemudian, Anda dapat membuat perubahan yang diperlukan pada skema basis data atau data tabel asli dan menjalankan proses peningkatan lagi.

Jika proses peningkatan ke Aurora MySQL versi 3 gagal, masalahnya akan terdeteksi saat membuat lalu meningkatkan instans penulis untuk snapshot yang dipulihkan. Aurora meninggalkan instans penulis asli yang kompatibel dengan versi 5.7. Dengan begitu, Anda dapat memeriksa log dari pemeriksaan awal yang dijalankan Aurora sebelum melakukan peningkatan. Contoh berikut dimulai dengan basis data yang kompatibel dengan 5.7 yang memerlukan beberapa perubahan sebelum dapat ditingkatkan ke Aurora MySQL versi 3. Contoh ini menunjukkan bagaimana percobaan peningkatan pertama tidak berhasil dan bagaimana cara memeriksa file log-nya. Contoh ini juga menunjukkan cara memperbaiki masalah dan menjalankan peningkatan yang berhasil.

Pertama, kita membuat klaster yang kompatibel dengan MySQL 5.7 bernama `problematic-57-80-upgrade`. Seperti namanya, klaster ini berisi setidaknya satu objek skema yang menyebabkan masalah selama peningkatan ke versi yang kompatibel dengan MySQL 8.0.

```
$ aws rds create-db-cluster --engine aurora-mysql \
  --engine-version 5.7.mysql_aurora.2.10.0 \
  --db-cluster-identifier problematic-57-80-upgrade \
  --master-username my_username \
  --manage-master-user-password
{
  "DBClusterIdentifier": "problematic-57-80-upgrade",
  "Status": "creating"
}

$ aws rds create-db-instance \
  --db-instance-identifier instance-preupgrade \
  --db-cluster-identifier problematic-57-80-upgrade \
  --db-instance-class db.t2.small --engine aurora-mysql
{
  "DBInstanceIdentifier": "instance-preupgrade",
  "DBClusterIdentifier": "problematic-57-80-upgrade",
  "DBInstanceClass": "db.t2.small",
  "DBInstanceStatus": "creating"
```

```
}

$ aws rds wait db-instance-available \
  --db-instance-identifier instance-preupgrade
```

Di klaster yang ingin kita peningkatan, kita memasukkan tabel yang bermasalah. Saat indeks FULLTEXT dibuat lalu dihapus, ada beberapa metadata yang tertinggal yang menyebabkan masalah selama peningkatan. Contoh ini menentukan opsi `--manage-master-user-password` untuk menghasilkan kata sandi pengguna master dan mengelolanya di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. Alternatifnya, Anda dapat menggunakan opsi `--master-password` untuk menentukan dan mengelola kata sandi sendiri.

```
$ mysql -u my_username -p \
  -h problematic-57-80-upgrade.cluster-example123.us-east-1.rds.amazonaws.com

mysql> create database problematic_upgrade;
Query OK, 1 row affected (0.02 sec)

mysql> use problematic_upgrade;
Database changed
mysql> CREATE TABLE dangling_fulltext_index
    ->   (id INT AUTO_INCREMENT PRIMARY KEY, txtcol TEXT NOT NULL)
    ->   ENGINE=InnoDB;
Query OK, 0 rows affected (0.05 sec)

mysql> ALTER TABLE dangling_fulltext_index ADD FULLTEXT(txtcol);
Query OK, 0 rows affected, 1 warning (0.14 sec)

mysql> ALTER TABLE dangling_fulltext_index DROP INDEX txtcol;
Query OK, 0 rows affected (0.06 sec)
```

Contoh ini mencoba melakukan prosedur peningkatan. Kita mengambil snapshot dari klaster asli dan menunggu pembuatan snapshot selesai. Kemudian, kita memulihkan snapshot, dengan menentukan nomor versi yang kompatibel dengan MySQL 8.0. Kita juga membuat instans penulis untuk klaster. Itulah titik saat proses peningkatan benar-benar terjadi. Kemudian, kita tunggu sampai instans menjadi tersedia. Itulah titik saat proses peningkatan selesai, baik berhasil maupun gagal.

> **Tip**
>
> Jika Anda mengembalikan snapshot menggunakan AWS Management Console, Aurora membuat instance penulis secara otomatis dan memutakhirkannya ke versi mesin yang diminta.

```
$ aws rds create-db-cluster-snapshot --db-cluster-id problematic-57-80-upgrade \
  --db-cluster-snapshot-id problematic-57-80-upgrade-snapshot
{
  "DBClusterSnapshotIdentifier": "problematic-57-80-upgrade-snapshot",
  "DBClusterIdentifier": "problematic-57-80-upgrade",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.0"
}

$ aws rds wait db-cluster-snapshot-available \
  --db-cluster-snapshot-id problematic-57-80-upgrade-snapshot

$ aws rds restore-db-cluster-from-snapshot \
  --snapshot-id problematic-57-80-upgrade-snapshot \
  --db-cluster-id cluster-80-attempt-1 --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.02.0 \
  --enable-cloudwatch-logs-exports '["error","general","slowquery","audit"]'
{
  "DBClusterIdentifier": "cluster-80-attempt-1",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "Status": "creating"
}

$ aws rds create-db-instance --db-instance-identifier instance-attempt-1 \
  --db-cluster-identifier cluster-80-attempt-1 \
  --db-instance-class db.r5.xlarge --engine aurora-mysql
{
  "DBInstanceIdentifier": "instance-attempt-1",
  "DBClusterIdentifier": "cluster-80-attempt-1",
  "DBInstanceClass": "db.r5.xlarge",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceStatus": "creating"
}
```

```
$ aws rds wait db-instance-available \
  --db-instance-identifier instance-attempt-1
```

Sekarang kita memeriksa klaster yang baru dibuat dan instans terkait untuk memverifikasi apakah peningkatan-nya berhasil. Klaster dan instans masih menjalankan versi MySQL yang kompatibel dengan versi 5.7. Itu berarti peningkatan-nya gagal. Ketika peningkatan gagal, Aurora hanya meninggalkan instans penulis sehingga Anda dapat memeriksa file log apa pun. Anda tidak dapat memulai ulang proses peningkatan dengan klaster yang baru dibuat tersebut. Setelah Anda memperbaiki masalah dengan membuat perubahan di klaster asli Anda, Anda harus menjalankan langkah-langkah peningkatan lagi: membuat snapshot baru dari klaster asli dan memulihkannya ke klaster MySQL 8.0 lain yang kompatibel.

```
$ aws rds describe-db-clusters \
  --db-cluster-identifier cluster-80-attempt-1 \
  --query '*[].[Status]' --output text
available
$ aws rds describe-db-clusters \
  --db-cluster-identifier cluster-80-attempt-1 \
  --query '*[].[EngineVersion]' --output text
5.7.mysql_aurora.2.10.0

$ aws rds describe-db-instances \
  --db-instance-identifier instance-attempt-1 \
  --query '*[].{DBInstanceStatus:DBInstanceStatus}' --output text
available
$ aws rds describe-db-instances \
  --db-instance-identifier instance-attempt-1 \
  --query '*[].[EngineVersion]' --output text
5.7.mysql_aurora.2.10.0
```

Untuk mendapatkan ringkasan tentang apa yang terjadi selama proses peningkatan, kita mendapatkan daftar peristiwa untuk instans penulis yang baru dibuat. Dalam contoh ini, kita menampilkan daftar peristiwa selama 600 menit terakhir untuk mencakup seluruh interval waktu proses peningkatan. Peristiwa dalam daftar tidak selalu kronologis. Peristiwa yang disorot menunjukkan masalah yang mengonfirmasi bahwa klaster tidak dapat ditingkatkan.

```
$ aws rds describe-events \
  --source-identifier instance-attempt-1 --source-type db-instance \
  --duration 600
{
    "Events": [
```

```
        {
            "SourceIdentifier": "instance-attempt-1",
            "SourceType": "db-instance",
            "Message": "Binlog position from crash recovery is mysql-bin-
changelog.000001 154",
            "EventCategories": [],
            "Date": "2021-12-03T20:26:17.862000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:instance-attempt-1"
        },
        {
            "SourceIdentifier": "instance-attempt-1",
            "SourceType": "db-instance",
            "Message": "Database cluster is in a state that cannot be upgraded:
 PreUpgrade checks failed. More details can
             be found in the upgrade-prechecks.log file. Please refer to
             https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
AuroraMySQL.MySQL80.html#AuroraMySQL.mysql80-upgrade-troubleshooting
             for more details on troubleshooting the cause of the upgrade failure",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2021-12-03T20:26:50.436000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:instance-attempt-1"
        },
        {
            "SourceIdentifier": "instance-attempt-1",
            "SourceType": "db-instance",
            "Message": "DB instance created",
            "EventCategories": [
                "creation"
            ],
            "Date": "2021-12-03T20:26:58.830000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:instance-attempt-1"
        },
...
```

Untuk mendiagnosis penyebab persis masalahnya, periksa log basis data untuk instans penulis yang baru dibuat. Ketika peningkatan ke versi yang kompatibel dengan 8.0 gagal, instans ini akan berisi file log dengan nama file `upgrade-prechecks.log`. Contoh ini menunjukkan cara mendeteksi keberadaan log tersebut lalu mengunduhnya ke file lokal untuk diperiksa.

```
$ aws rds describe-db-log-files --db-instance-identifier instance-attempt-1 \
  --query '*[].[LogFileName]' --output text
```

```
error/mysql-error-running.log
error/mysql-error-running.log.2021-12-03.20
error/mysql-error-running.log.2021-12-03.21
error/mysql-error.log
external/mysql-external.log
upgrade-prechecks.log

$ aws rds download-db-log-file-portion --db-instance-identifier instance-attempt-1 \
  --log-file-name upgrade-prechecks.log --starting-token 0 \
  --output text >upgrade_prechecks.log
```

File `upgrade-prechecks.log` memiliki format JSON. Kita mengunduhnya menggunakan opsi `--output text` untuk menghindari pengenkodean output JSON dalam wrapper JSON lain. Untuk peningkatan Aurora MySQL versi 3, log ini selalu menyertakan pesan informasi dan peringatan tertentu. Log ini hanya berisi pesan kesalahan jika peningkatan gagal. Jika peningkatan berhasil, file log tidak dibuat sama sekali. Kutipan berikut menunjukkan jenis entri yang dapat Anda temukan.

```
$ cat upgrade-prechecks.log
{
    "serverAddress": "/tmp%2Fmysql.sock",
    "serverVersion": "5.7.12",
    "targetVersion": "8.0.23",
    "auroraServerVersion": "2.10.0",
    "auroraTargetVersion": "3.02.0",
    "outfilePath": "/rdsdbdata/tmp/PreChecker.log",
    "checksPerformed": [
```

Jika `"detectedProblems"` kosong, tingkatkan tidak mengalami kemunculan jenis masalah tersebut. Anda dapat mengabaikan entri tersebut.

```
{
    "id": "oldTemporalCheck",
    "title": "Usage of old temporal type",
    "status": "OK",
   "detectedProblems": []
},
```

Pemeriksaan untuk variabel yang dihapus atau nilai default yang diubah tidak dilakukan secara otomatis. Aurora menggunakan mekanisme grup parameter alih-alih file konfigurasi fisik. Anda selalu menerima beberapa pesan dengan status `CONFIGURATION_ERROR` ini, baik perubahan variabel

berpengaruh pada basis data Anda maupun tidak. Anda dapat membaca dokumentasi MySQL untuk detail tentang perubahan ini.

```
{
    "id": "removedSysLogVars",
    "title": "Removed system variables for error logging to the system log
 configuration",
    "status": "CONFIGURATION_ERROR",
    "description": "To run this check requires full path to MySQL server
configuration file to be specified at 'configPath' key of options dictionary",
    "documentationLink": "https://dev.mysql.com/doc/relnotes/mysql/8.0/en/
news-8-0-13.html#mysqld-8-0-13-logging"
},
```

Peringatan tentang jenis data tanggal dan waktu yang sudah tidak berlaku ini terjadi jika pengaturan `SQL_MODE` dalam grup parameter Anda dibiarkan pada nilai default. Basis data Anda mungkin atau mungkin tidak berisi kolom dengan jenis yang terpengaruh.

```
{
    "id": "zeroDatesCheck",
    "title": "Zero Date, Datetime, and Timestamp values",
    "status": "OK",
    "description": "Warning: By default zero date/datetime/timestamp
values are no longer allowed in MySQL, as of 5.7.8 NO_ZERO_IN_DATE and
NO_ZERO_DATE are included in SQL_MODE by default. These modes should be used
with strict mode as they will be merged with strict mode in a future release.
If you do not include these modes in your SQL_MODE setting, you are able to
insert date/datetime/timestamp values that contain zeros. It is strongly
advised to replace zero values with valid ones, as they may not work
correctly in the future.",
    "documentationLink": "https://lefred.be/content/mysql-8-0-and-wrong-dates/",
    "detectedProblems": [
        {
            "level": "Warning",
            "dbObject": "global.sql_mode",
            "description": "does not contain either NO_ZERO_DATE or
NO_ZERO_IN_DATE which allows insertion of zero dates"
        }
    ]
},
```

Jika bidang `detectedProblems` berisi entri dengan nilai `level` yang menampilkan `Error`, berarti peningkatan tidak dapat berhasil sampai Anda memperbaiki masalah tersebut.

```
{
    "id": "getDanglingFulltextIndex",
    "title": "Tables with dangling FULLTEXT index reference",
    "status": "OK",
    "description": "Error: The following tables contain dangling
FULLTEXT index which is not supported. It is recommended to rebuild the
table before upgrade.",
    "detectedProblems": [
        {
            "level": "Error",
            "dbObject": "problematic_upgrade.dangling_fulltext_index",
            "description": "Table `problematic_upgrade.dangling_fulltext_index` contains
 dangling FULLTEXT index. Kindly recreate the table before upgrade."
        }
    ]
},
```

Tip

Untuk meringkas semua kesalahan tersebut serta menampilkan bidang objek dan deskripsi terkait, Anda dapat menjalankan perintah `grep -A 2 '"level": "Error"'` pada konten file `upgrade-prechecks.log`. Tindakan ini akan menampilkan setiap baris kesalahan dan dua baris setelahnya. Baris ini berisi nama objek basis data yang sesuai dan panduan tentang cara memperbaiki masalahnya.

```
$ cat upgrade-prechecks.log | grep -A 2 '"level": "Error"'
"level": "Error",
"dbObject": "problematic_upgrade.dangling_fulltext_index",
"description": "Table `problematic_upgrade.dangling_fulltext_index` contains
 dangling FULLTEXT index. Kindly recreate the table before upgrade."
```

Pemeriksaan `defaultAuthenticationPlugin` ini selalu menampilkan pesan peringatan ini untuk peningkatan Aurora MySQL versi 3. Itu karena Aurora MySQL versi 3 menggunakan plugin `mysql_native_password` sebagai ganti `caching_sha2_password`. Anda tidak perlu mengambil tindakan apa pun untuk peringatan ini.

```
{
    "id": "defaultAuthenticationPlugin",
    "title": "New default authentication plugin considerations",
    "description": "Warning: The new default authentication plugin
'caching_sha2_password' offers more secure password hashing than previously
used 'mysql_native_password' (and consequent improved client connection
...
    "documentationLink": "https://dev.mysql.com/doc/refman/8.0/en/upgrading-from-
previous-series.html#upgrade-caching-sha2-password-compatibility-issues\nhttps://
dev.mysql.com/doc/refman/8.0/en/upgrading-from-previous-series.html#upgrade-caching-
sha2-password-replication"
}
```

Bagian akhir file `upgrade-prechecks.log` akan merangkum berapa banyak pemeriksaan yang menemukan setiap jenis masalah ringan atau berat. `errorCount` bukan nol menunjukkan bahwa peningkatan gagal.

```
    ],
    "errorCount": 1,
    "warningCount": 2,
    "noticeCount": 0,
    "Summary": "1 errors were found. Please correct these issues before upgrading to
 avoid compatibility issues."
}
```

Urutan contoh berikutnya menunjukkan cara memperbaiki masalah khusus ini dan menjalankan proses peningkatan lagi. Kali ini, tingkatkan berhasil.

Pertama, kita kembali ke klaster asli. Kemudian, kita menjalankan `OPTIMIZE TABLE tbl_name [, tbl_name] ...` pada tabel yang menyebabkan kesalahan berikut:

Tabel `tbl_name` berisi indeks FULLTEXT yang menggantung. Buat ulang tabel tersebut sebelum peningkatan.

```
$ mysql -u my_username -p \
  -h problematic-57-80-upgrade.cluster-example123.us-east-1.rds.amazonaws.com
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema  |
```

```
| mysql              |
| performance_schema |
| problematic_upgrade |
| sys                |
+--------------------+
5 rows in set (0.00 sec)
mysql> use problematic_upgrade;
mysql> show tables;
+------------------------------+
| Tables_in_problematic_upgrade |
+------------------------------+
| dangling_fulltext_index      |
+------------------------------+
1 row in set (0.00 sec)
mysql> OPTIMIZE TABLE dangling_fulltext_index;
Query OK, 0 rows affected (0.05 sec)
```

Untuk informasi selengkapnya, lihat Optimizing InnoDB Full-Text Indexes dan OPTIMIZE TABLE Statement dalam dokumentasi MySQL.

Sebagai solusi alternatif, Anda dapat membuat tabel baru dengan struktur dan konten yang sama dengan tabel yang memiliki metadata yang salah. Dalam praktiknya, Anda mungkin akan mengganti nama tabel ini kembali ke nama tabel asli setelah peningkatan.

```
$ mysql -u my_username -p \
  -h problematic-57-80-upgrade.cluster-example123.us-east-1.rds.amazonaws.com

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| problematic_upgrade |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> use problematic_upgrade;
mysql> show tables;
+------------------------------+
| Tables_in_problematic_upgrade |
```

```
+------------------------------+
| dangling_fulltext_index      |
+------------------------------+
1 row in set (0.00 sec)

mysql> desc dangling_fulltext_index;
+--------+---------+------+-----+---------+----------------+
| Field  | Type    | Null | Key | Default | Extra          |
+--------+---------+------+-----+---------+----------------+
| id     | int(11) | NO   | PRI | NULL    | auto_increment |
| txtcol | text    | NO   |     | NULL    |                |
+--------+---------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE recreated_table LIKE dangling_fulltext_index;
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO recreated_table SELECT * FROM dangling_fulltext_index;
Query OK, 0 rows affected (0.00 sec)

mysql> drop table dangling_fulltext_index;
Query OK, 0 rows affected (0.05 sec)
```

Sekarang kita menjalankan proses yang sama seperti sebelumnya. Kita membuat snapshot dari klaster asli dan memulihkan snapshot ini ke klaster baru yang kompatibel dengan MySQL 8.0. Kemudian, kita membuat instans penulis untuk menyelesaikan proses peningkatan.

```
$ aws rds create-db-cluster-snapshot --db-cluster-id problematic-57-80-upgrade \
  --db-cluster-snapshot-id problematic-57-80-upgrade-snapshot-2
{
  "DBClusterSnapshotIdentifier": "problematic-57-80-upgrade-snapshot-2",
  "DBClusterIdentifier": "problematic-57-80-upgrade",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.0"
}

$ aws rds wait db-cluster-snapshot-available \
  --db-cluster-snapshot-id problematic-57-80-upgrade-snapshot-2

$ aws rds restore-db-cluster-from-snapshot \
  --snapshot-id problematic-57-80-upgrade-snapshot-2 \
  --db-cluster-id cluster-80-attempt-2 --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.02.0 \
```

```
  --enable-cloudwatch-logs-exports '["error","general","slowquery","audit"]'
{
  "DBClusterIdentifier": "cluster-80-attempt-2",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "Status": "creating"
}

$ aws rds create-db-instance --db-instance-identifier instance-attempt-2 \
  --db-cluster-identifier cluster-80-attempt-2 \
  --db-instance-class db.r5.xlarge --engine aurora-mysql
{
  "DBInstanceIdentifier": "instance-attempt-2",
  "DBClusterIdentifier": "cluster-80-attempt-2",
  "DBInstanceClass": "db.r5.xlarge",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceStatus": "creating"
}

$ aws rds wait db-instance-available \
  --db-instance-identifier instance-attempt-2
```

Kali ini, dengan memeriksa versi setelah peningkatan selesai, akan ditemukan bahwa nomor versi berubah untuk mencerminkan Aurora MySQL versi 3. Kita dapat terhubung ke basis data dan mengonfirmasi bahwa versi mesin MySQL adalah versi yang kompatibel dengan 8.0. Kita mengonfirmasi bahwa klaster yang ditingkatkan berisi versi tetap dari tabel yang menyebabkan masalah peningkatan sebelumnya.

```
$ aws rds describe-db-clusters \
  --db-cluster-identifier cluster-80-attempt-2 \
  --query '*[].[EngineVersion]' --output text
8.0.mysql_aurora.3.02.0
$ aws rds describe-db-instances \
  --db-instance-identifier instance-attempt-2 \
  --query '*[].[EngineVersion]' --output text
8.0.mysql_aurora.3.02.0

$ mysql -h cluster-80-attempt-2.cluster-example123.us-east-1.rds.amazonaws.com \
  -u my_username -p

mysql> select version();
+-----------+
| version() |
```

```
+-----------+
| 8.0.23    |
+-----------+
1 row in set (0.00 sec)

mysql> show databases;
+---------------------+
| Database            |
+---------------------+
| information_schema  |
| mysql               |
| performance_schema  |
| problematic_upgrade |
| sys                 |
+---------------------+
5 rows in set (0.00 sec)

mysql> use problematic_upgrade;
Database changed
mysql> show tables;
+------------------------------+
| Tables_in_problematic_upgrade |
+------------------------------+
| recreated_table               |
+------------------------------+
1 row in set (0.00 sec)
```

Pembersihan pasca-peningkatan untuk Aurora MySQL versi 3

Setelah Anda selesai meningkatkan klaster Aurora MySQL versi 2 ke Aurora MySQL versi 3, Anda dapat melakukan tindakan pembersihan lainnya sebagai berikut:

- Buat versi baru yang kompatibel dengan MySQL 8.0 untuk setiap grup parameter kustom. Terapkan nilai parameter kustom yang diperlukan ke grup parameter baru.
- Perbarui CloudWatch alarm, skrip penyiapan, dan sebagainya untuk menggunakan nama baru untuk metrik apa pun yang namanya dipengaruhi oleh perubahan bahasa inklusif. Untuk daftar metrik tersebut, lihat Perubahan bahasa inklusif untuk Aurora MySQL versi 3.
- Perbarui AWS CloudFormation templat apa pun untuk menggunakan nama baru untuk parameter konfigurasi apa pun yang namanya dipengaruhi oleh perubahan bahasa inklusif. Untuk daftar parameter tersebut, lihat Perubahan bahasa inklusif untuk Aurora MySQL versi 3.

Indeks spasial

Setelah meningkatkan ke Aurora MySQL versi 3, periksa apakah Anda perlu menghapus atau membuat ulang objek dan indeks yang terkait dengan indeks spasial. Sebelum MySQL 8.0, Aurora dapat mengoptimalkan kueri spasial menggunakan indeks yang tidak berisi pengidentifikasi sumber daya spasial (SRID). Aurora MySQL versi 3 hanya menggunakan indeks spasial yang berisi SRID. Selama peningkatan, Aurora secara otomatis menghapus indeks spasial apa pun yang tidak memiliki SRID dan mencetak pesan peringatan di log basis data. Jika Anda mengamati pesan peringatan tersebut, buat indeks spasial baru dengan SRID setelah peningkatan. Untuk informasi selengkapnya tentang perubahan fungsi spasial dan tipe data di MySQL 8.0, lihat Changes in MySQL 8.0 dalam Panduan Referensi MySQL.

Aurora MySQL versi 2 yang kompatibel dengan MySQL 5.7

Topik ini menjelaskan perbedaan antara Aurora MySQL versi 2 dan MySQL 5.7 Community Edition.

Fitur yang tidak didukung di Aurora MySQL versi 2

Fitur berikut didukung di MySQL 5.7, tetapi saat ini tidak didukung di Aurora MySQL versi 2:

- Pernyataan SQL `CREATE TABLESPACE`
- Plugin replikasi grup
- Peningkatan ukuran halaman
- Pemuatan pool buffer InnoDB saat pengaktifan
- Plugin pengurai teks lengkap InnoDB
- Replikasi multisumber
- Perubahan ukuran pool buffer online
- Plugin validasi kata sandi – Anda dapat menginstal plugin ini, tetapi tidak didukung. Anda tidak dapat menyesuaikan plugin.
- Plugin tulis ulang kueri
- Pemfilteran replikasi
- Protokol X

Untuk informasi selengkapnya tentang fitur ini, lihat Dokumentasi MySQL 5.7.

Perilaku ruang tabel sementara di Aurora MySQL versi 2

Di MySQL 5.7, ruang tabel sementara dapat diperluas otomatis dan meningkatkan ukuran yang diperlukan untuk mengakomodasi tabel sementara di disk. Ketika tabel sementara dihapus, ruang kosong dapat digunakan kembali untuk tabel sementara baru, tetapi ruang tabel sementara tetap pada ukuran yang diperluas dan tidak menyusut. Ruang tabel sementara dihapus dan dibuat ulang saat mesin diaktifkan ulang.

Di Aurora MySQL versi 2, perilaku berikut berlaku:

- Untuk klaster DB Aurora MySQL baru yang dibuat dengan versi 2.10 dan lebih tinggi, ruang tabel sementara dihapus dan dibuat ulang saat Anda mengaktifkan ulang basis data. Hal ini memungkinkan fitur pengubahan ukuran dinamis mengklaim kembali ruang penyimpanan.
- Untuk klaster DB Aurora MySQL yang ada yang di-upgrade ke:
  - Versi 2.10 atau lebih tinggi – Ruang tabel sementara dihapus dan dibuat ulang saat Anda mengaktifkan ulang basis data. Hal ini memungkinkan fitur pengubahan ukuran dinamis mengklaim kembali ruang penyimpanan.
  - Versi 2.09 – Ruang tabel sementara tidak dihapus saat Anda mengaktifkan ulang basis data.

Anda dapat memeriksa ukuran ruang tabel sementara pada klaster DB Aurora MySQL versi 2 Anda dengan menggunakan kueri berikut:

```
SELECT
    FILE_NAME,
    TABLESPACE_NAME,
    ROUND((TOTAL_EXTENTS * EXTENT_SIZE) / 1024 / 1024 / 1024, 4) AS SIZE
FROM
    INFORMATION_SCHEMA.FILES
WHERE
    TABLESPACE_NAME = 'innodb_temporary';
```

Untuk informasi selengkapnya, lihat The Temporary Tablespace dalam dokumentasi MySQL.

Mesin penyimpanan untuk tabel sementara di disk

Aurora MySQL versi 2 menggunakan mesin penyimpanan yang berbeda untuk tabel sementara internal di disk bergantung pada peran instans.

- Pada instans penulis, tabel sementara di disk menggunakan mesin penyimpanan InnoDB secara default. Tabel tersebut disimpan di ruang tabel sementara di volume klaster Aurora.

  Anda dapat mengubah perilaku ini pada instans penulis dengan memodifikasi nilai untuk parameter DB `internal_tmp_disk_storage_engine`. Untuk informasi selengkapnya, lihat Parameter tingkat instans.
- Pada instans pembaca, tabel sementara di disk menggunakan mesin penyimpanan MyISAM, yang menggunakan penyimpanan lokal. Itu karena instans hanya baca tidak dapat menyimpan data apa pun pada volume klaster Aurora.

Keamanan dengan Amazon Aurora MySQL

Keamanan untuk Amazon Aurora MySQL dikelola di tiga tingkat:

- Untuk mengontrol siapa yang dapat melakukan tindakan pengelolaan Amazon RDS pada cluster DB MySQL Aurora dan instans DB, Anda menggunakan (IAM). AWS Identity and Access Management Saat Anda terhubung AWS menggunakan kredensyal IAM, AWS akun Anda harus memiliki kebijakan IAM yang memberikan izin yang diperlukan untuk melakukan operasi pengelolaan Amazon RDS. Lihat informasi yang lebih lengkap di Manajemen identitas dan akses untuk Amazon Aurora

  Jika Anda menggunakan IAM untuk mengakses konsol Amazon RDS, pastikan untuk masuk terlebih dahulu AWS Management Console dengan kredensyal pengguna IAM Anda. Kemudian, buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

- Pastikan untuk membuat klaster DB Aurora MySQL di cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. Untuk mengontrol perangkat dan instans Amazon EC2 mana yang dapat membuka koneksi ke titik akhir dan port instans DB untuk klaster DB Aurora MySQL di VPC, gunakan grup keamanan VPC. Anda dapat membuat titik akhir dan koneksi port ini menggunakan Keamanan Lapisan Pengangkutan (TLS). Selain itu, aturan firewall di perusahaan Anda dapat mengontrol apakah perangkat yang berjalan di perusahaan Anda dapat membuka koneksi ke instans DB. Untuk informasi selengkapnya tentang VPC, lihat Amazon VPC dan Amazon Aurora.

  Penghunian VPC yang didukung tergantung pada kelas instans yang digunakan oleh klaster DB Aurora MySQL Anda. Dengan penghunian VPC `default`, VPC berjalan di perangkat keras bersama. Dengan penghunian VPC `dedicated`, VPC berjalan pada instans perangkat keras khusus. Kelas instans DB performa yang dapat melonjak hanya mendukung penghunian VPC default. Kelas instans DB performa yang dapat melonjak mencakup kelas instans db.t2, db.t3, dan db.t4g DB. Semua kelas instans Aurora MySQL DB lainnya mendukung penghunian VPC default dan khusus.

> **Note**
>
> Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Menggunakan kelas instans T untuk pengembangan dan pengujian.

Untuk informasi selengkapnya tentang kelas instans, lihat Kelas instans DB Aurora. Untuk informasi selengkapnya tentang penghunian VPC `default` dan `dedicated`, lihat Instans khusus dalam Panduan Pengguna Amazon Elastic Compute Cloud.

- Untuk mengautentikasi login dan izin untuk klaster DB Amazon Aurora MySQL, Anda dapat mengambil salah satu dari pendekatan berikut, atau kombinasi dari pendekatan tersebut:
  - Anda dapat menggunakan pendekatan yang sama dengan instans MySQL mandiri.

    Perintah seperti `CREATE USER`, `RENAME USER`, `GRANT`, `REVOKE`, dan `SET PASSWORD` beroperasi dalam basis data on-premise seperti halnya memodifikasi langsung tabel skema basis data. Untuk informasi selengkapnya, lihat Access control and account management dalam dokumentasi MySQL.
  - Anda juga dapat menggunakan autentikasi basis data IAM.

    Dengan autentikasi basis data IAM, Anda mengautentikasi klaster DB Anda dengan menggunakan pengguna IAM atau peran IAM dan token autentikasi. Token autentikasi adalah nilai unik yang dihasilkan dengan menggunakan proses penandatanganan Signature Versi 4. Dengan menggunakan autentikasi basis data IAM, Anda dapat menggunakan kredensyal yang sama untuk mengontrol akses ke AWS sumber daya dan basis data Anda. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.

> **Note**
>
> Untuk informasi selengkapnya, lihat Keamanan dalam Amazon Aurora.

Hak akses pengguna master Amazon Aurora MySQL

Saat Anda membuat instans DB Amazon Aurora MySQL, pengguna master memiliki hak default yang tercantum di Hak akses akun pengguna master.

Untuk menyediakan layanan manajemen untuk setiap klaster DB, pengguna `admin` dan `rdsadmin` dibuat saat klaster DB dibuat. Mencoba menghapus, mengubah nama, mengubah kata sandi, atau mengubah hak akses untuk akun `rdsadmin` akan mengakibatkan kesalahan.

Dalam klaster DB Aurora MySQL versi 2, pengguna `admin` dan `rdsadmin` dibuat ketika klaster DB dibuat. Dalam klaster DB Aurora MySQL versi 3, pengguna `admin`, `rdsadmin`, dan `rds_superuser_role` dibuat.

> **Important**
>
> Kami sangat menyarankan agar Anda tidak menggunakan pengguna master secara langsung di aplikasi Anda. Sebagai gantinya, ikuti praktik terbaik menggunakan pengguna basis data yang dibuat dengan hak akses minimal yang diperlukan untuk aplikasi Anda.

Untuk manajemen klaster DB Aurora MySQL, perintah `kill` dan `kill_query` standar telah dibatasi. Sebagai gantinya, gunakan perintah Amazon RDS `rds_kill` dan `rds_kill_query` untuk menghentikan sesi pengguna atau kueri di instans DB Aurora MySQL.

> **Note**
>
> Enkripsi instans basis data dan snapshot tidak didukung untuk wilayah Tiongkok (Ningxia).

Menggunakan TLS dengan klaster DB Aurora MySQL

Klaster DB Amazon Aurora MySQL mendukung koneksi Keamanan Lapisan Pengangkutan (TLS) dari aplikasi yang menggunakan proses dan kunci publik yang sama seperti instans DB RDS for MySQL.

Amazon RDS membuat sertifikat TLS dan menginstal sertifikat tersebut pada instans DB saat Amazon RDS menyediakan instans. Sertifikat ini ditandatangani oleh otoritas sertifikat. Sertifikat TLS mencakup titik akhir instans DB sebagai Nama Umum (CN) untuk sertifikat TLS agar melindungi dari serangan spoofing. Oleh karena itu, Anda hanya dapat menggunakan titik akhir klaster DB untuk terhubung ke klaster DB menggunakan TLS jika klien Anda mendukung Nama Alternatif Subjek (SAN). Jika tidak, Anda harus menggunakan titik akhir instans milik instans penulis.

Untuk informasi tentang mengunduh sertifikat, lihat .

Kami merekomendasikan Driver AWS JDBC untuk MySQL sebagai klien yang mendukung SAN dengan TLS. Untuk informasi selengkapnya tentang Driver AWS JDBC untuk MySQL dan petunjuk lengkap untuk menggunakannya, lihat Driver AWS JDBC untuk repositori MySQL. GitHub

Topik

- Mewajibkan koneksi TLS ke klaster DB Aurora MySQL
- Versi TLS untuk Aurora MySQL
- Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora MySQL
- Mengenkripsi koneksi ke klaster DB Aurora MySQL

Mewajibkan koneksi TLS ke klaster DB Aurora MySQL

Anda dapat mewajibkan agar semua koneksi pengguna ke klaster DB Aurora MySQL Anda menggunakan TLS dengan parameter klaster DB `require_secure_transport`. Secara default, parameter `require_secure_transport` diatur ke `OFF`. Anda dapat mengatur parameter `require_secure_transport` ke `ON` untuk mewajibkan TLS untuk koneksi ke klaster DB Anda.

Anda dapat mengatur nilai parameter `require_secure_transport` dengan memperbarui grup parameter klaster DB untuk klaster DB Anda. Anda tidak perlu mem-boot ulang klaster DB Anda agar perubahan dapat diterapkan. Untuk informasi selengkapnya tentang grup parameter, lihat Bekerja dengan grup parameter.

> **Note**
>
> Parameter `require_secure_transport` tersedia untuk Aurora MySQL versi 2 dan 3. Anda dapat mengatur parameter ini dalam grup parameter klaster DB kustom. Parameter ini tidak tersedia dalam grup parameter instans DB.

Saat parameter `require_secure_transport` diatur ke `ON` untuk klaster DB, klien basis data dapat terhubung dengannya jika klien basis data ini dapat membuat koneksi terenkripsi. Jika tidak, pesan kesalahan seperti yang berikut ini ditampilkan kepada klien:

```
MySQL Error 3159 (HY000): Connections using insecure transport are prohibited while --
require_secure_transport=ON.
```

Versi TLS untuk Aurora MySQL

Aurora MySQL mendukung Keamanan Lapisan Pengangkutan (TLS) versi 1.0, 1.1, 1.2, 1.2, dan 1.3. Mulai dari Aurora MySQL versi 3.04.0 dan lebih tinggi, Anda dapat menggunakan protokol TLS 1.3

untuk mengamankan koneksi Anda. Tabel berikut menunjukkan dukungan TLS untuk versi Aurora MySQL.

Versi Aurora MySQL	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Default
Aurora MySQL versi 2	Didukung	Didukung	Didukung	Tidak Support	Semua versi TLS yang didukung
Aurora MySQL versi 3 (di bawah 3.04.0)	Didukung	Didukung	Didukung	Tidak Support	Semua versi TLS yang didukung
Aurora MySQL versi 3 (3.04.0 dan di atas)	Tidak didukung	Tidak didukung	Didukung	Didukung	Semua versi TLS yang didukung

Important

Jika Anda menggunakan grup parameter kustom untuk klaster Aurora MySQL Anda dengan versi 2 dan versi yang lebih rendah dari 3.04.0, sebaiknya gunakan TLS 1.2 karena TLS 1.0 dan 1.1 kurang aman. Edisi komunitas MySQL 8.0.26 dan Aurora MySQL 3.03 dan versi minornya menghentikan dukungan untuk TLS versi 1.1 dan 1.0.

Edisi komunitas MySQL 8.0.28 dan Aurora MySQL versi 3.04.0 dan lebih tinggi yang kompatibel tidak mendukung TLS 1.1 dan TLS 1.0. Jika Anda menggunakan Aurora MySQL versi 3.04.0 dan yang lebih tinggi, jangan atur protokol TLS ke 1.0 dan 1.1 di grup parameter kustom Anda.

Untuk Aurora MySQL versi 3.04.0 dan yang lebih tinggi, pengaturan default-nya adalah TLS 1.3 dan TLS 1.2.

Anda dapat menggunakan parameter klaster DB `tls_version` untuk menunjukkan versi protokol yang diizinkan. Parameter klien serupa ada untuk sebagian besar alat klien atau driver basis data.

Beberapa klien lama mungkin tidak mendukung versi TLS yang lebih baru. Secara default, klaster DB mencoba menggunakan versi protokol TLS tertinggi yang diizinkan oleh konfigurasi server dan klien.

Atur parameter klaster DB `tls_version` ke salah satu nilai berikut:

- `TLSv1.3`
- `TLSv1.2`
- `TLSv1.1`
- `TLSv1`

Anda juga dapat mengatur parameter `tls_version` sebagai string daftar dipisahkan koma. Jika Anda ingin menggunakan protokol TLS 1.2 dan TLS 1.0, parameter `tls_version` harus menyertakan semua protokol dari protokol terendah hingga protokol tertinggi. Dalam hal ini, `tls_version` ditetapkan sebagai:

```
tls_version=TLSv1,TLSv1.1,TLSv1.2
```

Untuk informasi tentang memodifikasi parameter dalam grup parameter klaster DB, lihat Mengubah parameter dalam grup parameter klaster DB. Jika Anda menggunakan AWS CLI untuk memodifikasi parameter cluster `tls_version` DB, `ApplyMethod` harus diatur k`epending-reboot`. Jika metode aplikasinya adalah `pending-reboot`, perubahan parameter diterapkan setelah Anda menghentikan dan mengaktifkan ulang klaster DB yang terkait dengan grup parameter.

Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora MySQL

Dengan menggunakan cipher suite yang dapat dikonfigurasi, Anda dapat memiliki kontrol lebih besar atas keamanan koneksi basis data Anda. Anda dapat menentukan daftar cipher suite yang ingin Anda izinkan untuk mengamankan koneksi TLS klien ke basis data Anda. Dengan cipher suite yang dapat dikonfigurasi, Anda dapat mengontrol enkripsi koneksi yang diterima server basis data Anda. Melakukan hal ini akan mencegah penggunaan cipher yang tidak aman atau usang.

Cipher suite yang dapat dikonfigurasi didukung di Aurora MySQL versi 3 dan Aurora MySQL versi 2. Untuk menentukan daftar cipher TLS 1.2, TLS 1.1, atau TLS 1.0 yang diizinkan untuk mengenkripsi koneksi, modifikasi parameter klaster `ssl_cipher`. Atur parameter `ssl_cipher` dalam grup parameter klaster menggunakan AWS Management Console, AWS CLI, atau API RDS.

Atur parameter `ssl_cipher` ke string nilai cipher yang dipisahkan koma untuk versi TLS Anda. Untuk aplikasi klien, Anda dapat menentukan cipher yang akan digunakan untuk koneksi terenkripsi

dengan menggunakan opsi `--ssl-cipher` saat menghubungkan ke basis data. Untuk informasi selengkapnya tentang melakukan koneksi ke basis data Anda, lihat Menghubungkan ke klaster DB Amazon Aurora MySQL.

Mulai dari Aurora MySQL versi 3.04.0 dan lebih tinggi, Anda dapat menentukan cipher suite TLS 1.3. Untuk menentukan cipher suite TLS 1.3 yang diizinkan, ubah parameter `tls_ciphersuites` dalam grup parameter Anda. TLS 1.3 telah mengurangi jumlah cipher suite yang tersedia karena perubahan dalam konvensi penamaan yang menghapus mekanisme pertukaran kunci dan sertifikat yang digunakan. Atur `tls_ciphersuites` ke string nilai cipher dipisahkan koma untuk TLS 1.3.

Tabel berikut menunjukkan cipher yang didukung bersama dengan protokol enkripsi TLS dan versi mesin Aurora MySQL yang valid untuk setiap cipher.

Cipher	Protokol enkripsi	Versi Aurora MySQL yang didukung
`DHE-RSA-AES128-SHA`	TLS 1.0	3.01.0 dan lebih tinggi, semua di bawah 2.11.0
`DHE-RSA-AES128-SHA256`	TLS 1.2	3.01.0 dan lebih tinggi, semua di bawah 2.11.0
`DHE-RSA-AES128-GCM-SHA256`	TLS 1.2	3.01.0 dan lebih tinggi, semua di bawah 2.11.0
`DHE-RSA-AES256-SHA`	TLS 1.0	3.03.0 dan lebih rendah, semua di bawah 2.11.0
`DHE-RSA-AES256-SHA256`	TLS 1.2	3.01.0 dan lebih tinggi, semua di bawah 2.11.0
`DHE-RSA-AES256-GCM-SHA384`	TLS 1.2	3.01.0 dan lebih tinggi, semua di bawah 2.11.0
`ECDHE-RSA-AES128-SHA`	TLS 1.0	3.01.0 dan lebih tinggi, 2.09.3 dan lebih tinggi, 2.10.2 dan lebih tinggi

Cipher	Protokol enkripsi	Versi Aurora MySQL yang didukung
ECDHE-RSA-AES128-SHA256	TLS 1.2	3.01.0 dan lebih tinggi, 2.09.3 dan lebih tinggi, 2.10.2 dan lebih tinggi
ECDHE-RSA-AES128-GCM-SHA256	TLS 1.2	3.01.0 dan lebih tinggi, 2.09.3 dan lebih tinggi, 2.10.2 dan lebih tinggi
ECDHE-RSA-AES256-SHA	TLS 1.0	3.01.0 dan lebih tinggi, 2.09.3 dan lebih tinggi, 2.10.2 dan lebih tinggi
ECDHE-RSA-AES256-SHA384	TLS 1.2	3.01.0 dan lebih tinggi, 2.09.3 dan lebih tinggi, 2.10.2 dan lebih tinggi
ECDHE-RSA-AES256-GCM-SHA384	TLS 1.2	3.01.0 dan lebih tinggi, 2.09.3 dan lebih tinggi, 2.10.2 dan lebih tinggi
TLS_AES_128_GCM_SHA256	TLS 1.3	3.04.0 dan lebih tinggi
TLS_AES_256_GCM_SHA384	TLS 1.3	3.04.0 dan lebih tinggi
TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	3.04.0 dan lebih tinggi

> **Note**
>
> Cipher DHE-RSA hanya didukung oleh versi Aurora MySQL sebelum 2.11.0. Versi 2.11.0 dan lebih tinggi mendukung cipher ECDHE saja.

Untuk informasi tentang memodifikasi parameter dalam grup parameter klaster DB, lihat Mengubah parameter dalam grup parameter klaster DB. Jika Anda menggunakan CLI untuk memodifikasi parameter klaster DB `ssl_cipher`, pastikan untuk mengatur `ApplyMethod` ke `pending-reboot`. Jika metode aplikasinya adalah `pending-reboot`, perubahan parameter diterapkan setelah Anda menghentikan dan mengaktifkan ulang klaster DB yang terkait dengan grup parameter.

Anda juga dapat menggunakan perintah describe-engine-default-cluster-parameter CLI untuk menentukan rangkaian sandi mana yang saat ini didukung untuk keluarga grup parameter tertentu. Contoh berikut menunjukkan cara mendapatkan nilai yang diizinkan untuk parameter klaster `ssl_cipher` untuk Aurora MySQL versi 2.

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
mysql5.7

        ...some output truncated...
 {
  "ParameterName": "ssl_cipher",
  "ParameterValue": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,ECDHE-RSA-
AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-RSA-AES256-
SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-SHA384",
  "Description": "The list of permissible ciphers for connection encryption.",
  "Source": "system",
  "ApplyType": "static",
  "DataType": "list",
  "AllowedValues": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,ECDHE-
RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-SHA256,ECDHE-RSA-AES256-
SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-SHA384",
  "IsModifiable": true,
  "SupportedEngineModes": [
   "provisioned"
  ]
 },
       ...some output truncated...
```

Untuk informasi selengkapnya tentang cipher, lihat variabel ssl_cipher dalam dokumentasi MySQL. Untuk informasi selengkapnya tentang format cipher suite, lihat openssl-ciphers list format dan dokumentasi openssl-ciphers string format di situs web OpenSSL.

Mengenkripsi koneksi ke klaster DB Aurora MySQL

Untuk mengenkripsi koneksi menggunakan klien `mysql` default, luncurkan klien mysql menggunakan parameter `--ssl-ca` untuk merujuk ke kunci publik, misalnya:

Untuk MySQL 5.7 dan 8.0:

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
--ssl-ca=full_path_to_CA_certificate --ssl-mode=VERIFY_IDENTITY
```

Untuk MySQL 5.6:

```
mysql -h myinstance.123456789012.rds-us-east-1.amazonaws.com
--ssl-ca=full_path_to_CA_certificate --ssl-verify-server-cert
```

Ganti *`full_path_to_CA_certificate`* dengan jalur lengkap ke sertifikat Otoritas Sertifikat (CA) Anda. Untuk informasi tentang mengunduh sertifikat, lihat .

Anda dapat mewajibkan koneksi TLS untuk akun pengguna tertentu. Misalnya, Anda dapat menggunakan salah satu pernyataan berikut, bergantung pada versi MySQL Anda, untuk mewajibkan koneksi TLS pada `encrypted_user` akun pengguna.

Untuk MySQL 5.7 dan 8.0:

```
ALTER USER 'encrypted_user'@'%' REQUIRE SSL;
```

Untuk MySQL 5.6:

```
GRANT USAGE ON *.* TO 'encrypted_user'@'%' REQUIRE SSL;
```

Saat Anda menggunakan Proksi RDS, Anda terhubung ke titik akhir proksi, bukan titik akhir klaster biasa. Anda dapat menjadikan SSL/TLS wajib atau opsional untuk koneksi ke proksi, dengan cara yang sama seperti untuk koneksi langsung ke klaster DB Aurora. Untuk informasi tentang menggunakan Proksi RDS, lihat Menggunakan Proksi Amazon RDS untuk Aurora.

> **Note**
>
> Untuk informasi selengkapnya tentang koneksi TLS dengan MySQL, lihat dokumentasi MySQL.

Memperbarui aplikasi untuk terhubung ke klaster DB Aurora MySQL menggunakan sertifikat TLS baru

Sejak 13 Januari 2023, Amazon RDS telah menerbitkan serifikat Otoritas Sertifikat (CA) baru untuk terhubung ke klaster DB Aurora menggunakan Keamanan Lapisan Pengangkutan (TLS). Setelah itu, Anda dapat menemukan informasi tentang pembaruan aplikasi untuk menggunakan sertifikat baru.

Topik ini dapat membantu Anda menentukan apakah aplikasi klien menggunakan TLS untuk terhubung ke klaster DB Anda. Jika demikian, Anda dapat memeriksa lebih lanjut apakah aplikasi tersebut memerlukan verifikasi sertifikat untuk terhubung.

> **Note**
>
> Beberapa aplikasi dikonfigurasi untuk terhubung ke klaster DB Aurora MySQL hanya jika aplikasi tersebut berhasil memverifikasi sertifikat pada server.
> Untuk aplikasi tersebut, Anda harus memperbarui penyimpanan kepercayaan aplikasi klien untuk menyertakan sertifikat CA baru.

Setelah memperbarui sertifikat CA di penyimpanan kepercayaan aplikasi klien, Anda dapat merotasi sertifikat di klaster DB Anda. Sebaiknya Anda menguji prosedur ini di lingkungan pengembangan dan penahapan sebelum menerapkannya di lingkungan produksi Anda.

Untuk informasi selengkapnya tentang rotasi sertifikat, lihat Merotasi sertifikat SSL/TLS. Untuk informasi selengkapnya tentang cara mengunduh sertifikat, lihat . Untuk informasi tetang cara menggunakan TLS dengan klaster DB Aurora MySQL, lihat Menggunakan TLS dengan klaster DB Aurora MySQL.

Topik

- Menentukan apakah ada aplikasi yang tersambung ke klaster DB Aurora MySQL menggunakan TLS
- Menentukan apakah klien memerlukan verifikasi sertifikat untuk terhubung
- Memperbarui penyimpanan kepercayaan aplikasi Anda
- Contoh kode Java untuk membangun koneksi TLS

Menentukan apakah ada aplikasi yang tersambung ke klaster DB Aurora MySQL menggunakan TLS

Jika Anda menggunakan Aurora MySQL versi 2 (kompatibel dengan MySQL 5.7) dan Skema Performa diaktifkan, jalankan kueri berikut untuk memeriksa apakah koneksi menggunakan TLS. Untuk informasi tentang cara mengaktifkan Skema Performa, lihat Memulai cepat Skema Performa dalam dokumentasi MySQL.

```
mysql> SELECT id, user, host, connection_type
       FROM performance_schema.threads pst
       INNER JOIN information_schema.processlist isp
       ON pst.processlist_id = isp.id;
```

Dalam output contoh ini, Anda dapat melihat sesi Anda sendiri (`admin`) dan aplikasi yang masuk sebagai `webapp1` menggunakan TLS.

```
+----+-----------------+------------------+-----------------+
| id | user            | host             | connection_type |
+----+-----------------+------------------+-----------------+
|  8 | admin           | 10.0.4.249:42590 | SSL/TLS         |
|  4 | event_scheduler | localhost        | NULL            |
| 10 | webapp1         | 159.28.1.1:42189 | SSL/TLS         |
+----+-----------------+------------------+-----------------+
3 rows in set (0.00 sec)
```

Menentukan apakah klien memerlukan verifikasi sertifikat untuk terhubung

Anda dapat memeriksa apakah klien JDBC dan klien MySQL memerlukan verifikasi sertifikat untuk terhubung.

JDBC

Contoh MySQL Connector/J 8.0 berikut menunjukkan satu cara untuk memeriksa properti koneksi JDBC aplikasi untuk menentukan apakah koneksi yang berhasil memerlukan sertifikat yang valid. Untuk informasi selengkapnya tentang semua opsi koneksi JDBC untuk MySQL, lihat Properti konfigurasi dalam dokumentasi MySQL.

Saat menggunakan MySQL Connector/J 8.0, koneksi TLS memerlukan verifikasi terhadap sertifikat CA server jika properti koneksi Anda memiliki `sslMode` yang diatur ke `VERIFY_CA` atau `VERIFY_IDENTITY`, seperti pada contoh berikut.

```
Properties properties = new Properties();
properties.setProperty("sslMode", "VERIFY_IDENTITY");
properties.put("user", DB_USER);
properties.put("password", DB_PASSWORD);
```

> **Note**
>
> Jika Anda menggunakan MySQL Java Connector v5.1.38 atau yang lebih baru, atau MySQL Java Connector v8.0.9 atau yang lebih baru untuk terhubung ke basis data Anda, meski Anda belum mengonfigurasi aplikasi secara ekplisit untuk menggunakan TLS saat terhubung ke basis data Anda, driver klien ini akan menggunakan TLS secara default. Selain itu, saat menggunakan TLS, aplikasi akan melakukan verifikasi sertifikat parsial dan gagal terhubung jika sertifikat server basis data kedaluwarsa.

MySQL

Contoh Klien MySQL berikut menunjukkan dua cara untuk memeriksa koneksi MySQL skrip untuk menentukan apakah koneksi yang berhasil memerlukan sertifikat yang valid. Untuk informasi selengkapnya tentang semua opsi koneksi dengan Klien MySQL, lihat Konfigurasi sisi klien untuk koneksi terenkripsi dalam dokumentasi MySQL.

Saat menggunakan Klien MySQL 5.7 atau MySQL 8.0, koneksi TLS memerlukan verifikasi terhadap sertifikat CA server jika, untuk opsi `--ssl-mode`, Anda menentukan `VERIFY_CA` atau `VERIFY_IDENTITY`, seperti pada contoh berikut.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-mode=VERIFY_CA
```

Saat menggunakan Klien MySQL 5.6, koneksi SSL memerlukan verifikasi terhadap sertifikat CA server jika Anda menentukan opsi `--ssl-verify-server-cert`, seperti pada contoh berikut.

```
mysql -h mysql-database.rds.amazonaws.com -uadmin -ppassword --ssl-ca=/tmp/ssl-cert.pem
 --ssl-verify-server-cert
```

Memperbarui penyimpanan kepercayaan aplikasi Anda

Untuk informasi tentang cara memperbarui penyimpanan kepercayaan untuk aplikasi MySQL, lihat Menginstal sertifikat SSL dalam dokumentasi MySQL.

> **Note**
>
> Saat memperbarui penyimpanan kepercayaan, Anda dapat mempertahankan sertifikat lama selain menambahkan sertifikat baru.

Memperbarui penyimpanan kepercayaan aplikasi Anda untuk JDBC

Anda dapat memperbarui penyimpanan kepercayaan untuk aplikasi yang menggunakan JDBC untuk koneksi TLS.

Untuk informasi tentang cara mengunduh sertifikat root, lihat .

Untuk contoh skrip yang mengimpor sertifikat, lihat Contoh skrip untuk mengimpor sertifikat ke trust store Anda.

Jika Anda menggunakan driver JDBC mysql dalam aplikasi, atur properti berikut dalam aplikasi.

```
System.setProperty("javax.net.ssl.trustStore", certs);
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

> **Note**
>
> Tentukan kata sandi selain perintah yang ditampilkan di sini sebagai praktik keamanan terbaik.

Saat Anda memulai aplikasi, atur properti berikut.

```
java -Djavax.net.ssl.trustStore=/path_to_truststore/MyTruststore.jks -
Djavax.net.ssl.trustStorePassword=my_truststore_password com.companyName.MyApplication
```

Contoh kode Java untuk membangun koneksi TLS

Contoh kode berikut menunjukkan cara menyiapkan koneksi SSL yang memvalidasi sertifikat server menggunakan JDBC.

```
public class MySQLSSLTest {
```

```
        private static final String DB_USER = "user name";
        private static final String DB_PASSWORD = "password";
        // This key store has only the prod root ca.
        private static final String KEY_STORE_FILE_PATH = "file-path-to-keystore";
        private static final String KEY_STORE_PASS = "keystore-password";

    public static void test(String[] args) throws Exception {
        Class.forName("com.mysql.jdbc.Driver");


        System.setProperty("javax.net.ssl.trustStore", KEY_STORE_FILE_PATH);
        System.setProperty("javax.net.ssl.trustStorePassword", KEY_STORE_PASS);

        Properties properties = new Properties();
        properties.setProperty("sslMode", "VERIFY_IDENTITY");
        properties.put("user", DB_USER);
        properties.put("password", DB_PASSWORD);

        Connection connection = DriverManager.getConnection("jdbc:mysql://jagdeeps-ssl-
test.cni62e2e7kwh.us-east-1.rds.amazonaws.com:3306",properties);
        Statement stmt=connection.createStatement();

        ResultSet rs=stmt.executeQuery("SELECT 1 from dual");

        return;
    }
}
```

Important

Setelah Anda menentukan bahwa koneksi database Anda menggunakan TLS dan telah memperbarui toko kepercayaan aplikasi Anda, Anda dapat memperbarui database Anda untuk menggunakan sertifikat rds-ca-rsa 2048-g1. Untuk petunjuk, lihat langkah 3 dalam Memperbarui sertifikat CA Anda dengan memodifikasi instans cluster DB.

Menggunakan autentikasi Kerberos untuk Aurora MySQL

Anda dapat menggunakan autentikasi Kerberos untuk mengautentikasi pengguna saat mereka terhubung ke klaster DB Aurora MySQL Anda. Untuk melakukannya, lakukan konfigurasi klaster DB Anda agar menggunakan AWS Directory Service for Microsoft Active Directory untuk autentikasi Kerberos. AWS Directory Service for Microsoft Active Directory juga disebut AWS Managed Microsoft AD. Ini adalah fitur yang tersedia dengan AWS Directory Service. Untuk mempelajari selengkapnya, lihat Apa itu AWS Directory Service? di dalam Panduan Administrasi AWS Directory Service.

Untuk memulai, buat direktori AWS Managed Microsoft AD untuk menyimpan kredensial pengguna. Kemudian, berikan domain Active Directory dan informasi lainnya ke klaster DB Aurora MySQL Anda. Saat pengguna mengautentikasi dengan klaster DB Aurora MySQL, permintaan autentikasi diteruskan ke direktori AWS Managed Microsoft AD.

Menyimpan semua kredensial Anda di direktori yang sama dapat menghemat waktu dan tenaga Anda. Dengan pendekatan ini, Anda memiliki sebuah lokasi terpusat untuk menyimpan dan mengelola kredensial bagi beberapa klaster DB. Menggunakan direktori juga dapat meningkatkan profil keamanan keseluruhan Anda.

Selain itu, Anda dapat mengakses kredensial dari Microsoft Active Directory on-premise Anda sendiri. Untuk melakukannya, buat hubungan domain tepercaya sehingga direktori AWS Managed Microsoft AD mempercayai Microsoft Active Directory on-premise Anda. Dengan cara ini, pengguna Anda dapat mengakses klaster DB Aurora MySQL Anda dengan pengalaman masuk tunggal (SSO) Windows yang sama seperti ketika mereka mengakses beban kerja di jaringan on-premise Anda.

Basis data dapat menggunakan Kerberos, AWS Identity and Access Management (IAM), atau autentikasi Kerberos dan IAM. Namun, karena autentikasi Kerberos dan IAM menyediakan metode autentikasi yang berbeda, pengguna tertentu dapat login ke basis data hanya menggunakan salah satu metode autentikasi, dan tidak bisa keduanya. Untuk informasi selengkapnya tentang autentikasi IAM, lihat Autentikasi basis data IAM.

Daftar Isi

- Ikhtisar autentikasi Kerberos untuk klaster DB Aurora MySQL
- Batasan autentikasi Kerberos untuk Aurora MySQL
- Menyiapkan autentikasi Kerberos untuk klaster DB Aurora MySQL
  - Langkah 1: Buat direktori menggunakan AWS Managed Microsoft AD
  - Langkah 2: (Opsional) Buat kepercayaan untuk Active Directory on-premise

- Langkah 3: Buat peran IAM untuk digunakan oleh Amazon Aurora
- Langkah 4: Buat dan konfigurasikan pengguna
- Langkah 5: Buat atau modifikasi klaster DB Aurora MySQL
- Langkah 6: Buat pengguna MySQL Aurora yang menggunakan autentikasi Kerberos
  - Memodifikasi login Aurora MySQL yang ada
- Langkah 7: Konfigurasi klien MySQL
- Langkah 8: (Opsional) Lakukan konfigurasi perbandingan nama pengguna yang tidak peka huruf besar/kecil
- Membuat koneksi dengan Aurora MySQL lewat autentikasi Kerberos
  - Menggunakan login Kerberos Aurora MySQL untuk terhubung ke klaster DB
  - Autentikasi Kerberos dengan basis data global Aurora
  - Migrasi dari RDS for MySQL ke Aurora MySQL
  - Mencegah caching tiket
  - Pencatatan log untuk autentikasi Kerberos
- Mengelola klaster DB di dalam domain
  - Memahami keanggotaan domain

Ikhtisar autentikasi Kerberos untuk klaster DB Aurora MySQL

Untuk menyiapkan autentikasi Kerberos untuk klaster DB Aurora MySQL, selesaikan langkah-langkah umum berikut. Langkah ini dijelaskan secara lebih mendetail nanti.

1. Gunakan AWS Managed Microsoft AD untuk membuat direktori AWS Managed Microsoft AD. Anda dapat menggunakan AWS Management Console, AWS CLI, atau AWS Directory Service untuk membuat direktori. Untuk petunjuk mendetail, lihat Membuat direktori AWS Managed Microsoft AD Anda di Panduan AdministrasiAWS Directory Service.
2. Buat peran AWS Identity and Access Management (IAM) yang menggunakan kebijakan IAM terkelola `AmazonRDSDirectoryServiceAccess`. Peran ini memungkinkan Amazon Aurora untuk melakukan panggilan ke direktori Anda.

   Agar peran dapat mengizinkan akses, titik akhir AWS Security Token Service (AWS STS) harus diaktifkan di Wilayah AWS untuk akun AWS Anda. Titik akhir AWS STS aktif secara default di semua Wilayah AWS, dan Anda dapat menggunakannya tanpa tindakan lebih lanjut. Lihat

informasi yang lebih lengkap di Mengaktifkan dan menonaktifkan AWS STS di Wilayah AWS dalam Panduan Pengguna IAM.

3. Buat dan konfigurasikan pengguna dalam direktori AWS Managed Microsoft AD dengan menggunakan alat Microsoft Active Directory. Untuk informasi selengkapnya tentang membuat pengguna di Active Directory Anda, lihat Mengelola pengguna dan grup di Microsoft AD yang dikelola AWS di Panduan Administrasi AWS Directory Service.
4. Buat atau modifikasi klaster DB Aurora MySQL. Jika Anda menggunakan CLI atau API RDS dalam permintaan pembuatan, tentukan pengidentifikasi domain dengan parameter `Domain`. Gunakan pengidentifikasi `d-*` yang dihasilkan saat Anda membuat direktori Anda dan nama peran IAM yang Anda buat.

   Jika Anda memodifikasi klaster DB Aurora MySQL yang sudah ada untuk menggunakan autentikasi Kerberos, atur domain dan parameter peran IAM untuk klaster DB. Cari klaster DB di dalam VPC yang sama dengan direktori domain.
5. Gunakan kredensial pengguna primer Amazon RDS untuk terhubung ke klaster DB Aurora MySQL. Buat pengguna basis data di Aurora MySQL dengan menggunakan instruksi di Langkah 6: Buat pengguna MySQL Aurora yang menggunakan autentikasi Kerberos.

   Pengguna yang Anda buat dengan cara ini dapat login ke klaster DB Aurora MySQL menggunakan autentikasi Kerberos. Untuk informasi selengkapnya, lihat Membuat koneksi dengan Aurora MySQL lewat autentikasi Kerberos.

Untuk menggunakan autentikasi Kerberos dengan Microsoft Active Directory on-premise atau yang di-host mandiri, buat sebuah trust forest. Trust forest adalah hubungan kepercayaan antara dua kelompok domain. Kepercayaan bisa satu arah atau dua arah. Untuk informasi selengkapnya tentang menyiapkan trust forest menggunakan AWS Directory Service, lihat Kapan membuat hubungan kepercayaan dalam Panduan Administrasi AWS Directory Service.

Batasan autentikasi Kerberos untuk Aurora MySQL

Pembatasan berikut ini berlaku untuk autentikasi Kerberos untuk Aurora MySQL:

- Autentikasi Kerberos didukung untuk Aurora MySQL versi 3.03 dan lebih tinggi.

  Untuk informasi tentang dukungan Wilayah AWS, lihat Autentikasi Kerberos dengan Aurora MySQL.

- Untuk menggunakan autentikasi Kerberos dengan Aurora MySQL, klien atau konektor MySQL Anda harus menggunakan versi 8.0.26 atau lebih tinggi pada platform Unix, 8.0.27 atau lebih tinggi di Windows. Jika tidak, plugin `authentication_kerberos_client` sisi klien tidak tersedia dan Anda tidak dapat mengautentikasi.
- Hanya AWS Managed Microsoft AD yang didukung di Aurora MySQL. Namun, Anda dapat menggabungkan klaster DB Aurora MySQL ke domain Microsoft AD Terkelola bersama yang dimiliki oleh akun-akun yang berbeda di dalam Wilayah AWS yang sama.

  Anda juga dapat menggunakan Active Directory on-premise Anda sendiri. Untuk informasi selengkapnya, lihat Langkah 2: (Opsional) Buat kepercayaan untuk Active Directory on-premise
- Saat menggunakan Kerberos untuk mengautentikasi pengguna yang terhubung ke klaster Aurora MySQL dari klien MySQL atau dari driver pada sistem operasi Windows, secara default huruf besar/kecil karakter nama pengguna basis data harus sesuai dengan huruf besar/kecil pengguna di Active Directory. Misalnya, jika pengguna di Active Directory muncul sebagai `Admin`, nama pengguna basis data harus `Admin`.

  Namun, Anda sekarang dapat menggunakan perbandingan nama pengguna yang tidak peka huruf besar/kecil dengan plugin `authentication_kerberos`. Untuk informasi selengkapnya, lihat Langkah 8: (Opsional) Lakukan konfigurasi perbandingan nama pengguna yang tidak peka huruf besar/kecil.
- Anda harus melakukan boot ulang instans DB pembaca setelah mengaktifkan fitur untuk menginstal plugin `authentication_kerberos`.
- Replikasi ke instans DB yang tidak mendukung plugin `authentication_kerberos` dapat menyebabkan kegagalan replikasi.
- Agar basis data global Aurora dapat menggunakan autentikasi Kerberos, Anda harus mengonfigurasinya untuk setiap klaster DB di dalam basis data global.
- Nama domain harus kurang dari 62 karakter.
- Jangan memodifikasi port klaster DB setelah mengaktifkan autentikasi Kerberos. Jika Anda memodifikasi port, autentikasi Kerberos tidak akan berfungsi lagi.

Menyiapkan autentikasi Kerberos untuk klaster DB Aurora MySQL

Gunakan AWS Managed Microsoft AD untuk menyiapkan autentikasi Kerberos untuk klaster DB Aurora MySQL. Untuk menyiapkan autentikasi Kerberos, lakukan langkah-langkah berikut.

Topik

- Langkah 1: Buat direktori menggunakan AWS Managed Microsoft AD
- Langkah 2: (Opsional) Buat kepercayaan untuk Active Directory on-premise
- Langkah 3: Buat peran IAM untuk digunakan oleh Amazon Aurora
- Langkah 4: Buat dan konfigurasikan pengguna
- Langkah 5: Buat atau modifikasi klaster DB Aurora MySQL
- Langkah 6: Buat pengguna MySQL Aurora yang menggunakan autentikasi Kerberos
- Langkah 7: Konfigurasi klien MySQL
- Langkah 8: (Opsional) Lakukan konfigurasi perbandingan nama pengguna yang tidak peka huruf besar/kecil

Langkah 1: Buat direktori menggunakan AWS Managed Microsoft AD

AWS Directory Service membuat Active Directory yang dikelola penuh di AWS Cloud. Saat Anda membuat direktori AWS Managed Microsoft AD, AWS Directory Service membuat dua server pengendali domain dan Sistem Nama Domain (DNS) atas nama Anda. Server direktori dibuat di subnet berbeda di VPC. Redundansi ini membantu memastikan bahwa direktori Anda tetap dapat diakses meski terjadi kegagalan.

Saat Anda membuat direktori AWS Managed Microsoft AD, AWS Directory Service melakukan tugas berikut ini atas nama Anda:

- Menyiapkan Active Directory dalam VPC.
- Membuat akun administrator direktori dengan nama pengguna `Admin` dan kata sandi yang ditentukan. Anda menggunakan akun ini untuk mengelola direktori Anda.

> **Note**
>
> Pastikan untuk menyimpan kata sandi ini. AWS Directory Service tidak menyimpannya. Anda dapat mengaturnya ulang, tetapi tidak dapat mengambilnya.

- Membuat grup keamanan untuk pengendali direktori.

Saat Anda meluncurkan sebuah AWS Managed Microsoft AD, AWS membuat Unit Organisasi (OU) yang berisi semua objek direktori Anda. OU ini memiliki nama NetBIOS yang Anda masukkan saat membuat direktori Anda. OU ini berada di root domain, yang dimiliki dan dikelola oleh AWS.

Akun `Admin` yang dibuat dengan direktori AWS Managed Microsoft AD Anda memiliki izin untuk aktivitas administratif paling umum untuk OU Anda, termasuk:

- Membuat, memperbarui, atau menghapus pengguna
- Tambahkan sumber daya ke domain Anda, seperti server file atau server cetak, kemudian tetapkan izin untuk sumber daya itu kepada pengguna di OU Anda
- Membuat OU dan kontainer tambahan
- Mendelegasikan kewenangan
- Memulihkan objek yang dihapus dari Keranjang Sampah Active Directory
- Jalankan PowerShell modul AD dan DNS Windows pada Layanan Web Direktori Aktif

Akun `Admin` juga memiliki hak untuk melakukan aktivitas di seluruh domain berikut:

- Mengelola konfigurasi DNS (menambahkan, menghapus, atau memperbarui catatan, zona, dan penerus)
- Melihat log peristiwa DNS
- Melihat log peristiwa keamanan

Untuk membuat direktori dengan AWS Managed Microsoft AD

1. Masuk ke AWS Management Console dan buka konsol AWS Directory Service di https://console.aws.amazon.com/directoryservicev2/.
2. Di panel navigasi, pilih Direktori, lalu pilih Siapkan Direktori.
3. Pilih AWS Managed Microsoft AD. AWS Managed Microsoft AD adalah satu-satunya opsi yang saat ini dapat Anda gunakan dengan Amazon RDS.
4. Masukkan informasi berikut:

   Nama DNS Direktori

   Nama yang sepenuhnya memenuhi syarat direktori, seperti **`corp.example.com`**.

   Nama NetBIOS direktori

   Nama singkat direktori, seperti **`CORP`**.

   Deskripsi direktori

   (Opsional) Deskripsi direktori.

Kata sandi admin

Kata sandi untuk administrator direktori. Proses pembuatan direktori menciptakan akun administrator dengan nama pengguna Admin dan kata sandi ini.

Kata sandi administrator direktori dan tidak boleh menyertakan kata “admin”. Kata sandi peka terhadap huruf besar/kecil dan harus terdiri dari 8-64 karakter. Kata sandi juga harus berisi setidaknya satu karakter dari tiga di antara empat kategori berikut:

- Huruf kecil (a-z)
- Huruf besar (A-Z)
- Angka (0–9)
- Karakter non-alfanumerik (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

Konfirmasikan kata sandi

Kata sandi administrator dimasukkan kembali.

5. Pilih Selanjutnya.
6. Masukkan informasi berikut di bagian Jaringan, lalu pilih Berikutnya:

   VPC

   VPC untuk direktori. Buat klaster DB Aurora MySQL di VPC yang sama ini.

   Subnet

   Subnet untuk server direktori. Kedua subnet harus berada di Zona Ketersediaan yang berbeda.
7. Tinjau informasi direktori dan buat perubahan yang diperlukan. Jika informasi sudah benar, pilih Buat direktori.

Dibutuhkan beberapa menit untuk membuat direktori. Setelah direktori berhasil dibuat, nilai Status berubah menjadi Aktif.

Untuk melihat informasi tentang direktori Anda, pilih nama direktori di daftar direktori. Catat nilai ID Direktori karena Anda memerlukan nilai ini saat membuat atau memodifikasi klaster DB Aurora MySQL Anda.

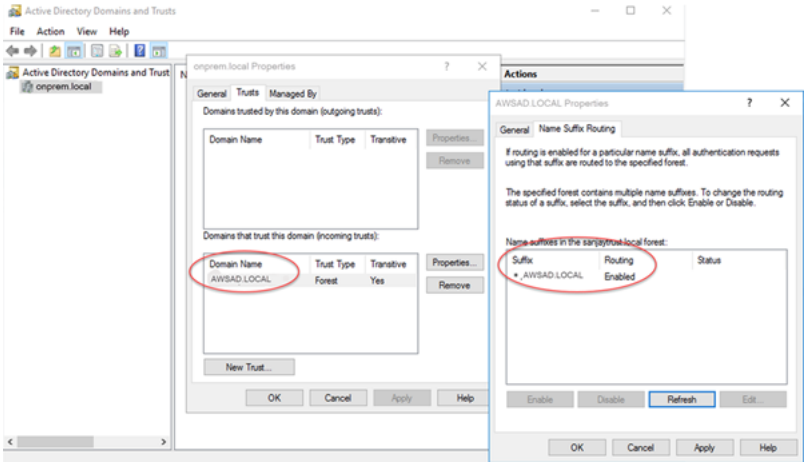
Langkah 2: (Opsional) Buat kepercayaan untuk Active Directory on-premise

Jika Anda tidak berencana menggunakan Microsoft Active Directory on-premise Anda sendiri, langsung lompat ke Langkah 3: Buat peran IAM untuk digunakan oleh Amazon Aurora.

Untuk menggunakan autentikasi Kerberos dengan Active Directory on-premise, Anda perlu membuat hubungan domain tepercaya menggunakan kepercayaan forest antara Microsoft Active Directory on-premise Anda dan direktori AWS Managed Microsoft AD (dibuat di Langkah 1: Buat direktori menggunakan AWS Managed Microsoft AD). Kepercayaan dapat bersifat satu arah, dimana direktori AWS Managed Microsoft AD mempercayai Microsoft Active Directory on-premise. Kepercayaan juga dapat bersifat dua arah, yang mana kedua Active Directory saling percaya. Untuk informasi selengkapnya tentang menyiapkan kepercayaan menggunakan AWS Directory Service, lihat Kapan membuat hubungan kepercayaan dalam Panduan Administrasi AWS Directory Service.

> **Note**
>
> Jika Anda menggunakan Microsoft Active Directory on-premise:
>
> - Klien Windows harus terhubung menggunakan nama domain AWS Directory Service di titik akhir, bukan rds.amazonaws.com. Untuk informasi selengkapnya, lihat Membuat koneksi dengan Aurora MySQL lewat autentikasi Kerberos.
> - Klien Windows tidak dapat terhubung menggunakan titik akhir kustom Aurora. Untuk mempelajari selengkapnya, lihat Manajemen koneksi Amazon Aurora.
> - Untuk basis data global:
>   - Klien Windows dapat terhubung menggunakan titik akhir instans atau titik akhir klaster di Wilayah AWS utama basis data global saja.
>   - Klien Windows tidak dapat terhubung menggunakan titik akhir klaster di Wilayah AWS sekunder.

Pastikan bahwa nama domain Microsoft Active Directory on-premise Anda mencakup perutean sufiks DNS yang sesuai dengan hubungan kepercayaan yang baru dibuat. Tangkapan layar berikut menunjukkan sebuah contoh.



Langkah 3: Buat peran IAM untuk digunakan oleh Amazon Aurora

Agar Amazon Aurora memanggil AWS Directory Service untuk Anda, Anda memerlukan peran AWS Identity and Access Management (IAM) yang menggunakan kebijakan IAM terkelola `AmazonRDSDirectoryServiceAccess`. Peran ini memungkinkan Aurora untuk melakukan panggilan ke AWS Directory Service.

Ketika Anda membuat klaster DB menggunakan AWS Management Console, dan Anda memiliki izin `iam:CreateRole`, konsol akan membuat peran ini secara otomatis. Dalam hal ini, nama perannya adalah `rds-directoryservice-kerberos-access-role`. Jika tidak, Anda harus membuat peran IAM secara manual. Saat Anda membuat peran IAM ini, pilih `Directory Service`, lalu lampirkan kebijakan terkelola AWS `AmazonRDSDirectoryServiceAccess` ke peran itu.

Lihat informasi yang lebih lengkap tentang membuat peran IAM untuk sebuah layanan di Membuat peran untuk melimpahkan izin ke layanan AWS dalam Panduan Pengguna IAM.

Anda memiliki opsi untuk membuat kebijakan dengan izin yang diperlukan alih-alih menggunakan kebijakan IAM yang dikelola `AmazonRDSDirectoryServiceAccess`. Untuk melakukannya, peran IAM harus memiliki kebijakan kepercayaan IAM berikut.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
```

```
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Peran ini juga harus memiliki kebijakan peran IAM berikut.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

Langkah 4: Buat dan konfigurasikan pengguna

Anda dapat membuat pengguna dengan alat Pengguna Active Directory dan Komputer. Alat ini adalah bagian dari alat Active Directory Domain Services dan Active Directory Lightweight Directory Services. Pengguna mewakili orang atau entitas individual yang memiliki akses ke direktori Anda.

Untuk membuat pengguna di direktori AWS Directory Service, Anda menggunakan instans on-premise atau Amazon EC2 berdasarkan Microsoft Windows yang digabungkan ke direktori AWS Directory Service Anda. Anda harus login ke instans sebagai pengguna yang memiliki hak istimewa untuk membuat pengguna. Untuk informasi selengkapnya, lihat Mengelola pengguna dan grup di AWS Managed Microsoft AD di Panduan Administrasi Layanan Direktori AWS.

Langkah 5: Buat atau modifikasi klaster DB Aurora MySQL

Buat atau modifikasi klaster DB Aurora MySQL untuk penggunaan dengan direktori Anda. Anda dapat menggunakan konsol, AWS CLI, atau API RDS untuk mengaitkan klaster DB dengan direktori. Anda dapat melakukan tugas ini dengan salah satu cara berikut:

- Buat cluster Aurora MySQL DB baru menggunakan konsol, create-db-clusterperintah CLI, atau operasi CreateDBCluster RDS API.

  Untuk petunjuk, lihat Membuat klaster DB Amazon Aurora.
- Ubah cluster Aurora MySQL DB yang ada menggunakan konsol, modify-db-clusterperintah CLI, atau operasi ModifyDBCluster RDS API.

  Untuk petunjuk, lihat Memodifikasi klaster DB Amazon Aurora.
- Kembalikan cluster DB MySQL Aurora dari snapshot DB menggunakan konsol, perintah restore-db-cluster-fromCLI -snapshot, atau operasi API RDS RestoreDB. ClusterFromSnapshot

  Untuk petunjuk, lihat Memulihkan dari snapshot klaster DB.
- Kembalikan cluster DB MySQL Aurora ke point-in-time menggunakan konsol, perintah - restore-db-cluster-topoint-in-timeCLI, atau operasi API RDS RestoreDB. ClusterToPointInTime

  Untuk petunjuk, lihat Memulihkan klaster DB ke waktu tertentu.

Autentikasi Kerberos hanya didukung untuk klaster DB Aurora MySQL dalam VPC. Klaster DB boleh berada dalam VPC yang sama dengan direktori, atau dalam VPC yang berbeda. VPC klaster DB harus memiliki grup keamanan VPC yang memungkinkan komunikasi keluar ke direktori Anda.

Konsol

Saat Anda menggunakan konsol untuk membuat, memodifikasi, atau memulihkan klaster DB, pilih Autentikasi Kerberos di bagian Autentikasi basis data. Pilih Jelajah Direktori lalu pilih direktori, atau pilih Buat direktori baru.

AWS CLI

Saat Anda menggunakan AWS CLI atau API RDS, kaitkan klaster DB dengan direktori. Parameter berikut diperlukan agar klaster DB dapat menggunakan direktori domain yang Anda buat:

- Untuk parameter , gunakan pengidentifikasi domain (pengidentifikasi "d-*") yang dihasilkan saat Anda membuat direktori.

- Untuk parameter `--domain-iam-role-name`, gunakan peran yang Anda buat yang menggunakan kebijakan IAM terkelola `AmazonRDSDirectoryServiceAccess`.

Misalnya, perintah CLI berikut memodifikasi klaster DB untuk menggunakan direktori.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --domain d-ID \
    --domain-iam-role-name role-name
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --domain d-ID ^
    --domain-iam-role-name role-name
```

> **Important**
>
> Jika Anda memodifikasi klaster DB untuk mengaktifkan autentikasi Kerberos, lakukan boot ulang terhadap instans DB pembaca setelah melakukan perubahan.

Langkah 6: Buat pengguna MySQL Aurora yang menggunakan autentikasi Kerberos

Klaster DB digabungkan ke domain AWS Managed Microsoft AD. Jadi, Anda dapat membuat pengguna Aurora MySQL dari pengguna Active Directory di domain Anda. Izin basis data dikelola melalui izin Aurora MySQL standar yang diberikan kepada dan dicabut dari pengguna ini.

Anda dapat mengizinkan pengguna Active Directory untuk mengautentikasi dengan Aurora MySQL. Untuk melakukannya, pertama-tama gunakan kredensial pengguna primer Amazon RDS untuk terhubung ke klaster DB Aurora MySQL seperti klaster DB lainnya. Setelah Anda login, buat pengguna yang diautentikasi secara eksternal dengan autentikasi Kerberos di Aurora MySQL seperti yang ditunjukkan di sini:

```
CREATE USER user_name@'host_name' IDENTIFIED WITH 'authentication_kerberos' BY
 'realm_name';
```

- Ganti *user_name* dengan nama pengguna. Pengguna (baik manusia maupun aplikasi) dari domain Anda sekarang dapat terhubung ke klaster DB dari mesin klien yang digabungkan ke domain menggunakan autentikasi Kerberos.
- Ganti *host_name* dengan nama host. Anda dapat menggunakan `%` sebagai wildcard. Anda juga dapat menggunakan alamat IP tertentu untuk nama host.
- Ganti *realm_name* dengan nama realm direktori domain. Nama realm biasanya sama dengan nama domain DNS dalam huruf besar, seperti `CORP.EXAMPLE.COM`. Realm adalah sekelompok sistem yang menggunakan Pusat Distribusi Kunci Kerberos yang sama.

Contoh berikut ini menciptakan pengguna basis data dengan nama `Admin` yang mengautentikasi terhadap Active Directory dengan nama realm `MYSQL.LOCAL`.

```
CREATE USER Admin@'%' IDENTIFIED WITH 'authentication_kerberos' BY 'MYSQL.LOCAL';
```

Memodifikasi login Aurora MySQL yang ada

Anda juga dapat memodifikasi login Aurora MySQL yang ada untuk menggunakan autentikasi Kerberos dengan menggunakan sintaks berikut:

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

Langkah 7: Konfigurasi klien MySQL

Untuk mengonfigurasi klien MySQL, lakukan langkah-langkah berikut:

1. Buat file `krb5.conf` (atau yang setara) untuk menunjuk ke domain.
2. Periksa bahwa lalu lintas dapat mengalir antara host klien dan AWS Directory Service. Gunakan utilitas jaringan seperti Netcat, untuk tugas-tugas berikut:
   - Periksa lalu lintas atas DNS untuk port 53.
   - Memeriksa lalu lintas melalui TCP/UDP untuk port 53 dan untuk Kerberos, yang mencakup port 88 dan 464 untuk AWS Directory Service.
3. Periksa bahwa trafik dapat mengalir antara host klien dan instans basis data melalui port basis data. Misalnya, gunakan `mysql` untuk menghubungkan dan mengakses basis data.

Berikut ini adalah contoh konten `krb5.conf` untuk AWS Managed Microsoft AD.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

Berikut ini adalah contoh konten `krb5.conf` untuk Microsoft Active Directory on-premise.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
  admin_server = onprem.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

Langkah 8: (Opsional) Lakukan konfigurasi perbandingan nama pengguna yang tidak peka huruf besar/kecil

Secara default, huruf besar/kecil nama pengguna basis data MySQL harus sesuai dengan login Active Directory. Namun, Anda sekarang dapat menggunakan perbandingan nama pengguna yang tidak peka huruf besar/kecil dengan plugin `authentication_kerberos`. Untuk melakukannya, Anda mengatur parameter klaster DB `authentication_kerberos_caseins_cmp` ke `true`.

Menggunakan perbandingan nama pengguna yang tidak peka huruf besar/kecil

1. Buat grup parameter klaster DB kustom. Ikuti prosedur di Membuat grup parameter klaster DB.
2. Edit grup parameter baru untuk mengatur nilai `authentication_kerberos_caseins_cmp` ke `true`. Ikuti prosedur di Mengubah parameter dalam grup parameter klaster DB.
3. Kaitkan grup parameter klaster DB dengan klaster DB Aurora MySQL Anda. Ikuti prosedur di Mengaitkan grup parameter klaster DB dengan klaster DB.
4. Boot ulang klaster DB.

Membuat koneksi dengan Aurora MySQL lewat autentikasi Kerberos

Untuk menghindari kesalahan, gunakan klien MySQL dengan versi 8.0.26 atau lebih tinggi pada platform Unix, 8.0.27 atau lebih tinggi di Windows.

Menggunakan login Kerberos Aurora MySQL untuk terhubung ke klaster DB

Untuk membuat koneksi ke Aurora MySQL dengan autentikasi Kerberos, Anda masuk sebagai pengguna basis data yang Anda buat menggunakan instruksi di Langkah 6: Buat pengguna MySQL Aurora yang menggunakan autentikasi Kerberos.

Pada prompt perintah, buat koneksi ke salah satu titik akhir yang terkait dengan klaster DB Aurora MySQL Anda. Saat Anda diminta memasukkan kata sandi, masukkan kata sandi Kerberos yang terkait dengan nama penggunanya.

Saat Anda mengautentikasi dengan Kerberos, sebuah tiket pemberian tiket (TGT) dibuat jika belum ada. Plugin `authentication_kerberos` menggunakan TGT untuk mendapatkan tiket layanan, yang kemudian disajikan ke server basis data Aurora MySQL.

Anda dapat menggunakan klien MySQL untuk terhubung ke Aurora MySQL dengan autentikasi Kerberos menggunakan Windows atau Unix.

Unix

Anda dapat terhubung dengan menggunakan salah satu metode berikut:

- Dapatkan TGT secara manual. Dalam hal ini, Anda tidak perlu memberikan kata sandi ke klien MySQL.
- Berikan kata sandi untuk login Active Directory langsung ke klien MySQL.

Plugin sisi klien didukung pada platform Unix untuk klien MySQL versi 8.0.26 dan yang lebih tinggi.

Terhubung dengan mendapatkan TGT secara manual

1. Pada antarmuka baris perintah, gunakan perintah berikut untuk mendapatkan TGT.

```
kinit user_name
```

2. Gunakan perintah `mysql` berikut ini untuk masuk ke titik akhir instans DB klaster DB Anda.

```
mysql -h DB_instance_endpoint -P 3306 -u user_name -p
```

> **Note**
>
> Autentikasi dapat gagal jika keytab dirotasi pada instans DB. Dalam hal ini, dapatkan TGT baru dengan menjalankan ulang `kinit`.

Terhubung secara langsung

1. Pada antarmuka baris perintah, gunakan perintah `mysql` berikut ini untuk masuk ke titik akhir instans DB klaster DB Anda.

```
mysql -h DB_instance_endpoint -P 3306 -u user_name -p
```

2. Masukkan kata sandi untuk pengguna Direktori Aktif.

Windows

Pada Windows, autentikasi biasanya dilakukan pada waktu login, jadi Anda tidak perlu mendapatkan TGT secara manual untuk terhubung ke klaster DB Aurora MySQL. Huruf besar/kecil pada nama pengguna basis data harus sesuai dengan karakter pengguna di Active Directory. Misalnya, jika pengguna di Active Directory muncul sebagai `Admin`, nama pengguna basis data harus `Admin`.

Plugin sisi klien didukung pada Windows untuk klien MySQL versi 8.0.27 dan yang lebih tinggi.

Terhubung secara langsung

- Pada antarmuka baris perintah, gunakan perintah `mysql` berikut ini untuk masuk ke titik akhir instans DB klaster DB Anda.

```
mysql -h DB_instance_endpoint -P 3306 -u user_name
```

Autentikasi Kerberos dengan basis data global Aurora

Autentikasi Kerberos untuk Aurora MySQL didukung untuk basis data global Aurora. Untuk mengautentikasi pengguna pada klaster DB sekunder menggunakan Active Directory klaster DB primer, lakukan replikasi Active Directory ke Wilayah AWS sekunder. Anda mengaktifkan autentikasi Kerberos pada klaster sekunder menggunakan ID domain yang sama seperti untuk klaster primer. Replikasi AWS Managed Microsoft AD hanya didukung dengan versi Active Directory Enterprise. Untuk informasi selengkapnya, lihat Replikasi Multi-Wilayah di Panduan Administrasi AWS Directory Service.

Migrasi dari RDS for MySQL ke Aurora MySQL

Setelah Anda bermigrasi dari RDS for MySQL dengan autentikasi Kerberos diaktifkan ke Aurora MySQL, ubah pengguna yang dibuat dengan plugin `auth_pam` untuk menggunakan plugin `authentication_kerberos`. Sebagai contoh:

```
ALTER USER user_name IDENTIFIED WITH 'authentication_kerberos' BY 'realm_name';
```

Mencegah caching tiket

Jika TGT yang valid tidak ada saat aplikasi klien MySQL dimulai, aplikasi dapat memperoleh dan meng-cache TGT. Jika Anda ingin mencegah caching TGT, atur parameter konfigurasi dalam file `/etc/krb5.conf`.

> **Note**
>
> Konfigurasi ini hanya berlaku untuk host klien yang menjalankan Unix, bukan Windows.

Mencegah caching TGT

- Tambahkan bagian `[appdefaults]` ke `/etc/krb5.conf` seperti berikut:

```
[appdefaults]
  mysql = {
```

```
  destroy_tickets = true
}
```

Pencatatan log untuk autentikasi Kerberos

Variabel lingkungan `AUTHENTICATION_KERBEROS_CLIENT_LOG` menetapkan tingkat pencatatan log untuk autentikasi Kerberos. Anda dapat menggunakan log untuk debugging sisi klien.

Nilai yang diizinkan adalah 1-5. Pesan log ditulis ke output kesalahan standar. Tabel berikut ini menjelaskan setiap tingkat pencatatan log.

Tingkat pencatatan log	Deskripsi
1 atau tidak diatur	Tidak ada pencatatan log
2	Pesan kesalahan
3	Pesan kesalahan dan peringatan
4	Pesan kesalahan, peringatan, dan informasi
5	Pesan kesalahan, peringatan, informasi, dan debug

Mengelola klaster DB di dalam domain

Anda dapat menggunakan AWS CLI atau API RDS untuk mengelola klaster DB dan hubungannya dengan Active Directory terkelola Anda. Misalnya, Anda dapat mengaitkan Active Directory untuk autentikasi Kerberos dan melepaskan Active Direktory untuk menonaktifkan autentikasi Kerberos. Anda juga dapat memindahkan klaster DB untuk diautentikasi secara eksternal oleh satu Active Directory ke yang lain.

Misalnya, menggunakan API Amazon RDS, Anda dapat melakukan hal berikut:

- Untuk mencoba ulang pengaktifan autentikasi Kerberos untuk keanggotaan yang gagal, gunakan operasi API `ModifyDBInstance` dan tentukan ID direktori keanggotaan saat ini.
- Untuk memperbarui nama peran IAM untuk keanggotaan, gunakan operasi API `ModifyDBInstance` dan tentukan ID direktori keanggotaan saat ini dan peran IAM baru.

- Untuk menonaktifkan autentikasi Kerberos pada klaster DB, gunakan operasi API `ModifyDBInstance` dan tentukan `none` sebagai parameter domain.
- Untuk memindahkan klaster DB dari satu domain ke domain lain, gunakan operasi API `ModifyDBInstance` dan tentukan pengidentifikasi domain baru sebagai parameter domain.
- Untuk memerinci keanggotaan bagi setiap klaster DB, gunakan operasi API `DescribeDBInstances`.

Memahami keanggotaan domain

Setelah Anda membuat atau mengubah klaster DB, klaster tersebut akan menjadi anggota domain. Anda dapat melihat status keanggotaan domain untuk klaster DB dengan menjalankan perintah CLI describe-db-clusters. Status klaster DB dapat berupa salah satu hal berikut ini:

- `kerberos-enabled` – Klaster DB mengaktifkan autentikasi Kerberos.
- `enabling-kerberos` – AWS sedang dalam proses mengaktifkan autentikasi Kerberos pada klaster DB ini.
- `pending-enable-kerberos` – Mengaktifkan autentikasi Kerberos tertunda pada klaster DB ini.
- `pending-maintenance-enable-kerberos` – AWS akan mencoba mengaktifkan autentikasi Kerberos pada klaster DB selama periode pemeliharaan terjadwal berikutnya.
- `pending-disable-kerberos` – Menonaktifkan autentikasi Kerberos tertunda pada klaster DB ini.
- `pending-maintenance-disable-kerberos` – AWS akan mencoba menonaktifkan autentikasi Kerberos pada klaster DB selama periode pemeliharaan terjadwal berikutnya.
- `enable-kerberos-failed` – Masalah konfigurasi telah mencegah AWS dari mengaktifkan autentikasi Kerberos pada klaster DB. Periksa dan perbaiki konfigurasi Anda sebelum menerbitkan ulang perintah modifikasi klaster DB.
- `disabling-kerberos` – AWS sedang dalam proses menonaktifkan autentikasi Kerberos pada klaster DB ini.

Permintaan untuk mengaktifkan autentikasi Kerberos dapat gagal karena masalah konektivitas jaringan atau peran IAM yang salah. Misalnya, anggaplah Anda membuat klaster DB atau memodifikasi klaster DB yang sudah ada dan upaya untuk mengaktifkan autentikasi Kerberos tersebut gagal. Dalam hal ini, terbitkan ulang perintah modifikasi atau modifikasi klaster DB yang baru dibuat untuk bergabung ke domain.

Memigrasikan data ke klaster DB Amazon Aurora MySQL

Anda memiliki beberapa opsi untuk memigrasikan data dari basis data yang sudah ada ke klaster DB Amazon Aurora MySQL. Opsi migrasi Anda juga bergantung pada basis data asal migrasi Anda dan ukuran data yang Anda migrasikan.

Ada dua jenis migrasi yang berbeda: fisik dan logis. Migrasi fisik berarti salinan fisik file basis data digunakan untuk memigrasikan basis data. Migrasi logis berarti bahwa migrasi dilakukan dengan menerapkan perubahan basis data logis, seperti penyisipan, pembaruan, dan penghapusan.

Migrasi fisik memiliki keuntungan sebagai berikut:

- Migrasi fisik lebih cepat daripada migrasi logis, terutama untuk basis data besar.
- Performa basis data tidak terpengaruh saat cadangan diambil untuk migrasi fisik.
- Migrasi fisik dapat memigrasikan semua hal dalam basis data sumber, termasuk komponen basis data yang kompleks.

Migrasi fisik memiliki batasan sebagai berikut:

- Parameter `innodb_page_size` harus diatur ke nilai default-nya (`16KB`).
- Parameter `innodb_data_file_path` harus dikonfigurasi dengan hanya satu file data yang menggunakan nama file data default `"ibdata1:12M:autoextend"`. Basis data yang berisi dua file data, atau memiliki file data dengan nama yang berbeda, tidak dapat dimigrasi menggunakan metode ini.

  Berikut ini adalah contoh nama file yang tidak diizinkan: `"innodb_data_file_path=ibdata1:50M; ibdata2:50M:autoextend"` dan `"innodb_data_file_path=ibdata01:50M:autoextend"`.
- Parameter `innodb_log_files_in_group` harus diatur ke nilai default-nya (`2`).

Migrasi logis memiliki keuntungan sebagai berikut:

- Anda dapat memigrasikan subset basis data, seperti tabel atau bagian tertentu dari tabel.
- Data dapat dimigrasikan terlepas dari struktur penyimpanan fisik.

Migrasi logis memiliki batasan sebagai berikut:

- Migrasi logis biasanya lebih lambat daripada migrasi fisik.
- Komponen basis data yang kompleks dapat memperlambat proses migrasi logis. Dalam beberapa kasus, komponen basis data kompleks bahkan dapat memblokir migrasi logis.

Tabel berikut menjelaskan opsi dan jenis migrasi untuk setiap opsi.

Migrasi dari	Jenis migrasi	Solusi
Instans DB RDS for MySQL	Fisik	Anda dapat bermigrasi dari instans DB RDS for MySQL dengan terlebih dahulu membuat replika baca Aurora MySQL dari instans DB MySQL. Saat lag replika antara instans DB MySQL dan replika baca Aurora MySQL adalah 0, Anda dapat mengarahkan aplikasi klien Anda untuk membaca dari replika baca Aurora, kemudian menghenti kan replikasi untuk menjadikan replika baca Aurora MySQL sebagai klaster DB Aurora MySQL mandiri untuk membaca dan menulis. Untuk detailnya, lihat Memigrasikan data dari instans DB RDS for MySQL ke klaster DB Amazon Aurora MySQL menggunakan replika baca Aurora.
Snapshot DB RDS for MySQL	Fisik	Anda dapat memigrasikan data secara langsung dari snapshot DB RDS for MySQL ke klaster DB Amazon Aurora MySQL. Untuk detailnya, lihat Memigrasikan snapshot RDS for MySQL ke Aurora.
Basis data MySQL eksternal di luar Amazon RDS	Logis	Anda dapat membuat dump data menggunakan utilitas `mysqldump` , lalu mengimpor data tersebut ke klaster DB Amazon Aurora MySQL yang ada. Untuk detailnya, lihat Migrasi logis dari MySQL ke Amazon Aurora MySQL dengan menggunakan mysqldump.

Migrasi dari	Jenis migrasi	Solusi
		Untuk mengekspor metadata bagi pengguna database selama migrasi dari database MySQL eksternal, Anda juga dapat menggunakan perintah MySQL Shell sebagai gantinya. `mysqldump` Untuk informasi selengkapnya, lihat Instance Dump Utility, Schema Dump Utility, dan Table Dump Utility. **Note** Utilitas mysqlpump tidak digunakan lagi pada MySQL 8.0.34.
Basis data MySQL eksternal di luar Amazon RDS	Fisik	Anda dapat menyalin file cadangan dari basis data Anda ke bucket Amazon Simple Storage Service (Amazon S3), lalu memulihkan klaster DB Amazon Aurora MySQL dari file tersebut. Opsi ini dapat jauh lebih cepat dibanding kan memigrasikan data menggunak an `mysqldump` . Untuk detailnya, lihat Migrasi fisik dari MySQL dengan menggunakan Percona XtraBackup dan Amazon S3.

Migrasi dari	Jenis migrasi	Solusi
Basis data MySQL eksternal di luar Amazon RDS	Logis	Anda dapat menyimpan data dari basis data Anda sebagai file teks dan menyalin file tersebut ke bucket Amazon S3. Kemudian, Anda dapat memuat data tersebut ke dalam klaster DB Aurora MySQL yang sudah ada menggunakan perintah `LOAD DATA FROM S3` MySQL. Untuk informasi selengkapnya, lihat Memuat data ke klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3.
Basis data yang tidak kompatibel dengan MySQL	Logis	Anda dapat menggunakan AWS Database Migration Service (AWS DMS) untuk memigrasikan data dari database yang tidak kompatibel dengan MySQL. Untuk informasi selengkapnya AWS DMS, lihat Apa itu layanan migrasi AWS database?

Note

Jika Anda memigrasikan basis data eksternal MySQL ke Amazon RDS, opsi migrasi yang dijelaskan dalam tabel hanya didukung jika basis data Anda mendukung ruang tabel InnoDB atau MyISAM.

Jika basis data MySQL yang Anda migrasikan ke Aurora MySQL menggunakan `memcached`, hapus `memcached` sebelum memigrasikannya.

Anda tidak dapat bermigrasi ke MySQL versi 3.05 dan yang lebih tinggi dari beberapa versi MySQL 8.0 yang lebih lama, termasuk 8.0.11, 8.0.13, dan 8.0.15. Kami menyarankan agar Anda meningkatkan ke MySQL 8.0.28 sebelum migrasi.

Memigrasikan data dari basis data MySQL eksternal ke klaster DB Amazon Aurora MySQL

Jika basis data Anda mendukung ruang tabel InnoDB atau MyISAM, Anda memiliki opsi ini untuk memigrasikan data Anda ke klaster DB Amazon Aurora MySQL:

- Anda dapat membuat dump data menggunakan utilitas `mysqldump`, lalu mengimpor data tersebut ke klaster DB Amazon Aurora MySQL yang ada. Untuk informasi selengkapnya, lihat Migrasi logis dari MySQL ke Amazon Aurora MySQL dengan menggunakan mysqldump.
- Anda dapat menyalin file cadangan penuh dan cadangan inkremental dari basis data Anda ke bucket Amazon S3, lalu memulihkan klaster DB Amazon Aurora MySQL dari file tersebut. Opsi ini dapat jauh lebih cepat dibandingkan memigrasikan data menggunakan `mysqldump`. Untuk informasi selengkapnya, lihat Migrasi fisik dari MySQL dengan menggunakan Percona XtraBackup dan Amazon S3.

Topik

- Migrasi fisik dari MySQL dengan menggunakan Percona XtraBackup dan Amazon S3
- Migrasi logis dari MySQL ke Amazon Aurora MySQL dengan menggunakan mysqldump

Migrasi fisik dari MySQL dengan menggunakan Percona XtraBackup dan Amazon S3

Anda dapat menyalin file cadangan penuh dan inkremental dari basis data sumber MySQL versi 5.7 atau 8.0 ke bucket Amazon S3. Kemudian, Anda dapat memulihkan ke klaster DB Amazon Aurora MySQL dengan versi mesin DB mayor yang sama dari file-file tersebut.

Opsi ini bisa jadi jauh lebih cepat dibandingkan memigrasi data menggunakan `mysqldump` karena penggunaan `mysqldump` akan mengulangi semua perintah untuk membuat kembali skema dan data dari basis data sumber di klaster DB Aurora MySQL baru Anda. Dengan menyalin file data MySQL sumber Anda, Aurora MySQL dapat segera menggunakan file tersebut sebagai data untuk klaster DB Aurora MySQL.

Anda juga dapat meminimalkan waktu henti dengan menggunakan replikasi log biner selama proses migrasi. Jika Anda menggunakan replikasi log biner, basis data MySQL eksternal tetap terbuka untuk transaksi saat data sedang dimigrasi ke klaster DB Aurora MySQL. Setelah klaster DB Aurora MySQL dibuat, gunakan replikasi log biner untuk menyinkronkan klaster DB Aurora MySQL dengan transaksi yang terjadi setelah pencadangan. Saat klaster DB Aurora MySQL sinkron dengan basis data MySQL, selesaikan migrasi dengan sepenuhnya beralih ke klaster DB Aurora MySQL untuk

transaksi baru. Untuk informasi selengkapnya, lihat Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL menggunakan replikasi.

Daftar Isi

- Pertimbangan dan batasan
- Sebelum Anda memulai
  - Menginstal Percona XtraBackup
  - Izin yang diperlukan
  - Membuat peran layanan IAM
- Mencadangkan file untuk dipulihkan sebagai klaster DB Amazon Aurora MySQL
  - Membuat cadangan penuh dengan Percona XtraBackup
  - Menggunakan cadangan inkremental dengan Percona XtraBackup
  - Pertimbangan cadangan
- Memulihkan klaster DB Amazon Aurora MySQL dari bucket Amazon S3
- Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL menggunakan replikasi
  - Mengonfigurasi basis data MySQL eksternal dan klaster DB Aurora MySQL Anda untuk replikasi terenkripsi
  - Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL eksternal
- Mengurangi waktu migrasi fisik ke Amazon Aurora MySQL
  - Jenis tabel yang tidak didukung
  - Akun pengguna dengan hak akses yang tidak didukung
  - Hak akses dinamis di Aurora MySQL versi 3
  - Objek tersimpan dengan 'rdsadmin'@'localhost' sebagai pendefinisi

Pertimbangan dan batasan

Batasan dan pertimbangan berikut berlaku untuk memulihkan klaster DB Amazon Aurora MySQL dari bucket Amazon S3:

- Anda dapat memigrasikan data Anda hanya ke klaster DB baru, bukan klaster DB yang sudah ada.
- Anda harus menggunakan Percona XtraBackup untuk mencadangkan data Anda ke S3. Untuk informasi selengkapnya, lihat Menginstal Percona XtraBackup.

- Bucket Amazon S3 dan klaster DB Aurora MySQL harus berada di Wilayah AWS yang sama.
- Anda tidak dapat memulihkan dari hal berikut:
  - Ekspor snapshot klaster DB ke Amazon S3. Anda juga tidak dapat memigrasikan data dari ekspor snapshot klaster DB ke bucket S3 Anda.
  - Basis data sumber terenkripsi, tetapi Anda dapat mengenkripsi data yang sedang dimigrasikan. Anda juga dapat membiarkan data tidak terenkripsi selama proses migrasi.
  - Basis data MySQL 5.5 atau 5.6
- Anda tidak dapat memulihkan ke klaster DB Aurora Serverless.
- Migrasi mundur tidak didukung untuk versi mayor atau versi minor. Misalnya, Anda tidak dapat bermigrasi dari MySQL versi 8.0 ke Aurora MySQL versi 2 (kompatibel dengan MySQL 5.7), dan Anda tidak dapat bermigrasi dari MySQL versi 8.0.32 ke Aurora MySQL versi 3.03, yang kompatibel dengan MySQL komunitas versi 8.0.26.
- Anda tidak dapat bermigrasi ke MySQL versi 3.05 dan yang lebih tinggi dari beberapa versi MySQL 8.0 yang lebih lama, termasuk 8.0.11, 8.0.13, dan 8.0.15. Kami menyarankan agar Anda meng-upgrade ke MySQL 8.0.28 sebelum migrasi.
- Pengimporan dari Amazon S3 tidak didukung di kelas instans DB db.t2.micro. Namun, Anda dapat memulihkan ke kelas instans DB yang berbeda, dan mengubah kelas instans DB di lain waktu. Untuk informasi selengkapnya tentang kelas instans DB, lihat Kelas instans DB Aurora.
- Amazon S3 membatasi ukuran file yang diunggah ke bucket S3 hingga 5 TB. Jika sebuah file cadangan melebihi 5 TB, maka Anda harus membagi file cadangan ke dalam beberapa file yang lebih kecil.
- Amazon RDS membatasi jumlah file yang dapat diunggah ke sebuah bucket S3 hingga 1 juta. Jika data cadangan untuk basis data Anda, termasuk semua cadangan penuh dan inkremental, melebihi 1 juta file, gunakan sebuah file Gzip (.gz), tar (.tar.gz), atau Percona xbstream (.xbstream) untuk menyimpan file cadangan penuh dan inkremental dalam bucket Amazon S3. Percona XtraBackup 8.0 hanya mendukung Percona xbstream untuk kompresi.
- Untuk menyediakan layanan manajemen untuk setiap klaster DB, pengguna `rdsadmin` dibuat saat klaster DB dibuat. Karena ini adalah pengguna yang dicadangkan di RDS, batasan berikut berlaku:
  - Fungsi, prosedur, tampilan, peristiwa, dan pemicu dengan pendefinisi `'rdsadmin'@'localhost'` tidak diimpor. Untuk informasi selengkapnya, lihat Objek tersimpan dengan 'rdsadmin'@'localhost' sebagai pendefinisi dan Hak akses pengguna master Amazon Aurora MySQL.

  - Ketika klaster DB Aurora MySQL dibuat, pengguna master akan dibuat dengan hak akses paling tinggi yang didukung. Saat memulihkan dari cadangan, hak akses yang tidak didukung yang diberikan kepada pengguna yang diimpor akan dihapus secara otomatis selama impor.

    Untuk mengidentifikasi pengguna yang mungkin terpengaruh oleh hal ini, lihat Akun pengguna dengan hak akses yang tidak didukung. Untuk informasi selengkapnya tentang hak akses yang didukung di Aurora MySQL, lihat Model hak akses berbasis peran.

- Untuk Aurora MySQL versi 3, hak akses dinamis tidak diimpor. Hak akses dinamis yang didukung Aurora dapat diimpor setelah migrasi. Untuk informasi selengkapnya, lihat Hak akses dinamis di Aurora MySQL versi 3.
- Tabel yang dibuat pengguna dalam skema `mysql` tidak dimigrasikan.
- Parameter `innodb_data_file_path` harus dikonfigurasi dengan hanya satu file data yang menggunakan nama file data default `ibdata1:12M:autoextend`. Basis data dengan dua file data, atau dengan file data yang memiliki nama yang berbeda, tidak dapat dimigrasikan menggunakan metode ini.

  Berikut ini adalah contoh nama file yang tidak diizinkan: `innodb_data_file_path=ibdata1:50M`, `ibdata2:50M:autoextend` dan `innodb_data_file_path=ibdata01:50M:autoextend`.
- Anda tidak dapat bermigrasi dari basis data sumber yang memiliki tabel di luar direktori data MySQL default.
- Ukuran maksimum yang didukung untuk cadangan yang tidak terkompresi menggunakan metode ini saat ini terbatas pada 64 TiB. Untuk cadangan terkompresi, batas ini lebih rendah untuk memperhitungkan persyaratan ruang yang tidak terkompresi. Dalam kasus seperti itu, ukuran cadangan maksimum yang didukung adalah (`64 TiB – compressed backup size`).
- Aurora MySQL tidak mendukung pengimporan komponen dan plugin MySQL serta komponen dan plugin eksternal lainnya.
- Aurora MySQL tidak memulihkan semuanya dari basis data Anda. Kami menyarankan agar Anda menyimpan skema dan nilai basis data untuk item berikut dari basis data MySQL sumber Anda, lalu menambahkannya ke klaster DB Aurora MySQL yang dipulihkan setelah dibuat:
  - Akun pengguna
  - Fungsi
  - Prosedur tersimpan

- Informasi zona waktu. Informasi zona waktu dimuat dari sistem operasi lokal klaster DB Aurora MySQL Anda. Untuk informasi selengkapnya, lihat Zona waktu lokal untuk klaster DB Amazon Aurora.

Sebelum Anda memulai

Sebelum Anda dapat menyalin data ke bucket Amazon S3 dan memulihkan klaster DB dari file tersebut, Anda harus melakukan hal berikut:

- Instal Percona XtraBackup di server lokal Anda.
- Izinkan Aurora MySQL untuk mengakses bucket Amazon S3 atas nama Anda.

Menginstal Percona XtraBackup

Amazon Aurora dapat memulihkan klaster DB dari file yang dibuat menggunakan Percona XtraBackup. Anda dapat menginstal Percona XtraBackup dari Unduhan Perangkat Lunak – Percona.

Untuk migrasi MySQL 5.7, gunakan Percona XtraBackup 2.4.

Untuk migrasi MySQL 8.0, gunakan Percona XtraBackup 8.0. Pastikan bahwa versi Percona XtraBackup kompatibel dengan versi mesin basis data sumber Anda.

Izin yang diperlukan

Untuk memigrasikan data MySQL Anda ke klaster DB Amazon Aurora MySQL, beberapa izin diperlukan:

- Pengguna yang meminta agar Aurora membuat klaster baru dari bucket Amazon S3 harus memiliki izin untuk menampilkan daftar bucket untuk akun AWS Anda. Anda memberikan izin ini kepada pengguna menggunakan kebijakan AWS Identity and Access Management (IAM).
- Aurora memerlukan izin untuk bertindak atas nama Anda untuk mengakses Amazon S3 bucket tempat Anda menyimpan file yang digunakan untuk membuat klaster DB Amazon Aurora MySQL Anda. Anda memberikan izin yang diperlukan Aurora menggunakan peran layanan IAM.
- Pengguna yang mengajukan permintaan juga harus memiliki izin untuk menampilkan daftar peran IAM untuk akun AWS Anda.
- Jika pengguna yang mengajukan permintaan tersebut bertujuan untuk membuat peran layanan IAM atau meminta Aurora membuat peran layanan IAM (dengan menggunakan konsol), pengguna tersebut harus memiliki izin untuk membuat peran IAM bagi akun AWS Anda.

- Jika Anda berencana untuk mengenkripsi data selama proses migrasi, perbarui kebijakan IAM bagi pengguna yang akan melakukan migrasi untuk memberikan akses RDS ke AWS KMS keys yang digunakan untuk mengenkripsi cadangan. Untuk petunjuk, lihat Membuat kebijakan IAM untuk mengakses sumber daya AWS KMS.

Misalnya, kebijakan IAM berikut memberikan izin minimum yang diperlukan pengguna untuk menggunakan konsol untuk menampilkan daftar peran IAM, membuat peran IAM, menampilkan daftar bucket Amazon S3 untuk akun Anda, dan menampilkan daftar kunci KMS.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:ListRoles",
                "iam:CreateRole",
                "iam:CreatePolicy",
                "iam:AttachRolePolicy",
                "s3:ListBucket",
                "kms:ListKeys"
            ],
            "Resource": "*"
        }
    ]
}
```

Selain itu, bagi pengguna yang ingin mengaitkan peran IAM dengan bucket Amazon S3, pengguna IAM harus memiliki izin `iam:PassRole` untuk peran IAM tersebut. Izin ini memungkinkan administrator membatasi peran IAM mana yang dapat dikaitkan dengan bucket Amazon S3.

Misalnya, kebijakan IAM berikut mengizinkan pengguna mengaitkan peran bernama `S3Access` dengan bucket Amazon S3.

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"AllowS3AccessRole",
            "Effect":"Allow",
            "Action":"iam:PassRole",
```

```
            "Resource":"arn:aws:iam::123456789012:role/S3Access"
        }
    ]
}
```

Untuk informasi selengkapnya tentang izin pengguna IAM, lihat Mengelola akses menggunakan kebijakan.

Membuat peran layanan IAM

Anda dapat meminta AWS Management Console membuatkan peran untuk Anda dengan memilih opsi Buat Peran Baru (ditampilkan nanti dalam topik ini). Jika Anda memilih opsi ini dan menentukan nama untuk peran baru, Aurora akan membuat peran layanan IAM yang diperlukan Aurora untuk mengakses bucket Amazon S3 Anda dengan nama yang Anda berikan.

Sebagai alternatif, Anda dapat membuat peran secara manual menggunakan prosedur berikut.

Untuk membuat peran IAM bagi Aurora untuk mengakses Amazon S3

1. Selesaikan langkah-langkah dalam Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3.
2. Selesaikan langkah-langkah dalam Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.
3. Selesaikan langkah-langkah dalam Mengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL.

Mencadangkan file untuk dipulihkan sebagai klaster DB Amazon Aurora MySQL

Anda dapat membuat cadangan penuh dari file basis data MySQL menggunakan Percona XtraBackup dan mengunggah file cadangan ke bucket Amazon S3. Alternatifnya, jika Anda sudah menggunakan Percona XtraBackup untuk mencadangkan file basis data MySQL, Anda dapat mengunggah direktori dan file cadangan penuh dan inkremental yang ada ke bucket Amazon S3.

Topik

- Membuat cadangan penuh dengan Percona XtraBackup
- Menggunakan cadangan inkremental dengan Percona XtraBackup
- Pertimbangan cadangan

Membuat cadangan penuh dengan Percona XtraBackup

Untuk membuat cadangan penuh file basis data MySQL Anda yang dapat dipulihkan dari Amazon S3 untuk membuat klaster DB Amazon Aurora MySQL, gunakan utilitas Percona XtraBackup (`xtrabackup`) untuk mencadangkan basis data Anda.

Misalnya, perintah berikut akan membuat cadangan dari basis data MySQL dan menyimpan file-nya dalam folder `/on-premises/s3-restore/backup`.

```
xtrabackup --backup --user=<myuser> --password=<password> --target-dir=</on-premises/
s3-restore/backup>
```

Jika Anda ingin mengompresi cadangan Anda ke dalam satu file (yang dapat dibagi, jika perlu), Anda dapat menggunakan opsi `--stream` untuk menyimpan cadangan Anda dalam salah satu format berikut:

- Gzip (.gz)
- tar (.tar)
- Percona xbstream (.xbstream)

Perintah berikut membuat cadangan basis data MySQL Anda yang dibagi menjadi beberapa file Gzip.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | gzip - | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar.gz
```

Perintah berikut membuat cadangan basis data MySQL Anda yang dibagi menjadi beberapa file tar.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=tar \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.tar
```

Perintah berikut membuat cadangan basis data MySQL Anda yang dibagi menjadi beberapa file xbstream.

```
xtrabackup --backup --user=<myuser> --password=<password> --stream=xbstream \
   --target-dir=</on-premises/s3-restore/backup> | split -d --bytes=500MB \
   - </on-premises/s3-restore/backup/backup>.xbstream
```

> **Note**
>
> Jika Anda melihat kesalahan berikut, itu mungkin disebabkan oleh pencampuran format file dalam perintah Anda:
>
> ```
> ERROR:/bin/tar: This does not look like a tar archive
> ```

Setelah Anda mencadangkan basis data MySQL menggunakan utilitas Percona XtraBackup, Anda dapat menyalin direktori dan file cadangan ke bucket Amazon S3.

Untuk informasi tentang membuat dan mengunggah file ke bucket Amazon S3, lihat Memulai Amazon Simple Storage Service dalam Panduan Memulai Amazon S3.

Menggunakan cadangan inkremental dengan Percona XtraBackup

Amazon Aurora MySQL mendukung cadangan penuh dan inkremental yang dibuat menggunakan Percona XtraBackup. Jika Anda sudah menggunakan Percona XtraBackup untuk melakukan cadangan penuh dan inkremental dari file basis data MySQL Anda, Anda tidak perlu membuat cadangan penuh dan mengunggah file cadangan ke Amazon S3. Sebagai gantinya, Anda dapat menghemat cukup banyak waktu dengan menyalin direktori dan file cadangan yang ada untuk cadangan penuh dan inkremental ke bucket Amazon S3. Untuk informasi selengkapnya, lihat Buat cadangan inkremental di situs web Percona.

Saat menyalin file cadangan penuh dan inkremental ke bucket Amazon S3, Anda harus menyalin konten direktori dasar secara berulang. Konten tersebut termasuk cadangan lengkap serta semua direktori dan file cadangan inkremental. Salinan ini harus mempertahankan struktur direktori di bucket Amazon S3. Aurora melakukan iterasi ke semua file dan direktori. Aurora menggunakan file `xtrabackup-checkpoints` yang disertakan dengan setiap cadangan inkremental untuk mengidentifikasi direktori dasar dan untuk mengurutkan cadangan inkremental berdasarkan rentang nomor urutan log (LSN).

Untuk informasi tentang membuat dan mengunggah file ke bucket Amazon S3, lihat Memulai Amazon Simple Storage Service dalam Panduan Memulai Amazon S3.

Pertimbangan cadangan

Aurora tidak mendukung cadangan sebagian yang dibuat menggunakan Percona XtraBackup. Anda tidak dapat menggunakan opsi berikut untuk membuat cadangan sebagian saat mencadangkan file

sumber untuk basis data Anda: `--tables`, `--tables-exclude`, `--tables-file`, `--databases`, `--databases-exclude`, atau `--databases-file`.

Untuk informasi selengkapnya tentang mencadangkan basis data Anda dengan Percona XtraBackup, lihat Percona XtraBackup - Documentation dan Work with binary logs di situs web Percona.

Aurora mendukung cadangan inkremental yang dibuat menggunakan Percona XtraBackup. Untuk informasi selengkapnya, lihat Buat cadangan inkremental di situs web Percona.

Aurora menggunakan file cadangan Anda berdasarkan nama file. Pastikan untuk menamai file cadangan Anda dengan ekstensi file yang sesuai berdasarkan format file—misalnya, `.xbstream` untuk file yang disimpan menggunakan format Percona xbstream.

Aurora menggunakan file cadangan Anda dalam urutan abjad dan juga dalam urutan angka alami. Selalu gunakan opsi `split` saat Anda mengeluarkan perintah `xtrabackup` untuk memastikan bahwa file cadangan Anda ditulis dan diberi nama sesuai urutan yang benar.

Amazon S3 membatasi ukuran file yang diunggah ke bucket Amazon S3 hingga 5 TB. Jika data cadangan untuk basis data Anda melebihi 5 TB, gunakan perintah `split` untuk membagi file cadangan menjadi beberapa file dengan ukuran masing-masing kurang dari 5 TB.

Aurora membatasi jumlah file sumber yang diunggah ke bucket Amazon S3 hingga 1 juta file. Dalam beberapa kasus, data cadangan untuk basis data Anda, termasuk semua cadangan penuh dan inkremental, dapat menghasilkan jumlah file yang besar. Dalam hal ini, gunakan file tarball (.tar.gz) untuk menyimpan file cadangan penuh dan inkremental di bucket Amazon S3.

Saat Anda mengunggah file ke bucket Amazon S3, Anda dapat menggunakan enkripsi di sisi server untuk mengenkripsi data. Anda kemudian dapat memulihkan klaster DB Amazon Aurora MySQL dari file terenkripsi tersebut. Amazon Aurora MySQL dapat memulihkan klaster DB dengan file terenkripsi menggunakan jenis enkripsi di sisi server berikut:

- Enkripsi di sisi server dengan kunci yang dikelola Amazon S3 (SSE-S3) – Setiap objek dienkripsi dengan kunci unik yang menggunakan enkripsi multifaktor yang kuat.
- Enkripsi di sisi server dengan kunci yang dikelola AWS KMS (SSE-KMS) – Serupa dengan SSE-S3, tetapi Anda memiliki opsi untuk membuat dan mengelola kunci enkripsi sendiri, dan juga perbedaan lainnya.

Untuk informasi tentang menggunakan enkripsi di sisi server saat mengunggah file ke bucket Amazon S3, lihat Melindungi data menggunakan enkripsi di sisi server dalam Panduan Developer Amazon S3.

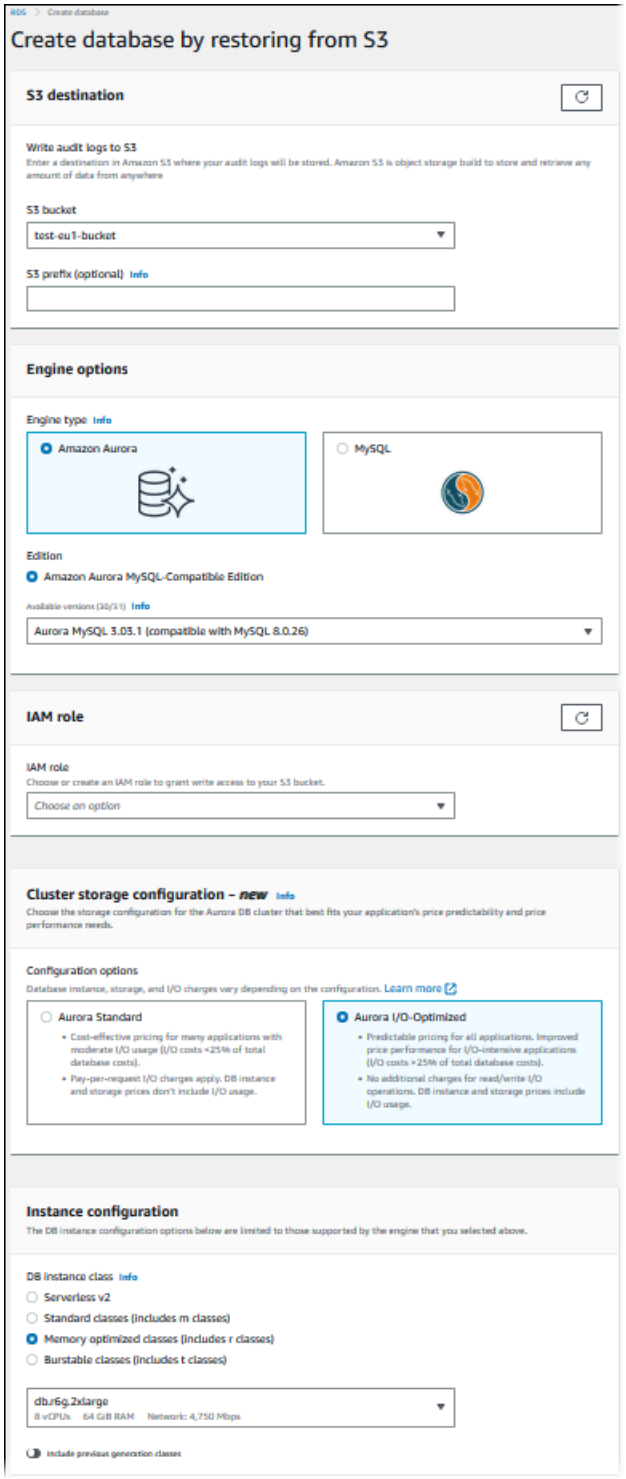
Memulihkan klaster DB Amazon Aurora MySQL dari bucket Amazon S3

Anda dapat memulihkan file cadangan dari bucket Amazon S3 untuk membuat klaster DB Amazon Aurora MySQL baru dengan menggunakan konsol Amazon RDS.

Untuk memulihkan klaster DB Amazon Aurora MySQL dari file di bucket Amazon S3

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas konsol Amazon RDS, pilih Wilayah AWS tempat klaster DB Anda akan dibuat. Pilih Wilayah AWS yang sama dengan bucket Amazon S3 yang berisi cadangan basis data Anda.
3. Di panel navigasi, pilih Basis data, lalu pilih Pulihkan dari S3.
4. Pilih Pulihkan dari S3.

   Halaman Buat basis data dengan memulihkan dari S3 akan muncul.



5. Di bagian Tujuan S3:

   a. Pilih Bucket S3 yang berisi file cadangan.

   b. (Opsional) Untuk Awalan jalur folder S3, masukkan awalan jalur file untuk file yang disimpan di bucket Amazon S3 Anda.

Jika Anda tidak menentukan awalan, RDS akan membuat instans DB Anda menggunakan semua file dan folder di folder root bucket S3. Jika Anda menentukan awalan, RDS akan membuat instans DB Anda menggunakan file dan folder dalam bucket S3 dengan jalur untuk file dimulai dengan awalan yang ditentukan.

Misalnya, anggaplah Anda menyimpan file cadangan di S3 dalam subfolder bernama "cadangan", dan Anda memiliki beberapa set file cadangan, masing-masing di direktorinya sendiri (gzip_backup1, gzip_backup2, dan seterusnya). Dalam hal ini, Anda menentukan awalan cadangan/gzip_backup1 untuk dipulihkan dari file dalam folder gzip_backup1.

6. Di bagian Opsi mesin:

   a. Untuk Tipe mesin, pilih Amazon Aurora.

   b. Untuk Versi, pilih versi mesin Aurora MySQL untuk instans DB Anda yang dipulihkan.

7. Untuk Peran IAM, Anda dapat memilih peran IAM yang sudah ada.

8. (Opsional) Anda juga dapat dibuatkan peran IAM dengan memilih Buat peran baru. Jika demikian:

   a. Masukkan Nama peran IAM.

   b. Pilih opsi untuk Izinkan akses ke kunci KMS:

      - Jika Anda tidak mengenkripsi file cadangan, pilih Tidak.
      - Jika Anda mengenkripsi file cadangan dengan AES-256 (SSE-S3) saat mengunggahnya ke Amazon S3, pilih Tidak. Dalam kasus ini, data didekripsi secara otomatis.
      - Jika Anda mengenkripsi file cadangan dengan enkripsi di sisi server AWS KMS (SSE-KMS) saat mengunggahnya ke Amazon S3, pilih Ya. Selanjutnya, pilih kunci KMS yang benar untuk AWS KMS key.

        AWS Management Console membuat kebijakan IAM yang memungkinkan Aurora mendekripsi data.

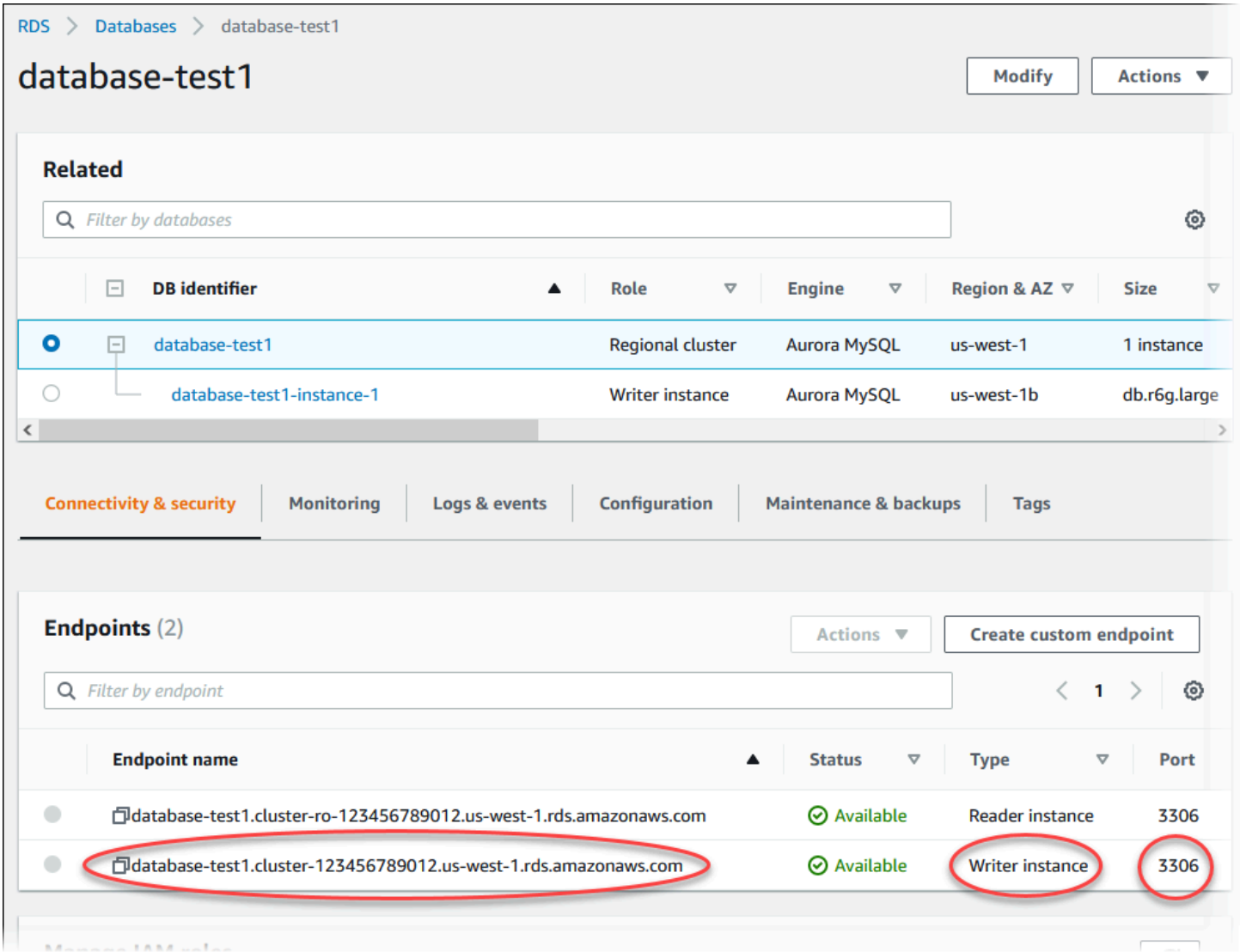
      Untuk informasi selengkapnya, lihat Melindungi data menggunakan enkripsi di sisi server dalam Panduan Developer Amazon S3.

9. Pilih pengaturan untuk klaster DB Anda, seperti konfigurasi penyimpanan klaster DB, kelas instans DB, pengidentifikasi klaster DB, dan kredensial login. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

10. Sesuaikan pengaturan tambahan untuk klaster DB Aurora MySQL Anda berdasarkan kebutuhan.
11. Pilih Buat basis data untuk meluncurkan instans DB Aurora Anda.

Pada konsol Amazon RDS, instans DB baru muncul dalam daftar instans DB. Instans DB memiliki status membuat hingga instans DB dibuat dan siap digunakan. Saat statusnya berubah menjadi tersedia, Anda dapat terhubung ke instans primer untuk klaster DB Anda. Bergantung pada kelas instans DB dan penyimpanan yang dialokasikan, perlu waktu beberapa menit agar instans baru tersedia.

Untuk melihat klaster yang baru dibuat, pilih tampilan Basis data di konsol Amazon RDS lalu pilih klaster DB. Untuk informasi selengkapnya, lihat Melihat klaster DB Amazon Aurora.



Catat port dan titik akhir penulis dari klaster DB. Gunakan titik akhir dan port klaster DB tersebut dalam string koneksi JDBC dan ODBC Anda untuk aplikasi apa pun yang menjalankan operasi tulis atau baca.

Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL menggunakan replikasi

Untuk mempersingkat atau meniadakan waktu henti selama migrasi, Anda dapat mereplikasi transaksi yang dilakukan pada basis data MySQL Anda ke klaster DB Aurora MySQL Anda. Replikasi memungkinkan klaster DB disinkronkan dengan transaksi di basis data MySQL yang terjadi selama migrasi. Ketika klaster DB telah sepenuhnya sinkron, Anda dapat menghentikan replikasi dan menyelesaikan migrasi ke Aurora MySQL.

Topik

- Mengonfigurasi basis data MySQL eksternal dan klaster DB Aurora MySQL Anda untuk replikasi terenkripsi
- Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL eksternal

Mengonfigurasi basis data MySQL eksternal dan klaster DB Aurora MySQL Anda untuk replikasi terenkripsi

Untuk mereplikasi data dengan aman, Anda dapat menggunakan replikasi terenkripsi.

> **Note**
>
> Jika Anda tidak perlu menggunakan replikasi terenkripsi, Anda dapat melewatkan langkah-langkah ini dan melanjutkan ke petunjuk dalam Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL eksternal.

Berikut ini adalah prasyarat untuk menggunakan replikasi terenkripsi:

- Lapisan Soket Aman (SSL) harus diaktifkan di basis data primer MySQL eksternal.
- Kunci klien dan sertifikat klien harus disiapkan untuk klaster DB Aurora MySQL.

Selama replikasi terenkripsi, klaster DB Aurora MySQL berperan sebagai klien untuk server basis data MySQL. Sertifikat dan kunci untuk klien Aurora MySQL ada di dalam file dengan format .pem.

Untuk mengonfigurasi basis data MySQL eksternal dan klaster DB Aurora MySQL Anda untuk replikasi terenkripsi

1. Pastikan bahwa Anda siap untuk replikasi terenkripsi:

   - Jika Anda tidak mengaktifkan SSL di basis data primer MySQL eksternal dan tidak menyiapkan kunci klien dan sertifikat klien, aktifkan SSL di server basis data MySQL dan buat kunci klien dan sertifikat klien yang diperlukan.
   - Jika SSL diaktifkan pada primer eksternal, berikan kunci klien dan sertifikat untuk klaster DB Aurora MySQL. Jika Anda tidak memilikinya, buat kunci dan sertifikat baru untuk klaster DB Aurora MySQL. Untuk menandatangani sertifikat klien, Anda harus memiliki kunci otoritas sertifikat yang Anda gunakan untuk mengonfigurasi SSL di basis data primer MySQL eksternal.

   Untuk informasi selengkapnya, lihat Creating SSL certificates and keys using openssl dalam dokumentasi MySQL.

   Anda memerlukan sertifikat otoritas sertifikat, kunci klien, dan sertifikat klien.
2. Hubungkan ke klaster DB Aurora MySQL sebagai pengguna primer menggunakan SSL.

   Untuk informasi tentang menghubungkan ke klaster DB Aurora MySQL dengan SSL, lihat Menggunakan TLS dengan klaster DB Aurora MySQL.
3. Jalankan prosedur tersimpan mysql.rds_import_binlog_ssl_material untuk mengimpor informasi SSL ke dalam klaster DB Aurora MySQL.

   Untuk parameter `ssl_material_value`, masukkan informasi dari file format .pem untuk klaster DB Aurora MySQL di payload JSON yang benar.

   Contoh berikut mengimpor informasi SSL ke dalam klaster DB Aurora MySQL. Dalam file berformat .pem, kode isi biasanya lebih panjang dari kode isi yang ditunjukkan dalam contoh.

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
```

```
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

Untuk informasi selengkapnya, lihat mysql.rds_import_binlog_ssl_material dan Menggunakan TLS dengan klaster DB Aurora MySQL.

> **Note**
>
> Setelah menjalankan prosedur ini, rahasia disimpan dalam file. Untuk menghapus file nanti, Anda dapat menjalankan prosedur tersimpan mysql.rds_remove_binlog_ssl_material.

Menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL eksternal

Anda dapat menyinkronkan klaster DB Amazon Aurora MySQL dengan basis data MySQL menggunakan replikasi.

Untuk menyinkronkan klaster DB Aurora MySQL Anda dengan basis data MySQL menggunakan replikasi

1. Pastikan file /etc/my.cnf untuk basis data MySQL eksternal memiliki entri yang relevan.

   Jika replikasi terenkripsi tidak diperlukan, pastikan basis data MySQL eksternal dimulai dengan log biner (binlog) diaktifkan dan SSL dinonaktifkan. Berikut ini adalah entri yang relevan dalam file /etc/my.cnf untuk data yang tidak terenkripsi.

   ```
   log-bin=mysql-bin
   server-id=2133421
   innodb_flush_log_at_trx_commit=1
   ```

```
sync_binlog=1
```

Jika replikasi terenkripsi diperlukan, pastikan basis data MySQL eksternal dimulai dengan SSL dan binlog diaktifkan. Entri dalam file /etc/my.cnf mencakup lokasi file .pem untuk server basis data MySQL.

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Setup SSL.
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

Anda dapat memverifikasi bahwa SSL diaktifkan dengan perintah berikut.

```
mysql> show variables like 'have_ssl';
```

Output Anda harus serupa dengan berikut ini.

```
+~-~-~-~-~-~-~-~-~-~-~-~-~-~--+~-~-~-~-~-~--+
| Variable_name | Value |
+~-~-~-~-~-~-~-~-~-~-~-~-~-~--+~-~-~-~-~-~--+
| have_ssl      | YES   |
+~-~-~-~-~-~-~-~-~-~-~-~-~-~--+~-~-~-~-~-~--+
1 row in set (0.00 sec)
```

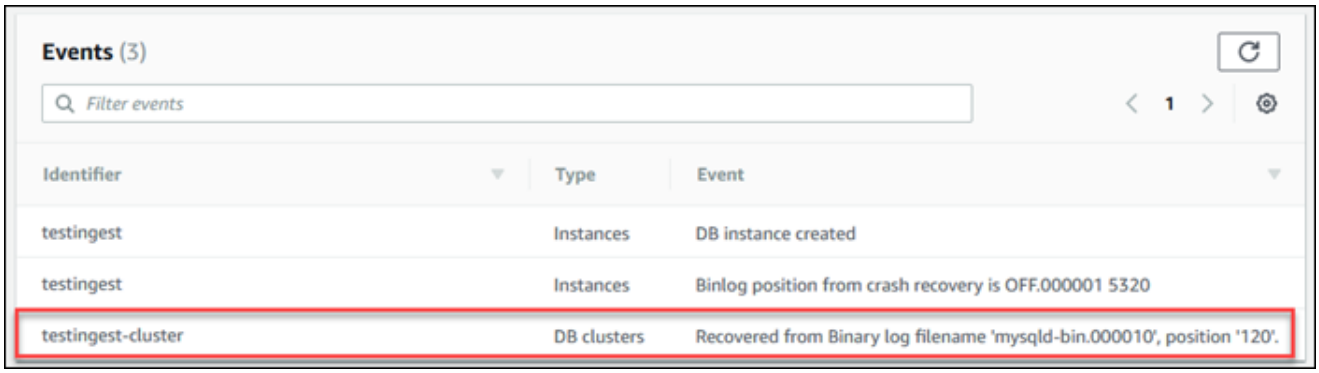
2. Tentukan posisi log biner awal untuk replikasi. Anda menentukan posisi untuk memulai replikasi di langkah selanjutnya.

   Menggunakan AWS Management Console

   a. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

   b. Pada panel navigasi, pilih Peristiwa.

c. Di daftar Peristiwa, catat posisi di peristiwa Dipulihkan dari nama file log biner.



Menggunakan AWS CLI

Anda juga bisa mendapatkan nama dan posisi file binlog dengan menggunakan perintah AWS CLI describe-events. Hal berikut menunjukkan contoh perintah `describe-events`.

```
PROMPT> aws rds describe-events
```

Pada output, identifikasi peristiwa yang menunjukkan posisi binlog.

3. Saat terhubung dengan basis data MySQL eksternal, buat pengguna yang akan digunakan untuk replikasi. Akun ini digunakan hanya untuk replikasi dan harus dibatasi pada domain Anda untuk meningkatkan keamanan. Berikut adalah contohnya.

```
mysql> CREATE USER '<user_name>'@'<domain_name>' IDENTIFIED BY '<password>';
```

Pengguna memerlukan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE`. Berikan hak akses ini kepada pengguna.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO
 '<user_name>'@'<domain_name>';
```

Jika Anda perlu menggunakan replikasi terenkripsi, wajibkan koneksi SSL untuk pengguna replikasi. Misalnya, Anda dapat menggunakan pernyataan berikut untuk mewajibkan koneksi SSL pada akun pengguna *<user_name>*.

```
GRANT USAGE ON *.* TO '<user_name>'@'<domain_name>' REQUIRE SSL;
```

> **Note**
>
> Jika `REQUIRE SSL` tidak disertakan, koneksi replikasi dapat kembali ke koneksi yang tidak terenkripsi tanpa peringatan.

4. Di konsol Amazon RDS, tambahkan alamat IP server yang meng-host basis data MySQL eksternal ke grup keamanan VPC untuk klaster DB Aurora MySQL. Untuk informasi selengkapnya tentang memodifikasi grup keamanan VPC, lihat Grup keamanan untuk VPC Anda dalam Panduan Pengguna Amazon Virtual Private Cloud.

   Anda mungkin juga perlu mengonfigurasi jaringan lokal Anda untuk mengizinkan koneksi dari alamat IP klaster DB Aurora MySQL Anda agar klaster DB ini dapat berkomunikasi dengan instans MySQL eksternal Anda. Untuk menemukan alamat IP klaster DB Aurora MySQL, gunakan perintah `host`.

   ```
   host <db_cluster_endpoint>
   ```

   Nama host adalah nama DNS dari titik akhir klaster DB Aurora MySQL.

5. Aktifkan replikasi log biner dengan menjalankan prosedur tersimpan mysql.rds_reset_external_master (Aurora MySQL versi 2) atau mysql.rds_reset_external_source (Aurora MySQL versi 3). Prosedur tersimpan ini memiliki sintaksis berikut.

   ```
   CALL mysql.rds_set_external_master (
     host_name
     , host_port
     , replication_user_name
     , replication_user_password
     , mysql_binary_log_file_name
     , mysql_binary_log_file_location
     , ssl_encryption
   );

   CALL mysql.rds_set_external_source (
     host_name
     , host_port
     , replication_user_name
   ```

```
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

Untuk informasi tentang parameter, lihat mysql.rds_reset_external_master (Aurora MySQL versi 2) dan mysql.rds_reset_external_source (Aurora MySQL versi 3).

Untuk `mysql_binary_log_file_name` dan `mysql_binary_log_file_location`, gunakan posisi dalam peristiwa Dipulihkan dari nama file log biner yang Anda catat sebelumnya.

Jika data dalam klaster DB Aurora MySQL tidak dienkripsi, parameter `ssl_encryption` harus diatur ke `0`. Jika data dienkripsi, parameter `ssl_encryption` harus diatur ke `1`.

Contoh berikut menjalankan prosedur untuk klaster DB Aurora MySQL yang memiliki data terenkripsi.

```
CALL mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
  1);

CALL mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'password',
  'mysql-bin.000010',
  120,
  1);
```

Prosedur tersimpan ini menetapkan parameter yang digunakan klaster DB Aurora MySQL untuk menghubungkan ke basis data MySQL eksternal dan membaca log binernya. Jika data dienkripsi, prosedur ini juga akan mengunduh sertifikat otoritas sertifikat SSL, sertifikat klien, dan kunci klien ke disk lokal.

6. Mulai replikasi log biner dengan menjalankan prosedur tersimpan mysql.rds_start_replication.

```
CALL mysql.rds_start_replication;
```

7. Pantau seberapa jauh klaster DB Aurora MySQL tertinggal dari basis data primer replikasi MySQL. Untuk melakukannya, hubungkan ke klaster DB Aurora MySQL dan jalankan perintah berikut.

```
Aurora MySQL version 2:
SHOW SLAVE STATUS;

Aurora MySQL version 3:
SHOW REPLICA STATUS;
```

   Dalam output perintah, bidang `Seconds Behind Master` menunjukkan seberapa jauh klaster DB Aurora MySQL tertinggal dari basis data primer MySQL. Ketika nilai ini `0` (nol), klaster DB Aurora MySQL telah sinkron dengan basis data primer, dan Anda dapat melanjutkan ke langkah berikutnya untuk menghentikan replikasi.

8. Hubungkan ke basis data primer replikasi MySQL dan hentikan replikasi. Untuk melakukannya, jalankan prosedur tersimpan mysql.rds_stop_replication.

```
CALL mysql.rds_stop_replication;
```

Mengurangi waktu migrasi fisik ke Amazon Aurora MySQL

Anda dapat membuat modifikasi basis data berikut untuk mempercepat proses migrasi basis data ke Amazon Aurora MySQL.

> **Important**
>
> Pastikan untuk melakukan pembaruan ini pada salinan basis data produksi, bukan pada basis data produksi. Kemudian, Anda dapat mencadangkan salinan ini dan memulihkannya ke klaster DB Aurora MySQL untuk menghindari gangguan layanan apa pun pada basis data produksi Anda.

Jenis tabel yang tidak didukung

Aurora MySQL hanya mendukung mesin InnoDB untuk tabel basis data. Jika Anda memiliki tabel MyISAM di basis data Anda, tabel tersebut harus dikonversi sebelum bermigrasi ke Aurora MySQL. Proses konversi memerlukan ruang tambahan untuk konversi MyISAM ke InnoDB selama prosedur migrasi.

Untuk mengurangi kemungkinan Anda kehabisan ruang atau untuk mempercepat proses migrasi, konversikan semua tabel MyISAM Anda ke tabel InnoDB sebelum memigrasikannya. Ukuran tabel InnoDB yang dihasilkan akan setara dengan ukuran yang diperlukan oleh Aurora MySQL untuk tabel tersebut. Untuk mengonversi tabel MyISAM ke InnoDB, jalankan perintah berikut:

```
ALTER TABLE schema.table_name engine=innodb, algorithm=copy;
```

Aurora MySQL tidak mendukung tabel atau halaman terkompresi, yaitu tabel yang dibuat dengan `ROW_FORMAT=COMPRESSED` atau `COMPRESSION = {"zlib"|"lz4"}`.

Untuk mengurangi kemungkinan Anda kehabisan ruang atau untuk mempercepat proses migrasi, perluas tabel terkompresi Anda dengan mengatur `ROW_FORMAT` ke `DEFAULT`, `COMPACT`, `DYNAMIC`, atau `REDUNDANT`. Untuk halaman terkompresi, atur `COMPRESSION="none"`.

Untuk informasi selengkapnya, lihat InnoDB row formats dan InnoDB table and page compression dalam dokumentasi MySQL.

Anda dapat menggunakan skrip SQL berikut pada instans DB MySQL yang ada untuk menampilkan daftar tabel dalam basis data Anda yang merupakan tabel MyISAM atau tabel terkompresi.

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Aurora MySQL.
-- It must be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
```

```
      as unsigned)
    as major_minor
  ) as T
where major_minor <> 506;


-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
  )
  or
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

Akun pengguna dengan hak akses yang tidak didukung

Akun pengguna dengan hak akses yang tidak didukung oleh Aurora MySQL akan diimpor tanpa hak akses yang tidak didukung. Untuk daftar hak akses yang didukung, lihat Model hak akses berbasis peran.

Anda dapat menjalankan kueri SQL berikut pada basis data sumber Anda untuk menampilkan daftar akun pengguna yang memiliki hak akses yang tidak didukung.

```
SELECT
    user,
    host
FROM
    mysql.user
WHERE
    Shutdown_priv = 'y'
    OR File_priv = 'y'
    OR Super_priv = 'y'
    OR Create_tablespace_priv = 'y';
```

Hak akses dinamis di Aurora MySQL versi 3

Hak akses dinamis tidak akan diimpor. Aurora MySQL versi 3 mendukung hak akses dinamis berikut.

```
'APPLICATION_PASSWORD_ADMIN',
'CONNECTION_ADMIN',
'REPLICATION_APPLIER',
'ROLE_ADMIN',
'SESSION_VARIABLES_ADMIN',
'SET_USER_ID',
'XA_RECOVER_ADMIN'
```

Contoh skrip berikut memberikan hak akses dinamis yang didukung ke akun pengguna di klaster DB Aurora MySQL.

```
-- This script finds the user accounts that have Aurora MySQL supported dynamic
 privileges
-- and grants them to corresponding user accounts in the Aurora MySQL DB cluster.

/home/ec2-user/opt/mysql/8.0.26/bin/mysql -uusername -pxxxxx -P8026 -h127.0.0.1 -BNe
 "SELECT
  CONCAT('GRANT ', GRANTS, ' ON *.* TO ', GRANTEE ,';') AS grant_statement
```

```
  FROM (select GRANTEE, group_concat(privilege_type) AS GRANTS FROM
 information_schema.user_privileges
      WHERE privilege_type IN (
        'APPLICATION_PASSWORD_ADMIN',
        'CONNECTION_ADMIN',
        'REPLICATION_APPLIER',
        'ROLE_ADMIN',
        'SESSION_VARIABLES_ADMIN',
        'SET_USER_ID',
        'XA_RECOVER_ADMIN')
      AND GRANTEE NOT IN (\"'mysql.session'@'localhost'\",
\"'mysql.infoschema'@'localhost'\",\"'mysql.sys'@'localhost'\") GROUP BY GRANTEE)
      AS PRIVGRANTS; " | /home/ec2-user/opt/mysql/8.0.26/bin/mysql -u master_username -
p master_password -h DB_cluster_endpoint
```

Objek tersimpan dengan 'rdsadmin'@'localhost' sebagai pendefinisi

Fungsi, prosedur, tampilan, peristiwa, dan pemicu dengan `'rdsadmin'@'localhost'` sebagai pendefinisi tidak akan diimpor.

Anda dapat menggunakan skrip SQL berikut pada basis data MySQL sumber Anda untuk menampilkan daftar objek tersimpan yang memiliki pendefinisi yang tidak didukung.

```
-- This SQL query lists routines with `rdsadmin`@`localhost` as the definer.

SELECT
    ROUTINE_SCHEMA,
    ROUTINE_NAME
FROM
    information_schema.routines
WHERE
    definer = 'rdsadmin@localhost';

-- This SQL query lists triggers with `rdsadmin`@`localhost` as the definer.

SELECT
    TRIGGER_SCHEMA,
    TRIGGER_NAME,
    DEFINER
FROM
    information_schema.triggers
WHERE
    DEFINER = 'rdsadmin@localhost';
```

```
-- This SQL query lists events with `rdsadmin`@`localhost` as the definer.

SELECT
    EVENT_SCHEMA,
    EVENT_NAME
FROM
    information_schema.events
WHERE
    DEFINER = 'rdsadmin@localhost';

-- This SQL query lists views with `rdsadmin`@`localhost` as the definer.
SELECT
    TABLE_SCHEMA,
    TABLE_NAME
FROM
    information_schema.views
WHERE
    DEFINER = 'rdsadmin@localhost';
```

Migrasi logis dari MySQL ke Amazon Aurora MySQL dengan menggunakan mysqldump

Karena Amazon Aurora MySQL adalah basis data yang kompatibel dengan MySQL, Anda dapat menggunakan utilitas `mysqldump` untuk menyalin data dari basis data MySQL atau MariaDB Anda ke klaster DB Aurora MySQL yang sudah ada.

Untuk pembahasan tentang cara melakukannya dengan basis data MySQL yang sangat besar, lihat Mengimpor data ke instans DB MySQL atau MariaDB dengan waktu henti yang lebih singkat. Untuk basis data MySQL yang memiliki jumlah data lebih kecil, lihat Mengimpor data dari DB MySQL atau MariaDB ke instans DB MySQL atau MariaDB.

Memigrasikan data dari instans DB RDS for MySQL ke klaster DB Amazon Aurora MySQL

Anda dapat memigrasikan (menyalin) data ke klaster DB Amazon Aurora MySQL dari instans DB RDS for MySQL.

Topik

- Memigrasikan snapshot RDS for MySQL ke Aurora
- Memigrasikan data dari instans DB RDS for MySQL ke klaster DB Amazon Aurora MySQL menggunakan replika baca Aurora

> **Note**
>
> Karena Amazon Aurora MySQL kompatibel dengan MySQL, Anda dapat memigrasikan data dari basis data MySQL dengan mengatur replikasi antara basis data MySQL Anda dan klaster DB Amazon Aurora MySQL. Untuk informasi selengkapnya, lihat Replikasi dengan Amazon Aurora.

Memigrasikan snapshot RDS for MySQL ke Aurora

Anda dapat memigrasikan snapshot DB dari instans DB RDS for MySQL untuk membuat klaster DB Aurora MySQL. Klaster DB Aurora MySQL yang baru akan diisi dengan data dari instans DB RDS for MySQL asli. Snapshot DB ini harus dibuat dari instans DB Amazon RDS yang menjalankan versi MySQL yang kompatibel dengan Aurora MySQL.

Anda dapat memigrasikan snapshot DB manual atau otomatis. Setelah klaster DB dibuat, Anda dapat membuat Replika Aurora opsional.

> **Note**
>
> Anda juga dapat memigrasikan instans DB RDS for MySQL ke klaster DB Aurora MySQL dengan membuat replika baca Aurora dari instans DB RDS for MySQL sumber Anda. Untuk informasi selengkapnya, lihat Memigrasikan data dari instans DB RDS for MySQL ke klaster DB Amazon Aurora MySQL menggunakan replika baca Aurora.

Anda tidak dapat bermigrasi ke MySQL versi 3.05 dan yang lebih tinggi dari beberapa versi MySQL 8.0 yang lebih lama, termasuk 8.0.11, 8.0.13, dan 8.0.15. Kami menyarankan agar Anda meng-upgrade ke MySQL 8.0.28 sebelum migrasi.

Langkah-langkah umum yang harus Anda ambil adalah sebagai berikut:

1. Tentukan jumlah ruang yang perlu disediakan untuk klaster DB Aurora MySQL Anda. Untuk informasi selengkapnya, lihat Berapa banyak ruang yang saya butuhkan?.
2. Gunakan konsol untuk membuat snapshot di Wilayah AWS tempat instans Amazon RDS MySQL berada. Untuk informasi tentang membuat snapshot DB, lihat Membuat snapshot DB.
3. Jika snapshot DB tidak berada di Wilayah AWS yang sama dengan klaster DB Anda, gunakan konsol Amazon RDS untuk menyalin snapshot DB ke Wilayah AWS tersebut. Untuk informasi tentang menyalin snapshot DB, lihat Menyalin snapshot DB.
4. Gunakan konsol untuk memigrasikan snapshot DB dan membuat klaster DB Aurora MySQL dengan basis data yang sama dengan instans DB MySQL asli.

⚠ Warning

Amazon RDS membatasi setiap akun AWS ke satu salinan snapshot di setiap Wilayah AWS pada satu waktu.

Berapa banyak ruang yang saya butuhkan?

Saat Anda memigrasikan snapshot dari instans DB MySQL ke dalam klaster DB Aurora MySQL, Aurora akan menggunakan volume Amazon Elastic Block Store (Amazon EBS) untuk memformat data dari snapshot ini sebelum memigrasikannya. Dalam beberapa kasus, ruang tambahan diperlukan untuk memformat data untuk migrasi.

Tabel yang bukan berupa tabel MyISAM dan tidak dikompresi dapat berukuran hingga 16 TB. Jika Anda memiliki tabel MyISAM, Aurora harus menggunakan ruang tambahan dalam volume untuk mengonversi tabel ini agar kompatibel dengan Aurora MySQL. Jika Anda memiliki tabel yang dikompresi, Aurora harus menggunakan ruang tambahan dalam volume untuk memperluas tabel tersebut sebelum menyimpannya di volume klaster Aurora. Karena persyaratan ruang tambahan ini, Anda harus memastikan bahwa tidak ada tabel MyISAM dan tabel terkompresi yang dimigrasikan dari instans DB MySQL Anda yang berukuran lebih dari 8 TB.

Mengurangi jumlah ruang yang diperlukan untuk memigrasikan data ke Amazon Aurora MySQL

Anda sebaiknya memodifikasi skema basis data Anda sebelum memigrasikannya ke Amazon Aurora. Modifikasi tersebut dapat membantu dalam kasus berikut:

- Anda ingin mempercepat proses migrasi.
- Anda tidak yakin berapa banyak ruang yang perlu disediakan.
- Anda telah mencoba memigrasikan data Anda dan migrasi gagal karena kurangnya ruang terprovisi.

Anda dapat membuat perubahan berikut untuk meningkatkan proses migrasi basis data ke Amazon Aurora.

> **Important**
>
> Pastikan untuk melakukan pembaruan ini pada instans DB baru yang dipulihkan dari snapshot basis data produksi, alih-alih pada instans produksi. Kemudian, Anda dapat memigrasikan data dari snapshot instans DB baru Anda ke dalam klaster DB Aurora untuk menghindari gangguan layanan apa pun pada basis data produksi Anda.

Jenis tabel	Batasan atau pedoman
Tabel MyISAM	Aurora MySQL hanya mendukung tabel InnoDB. Jika Anda memiliki tabel MyISAM di basis data Anda, tabel tersebut harus dikonversi sebelum dimigrasikan ke Aurora MySQL. Proses konversi memerlukan ruang tambahan untuk konversi MyISAM ke InnoDB selama prosedur migrasi. Untuk mengurangi kemungkinan Anda kehabisan ruang atau untuk mempercepat proses migrasi, konversikan semua tabel MyISAM Anda ke tabel InnoDB sebelum memigrasikannya. Ukuran tabel InnoDB yang dihasilkan akan setara dengan ukuran yang diperlukan oleh Aurora MySQL untuk tabel tersebut. Untuk mengonversi tabel MyISAM ke InnoDB, jalankan perintah berikut:

Jenis tabel	Batasan atau pedoman
	`alter table <schema>.<table_name> engine=innodb, algorithm=copy;`
Tabel terkompresi	Aurora MySQL tidak mendukung tabel terkompresi (yaitu, tabel yang dibuat dengan `ROW_FORMAT=COMPRESSED`). Untuk mengurangi kemungkinan Anda kehabisan ruang atau untuk mempercepat proses migrasi, perluas tabel terkompresi Anda dengan mengatur `ROW_FORMAT` ke `DEFAULT`, `COMPACT`, `DYNAMIC`, atau `REDUNDANT`. Untuk informasi selengkapnya, lihat InnoDB row formats dalam dokumentasi MySQL.

Anda dapat menggunakan skrip SQL berikut pada instans DB MySQL yang ada untuk menampilkan daftar tabel dalam basis data Anda yang merupakan tabel MyISAM atau tabel terkompresi.

```
-- This script examines a MySQL database for conditions that block
-- migrating the database into Amazon Aurora.
-- It needs to be run from an account that has read permission for the
-- INFORMATION_SCHEMA database.

-- Verify that this is a supported version of MySQL.

select msg as `==> Checking current version of MySQL.`
from
  (
  select
    'This script should be run on MySQL version 5.6 or higher. ' +
    'Earlier versions are not supported.' as msg,
    cast(substring_index(version(), '.', 1) as unsigned) * 100 +
      cast(substring_index(substring_index(version(), '.', 2), '.', -1)
      as unsigned)
    as major_minor
  ) as T
where major_minor <> 506;


-- List MyISAM and compressed tables. Include the table size.

select concat(TABLE_SCHEMA, '.', TABLE_NAME) as `==> MyISAM or Compressed Tables`,
```

```
round(((data_length + index_length) / 1024 / 1024), 2) "Approx size (MB)"
from INFORMATION_SCHEMA.TABLES
where
  ENGINE <> 'InnoDB'
  and
  (
    -- User tables
    TABLE_SCHEMA not in ('mysql', 'performance_schema',
                         'information_schema')
    or
    -- Non-standard system tables
    (
      TABLE_SCHEMA = 'mysql' and TABLE_NAME not in
        (
          'columns_priv', 'db', 'event', 'func', 'general_log',
          'help_category', 'help_keyword', 'help_relation',
          'help_topic', 'host', 'ndb_binlog_index', 'plugin',
          'proc', 'procs_priv', 'proxies_priv', 'servers', 'slow_log',
          'tables_priv', 'time_zone', 'time_zone_leap_second',
          'time_zone_name', 'time_zone_transition',
          'time_zone_transition_type', 'user'
        )
    )
  )
  or
  (
    -- Compressed tables
       ROW_FORMAT = 'Compressed'
  );
```

Skrip ini menghasilkan output yang mirip dengan output dalam contoh berikut. Contoh ini menunjukkan dua tabel yang harus dikonversi dari MyISAM ke InnoDB. Output-nya juga berisi perkiraan ukuran setiap tabel dalam megabyte (MB).

```
+---------------------------------+------------------+
| ==> MyISAM or Compressed Tables | Approx size (MB) |
+---------------------------------+------------------+
| test.name_table                 |          2102.25 |
| test.my_table                   |            65.25 |
+---------------------------------+------------------+
2 rows in set (0.01 sec)
```

Memigrasikan snapshot DB RDS for MySQL ke klaster DB Aurora MySQL

Anda dapat memigrasikan snapshot DB dari instans DB RDS for MySQL untuk membuat klaster DB Aurora MySQL menggunakan AWS Management Console atau AWS CLI. Klaster DB Aurora MySQL yang baru akan diisi dengan data dari instans DB RDS for MySQL asli. Untuk informasi tentang membuat snapshot DB, lihat Membuat snapshot DB.

Jika DB snapshot tidak berada di Wilayah AWS tempat Anda ingin menempatkan data Anda, salin snapshot DB ini ke Wilayah AWS tersebut. Untuk informasi tentang menyalin snapshot DB, lihat Menyalin snapshot DB.

Konsol

Saat Anda memigrasikan snapshot DB dengan menggunakan AWS Management Console, konsol akan mengambil tindakan yang diperlukan untuk membuat klaster DB dan instans primer.

Anda juga dapat memilih klaster DB Aurora MySQL baru Anda untuk dienkripsi saat diam menggunakan AWS KMS key.

Untuk memigrasikan snapshot DB MySQL dengan menggunakan AWS Management Console

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Mulai migrasi dari instans DB MySQL atau dari snapshot:

   Untuk memulai migrasi dari instans DB:

   1. Di panel navigasi, pilih Basis data, lalu pilih instans DB MySQL.
   2. Untuk Tindakan, pilih Migrasikan snapshot terbaru.

   Untuk memulai migrasi dari snapshot:

   1. Pilih Snapshot.
   2. Di Snapshot, pilih snapshot yang ingin Anda migrasikan ke klaster DB Aurora MySQL.
   3. Pilih Tindakan Snapshot, lalu pilih Migrasikan Snapshot.

   Halaman Migrasikan Basis Data akan muncul.
3. Tetapkan nilai-nilai berikut pada halaman Migrasikan Basis Data:

- Migrasikan ke Mesin DB: Pilih `aurora`.
- Versi Mesin DB: Pilih versi mesin DB untuk klaster DB Aurora MySQL.
- Kelas Instans DB: Pilih kelas instans DB yang memiliki penyimpanan dan kapasitas yang diperlukan untuk basis data Anda, misalnya `db.r3.large`. Volume klaster Aurora secara otomatis bertambah seiring peningkatan jumlah data dalam basis data Anda. Volume klaster Aurora dapat bertambah hingga ukuran maksimum 128 tebibyte (TiB). Jadi, Anda hanya perlu memilih kelas instans DB yang memenuhi persyaratan penyimpanan Anda saat ini. Untuk informasi selengkapnya, lihat Gambaran umum penyimpanan Amazon Aurora.
- Pengidentifikasi Instans DB: Ketikkan nama untuk klaster DB yang unik untuk akun Anda di Wilayah AWS yang Anda pilih. Pengidentifikasi ini digunakan di alamat titik akhir untuk instans di klaster DB Anda. Anda dapat memilih untuk menambahkan beberapa detail ke nama ini, seperti menyertakan Wilayah AWS dan mesin DB yang Anda pilih, misalnya **`aurora-cluster1`**.

  Pengidentifikasi instans DB memiliki batasan berikut:

  - Pengidentifikasi ini harus berisi 1 hingga 63 karakter alfanumerik atau tanda hubung.
  - Karakter pertamanya harus berupa huruf.
  - Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut.
  - Pengidentifikasi ini harus unik untuk semua instans DB per akun AWS, per Wilayah AWS.
- Cloud Privat Virtual (VPC): Jika Anda sudah memiliki VPC, Anda dapat menggunakan VPC tersebut dengan klaster DB Aurora MySQL dengan memilih pengidentifikasi VPC Anda, misalnya `vpc-a464d1c1`. Untuk informasi tentang membuat VPC, lihat Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4).

  Atau, Anda dapat memilih agar Aurora membuat VPC untuk Anda dengan memilih Buat VPC baru.
- Grup subnet: Jika Anda memiliki grup subnet yang ada, Anda dapat menggunakan grup subnet tersebut dengan klaster DB Aurora MySQL Anda dengan memilih pengidentifikasi grup subnet Anda, misalnya `gs-subnet-group1`.

  Jika tidak, Anda dapat memilih agar Aurora membuat grup subnet untuk Anda dengan memilih Buat grup subnet baru.
- Aksesibilitas publik: Pilih Tidak untuk menentukan bahwa instans di klaster DB Anda hanya dapat diakses oleh sumber daya di dalam VPC Anda. Pilih Ya untuk menentukan bahwa

instans dalam klaster DB Anda dapat diakses oleh sumber daya di jaringan publik. Default-nya adalah Ya.

> **Note**
>
> Klaster DB produksi Anda mungkin tidak perlu berada di subnet publik karena hanya server aplikasi Anda yang memerlukan akses ke klaster DB Anda. Jika klaster DB Anda tidak perlu berada di subnet publik, tetapkan Dapat Diakses Publik ke Tidak.

- Ketersediaan Zona: Pilih Zona Ketersediaan guna meng-host instans primer untuk klaster DB Aurora MySQL Anda. Agar Aurora memilih Zona Ketersediaan untuk Anda, pilih Tidak Ada Preferensi.
- Port Basis Data: Ketik port default yang akan digunakan saat menghubungkan ke instans di klaster DB Aurora MySQL. Default-nya adalah `3306`.

> **Note**
>
> Anda mungkin berada di belakang firewall perusahaan yang tidak mengizinkan akses ke port default seperti port default MySQL, 3306. Dalam hal ini, berikan nilai port yang diizinkan oleh firewall perusahaan Anda. Ingat nilai port tersebut nanti saat Anda terhubung ke klaster DB Aurora MySQL.

- Enkripsi: Pilih Aktifkan Enkripsi untuk klaster DB Aurora MySQL baru Anda agar dienkripsi saat diam. Jika Anda memilih Aktifkan Enkripsi, Anda harus memilih kunci KMS sebagai nilai AWS KMS key.

  Jika snapshot DB Anda tidak terenkripsi, tentukan kunci enkripsi agar klaster DB Anda dienkripsi saat diam.

  Jika snapshot DB Anda dienkripsi, tentukan kunci enkripsi agar klaster DB Anda dienkripsi saat diam menggunakan kunci enkripsi yang ditentukan. Anda dapat menentukan kunci enkripsi yang digunakan oleh snapshot DB atau kunci yang berbeda. Anda tidak dapat membuat klaster DB yang tidak terenkripsi dari snapshot DB terenkripsi.
- Peningkatan Versi Minor Otomatis: Pengaturan ini tidak berlaku untuk klaster DB Aurora MySQL.

  Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora MySQL, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL.

4. Pilih Migrasikan untuk memigrasikan snapshot DB Anda.
5. Pilih Instans, lalu pilih ikon panah untuk menampilkan detail klaster DB dan memantau progres migrasi. Di halaman detail, Anda dapat menemukan titik akhir klaster yang digunakan untuk terhubung ke instans primer klaster DB. Untuk informasi selengkapnya tentang menghubungkan ke klaster DB Aurora MySQL, lihat Menghubungkan ke klaster DB Amazon Aurora.

AWS CLI

Anda dapat membuat klaster DB Aurora dari snapshot DB milik instans DB RDS for MySQL menggunakan perintah `restore-db-cluster-from-snapshot` dengan parameter berikut:

- `--db-cluster-identifier` – Nama klaster DB yang akan dibuat.
- `--engine aurora-mysql` – Untuk klaster DB yang kompatibel dengan MySQL 5.7 atau yang kompatibel dengan MySQL 8.0.
- `--kms-key-id` – AWS KMS key yang digunakan untuk mengenkripsi klaster DB secara opsional, tergantung pada apakah snapshot DB Anda dienkripsi atau tidak.
  - Jika snapshot DB Anda tidak terenkripsi, tentukan kunci enkripsi agar klaster DB Anda dienkripsi saat diam. Jika tidak, klaster DB Anda tidak dienkripsi.
  - Jika snapshot DB Anda dienkripsi, tentukan kunci enkripsi agar klaster DB Anda dienkripsi saat diam menggunakan kunci enkripsi yang ditentukan. Jika tidak, klaster DB Anda dienkripsi saat diam menggunakan kunci enkripsi untuk snapshot DB.

  > **Note**
  >
  > Anda tidak dapat membuat klaster DB yang tidak terenkripsi dari snapshot DB terenkripsi.

- `--snapshot-identifier` – Nama Amazon Resource Name (ARN) untuk snapshot DB yang akan dimigrasikan. Untuk informasi selengkapnya tentang ARN Amazon RDS, lihat Amazon Relational Database Service (Amazon RDS).

Saat Anda memigrasikan snapshot DB dengan menggunakan perintah `RestoreDBClusterFromSnapshot`, perintah ini akan membuat klaster DB dan instans primer.

Dalam contoh ini, Anda membuat klaster DB yang kompatibel dengan MySQL 5.7 bernama *`mydbcluster`* dari snapshot DB dengan ARN yang diatur ke *`mydbsnapshotARN`*.

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

Dalam contoh ini, Anda membuat klaster DB yang kompatibel dengan MySQL 5.7 bernama *mydbcluster* dari snapshot DB dengan ARN yang diatur ke *mydbsnapshotARN*.

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mydbcluster \
    --snapshot-identifier mydbsnapshotARN \
    --engine aurora-mysql
```

Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier mydbcluster ^
    --snapshot-identifier mydbsnapshotARN ^
    --engine aurora-mysql
```

Memigrasikan data dari instans DB RDS for MySQL ke klaster DB Amazon Aurora MySQL menggunakan replika baca Aurora

Aurora menggunakan fungsi replikasi biner log mesin DB MySQL untuk membuat jenis khusus klaster DB yang disebut replika baca Aurora untuk instans DB MySQL sumber. Pembaruan yang dibuat ke instans DB RDS for MySQL sumber direplikasi secara asinkron ke replika baca Aurora.

Kami merekomendasikan untuk menggunakan fungsionalitas ini untuk bermigrasi dari instans DB RDS for MySQL ke klaster DB Aurora MySQL dengan membuat replika baca Aurora dari instans DB RDS for Aurora MySQL. Ketika lag replika antara instans DB MySQL dan replika baca Aurora adalah 0, Anda dapat mengarahkan aplikasi klien Anda ke replika baca Aurora lalu menghentikan replikasi untuk menjadikan replika baca Aurora sebagai klaster DB Aurora MySQL mandiri. Lakukan antisipasi sesuai kebutuhan karena migrasi dapat memerlukan waktu yang lama, sekitar beberapa jam per tebibyte (TiB) data.

Untuk daftar wilayah tempat Aurora tersedia, lihat Amazon Aurora dalam Referensi Umum AWS.

Saat Anda membuat replika baca Aurora dari instans DB RDS for MySQL, Amazon RDS membuat snapshot DB dari instans DB RDS for MySQL sumber Anda (privat ke Amazon RDS, dan tidak dikenai biaya). Amazon RDS kemudian memigrasikan data dari snapshot DB ke replika baca Aurora. Setelah data dari snapshot DB dimigrasikan ke klaster DB Aurora MySQL baru, Amazon RDS memulai replikasi antara instans DB MySQL Anda dan klaster DB Aurora MySQL. Jika instans DB MySQL Anda berisi tabel yang menggunakan mesin penyimpanan selain InnoDB, atau yang menggunakan format baris terkompresi, Anda dapat mempercepat proses pembuatan replika baca Aurora dengan mengubah tabel tersebut untuk menggunakan mesin penyimpanan InnoDB dan format baris dinamis sebelum Anda membuat replika baca Aurora Anda. Untuk informasi selengkapnya tentang proses menyalin snapshot DB MySQL ke klaster DB Aurora MySQL, lihat Memigrasikan data dari instans DB RDS for MySQL ke klaster DB Amazon Aurora MySQL.

Anda hanya dapat memiliki satu replika baca Aurora untuk instans DB RDS for MySQL.

> **Note**
>
> Masalah replikasi dapat muncul karena perbedaan fitur antara Amazon Aurora MySQL dan versi mesin basis data MySQL dari instans DB RDS for MySQL Anda yang merupakan replikasi primer. Jika Anda menemukan kesalahan, Anda dapat menemukan bantuan di forum komunitas Amazon RDS atau dengan menghubungi AWS Support.
> Anda tidak dapat membuat replika baca Aurora jika instans DB RDS for MySQL Anda sudah menjadi sumber replika baca lintas Wilayah.

Anda tidak dapat bermigrasi ke Aurora MySQL versi 3.05 dan lebih tinggi dari beberapa versi RDS for MySQL 8.0 yang lebih lama, termasuk 8.0.11, 8.0.13, dan 8.0.15. Kami menyarankan Anda meng-upgrade ke RDS for MySQL versi 8.0.28 sebelum bermigrasi.

Untuk informasi selengkapnya tentang replika baca MySQL, lihat Menggunakan replika baca instans DB MariaDB, MySQL, dan PostgreSQL.

Membuat replika baca Aurora

Anda dapat membuat replika baca Aurora untuk instans DB RDS for MySQL dengan menggunakan konsol, AWS CLI, atau API RDS.

Konsol

Untuk membuat replika baca Aurora dari instans DB RDS for MySQL sumber

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Database.
3. Pilih instans DB MySQL yang ingin Anda gunakan sebagai sumber untuk replika baca Aurora Anda.
4. Untuk Tindakan, pilih Buat replika baca Aurora.
5. Pilih spesifikasi klaster DB yang ingin Anda gunakan untuk replika baca Aurora, seperti yang dijelaskan dalam tabel berikut.

Opsi	Deskripsi
Kelas instans DB	Pilih kelas instans DB yang menentukan persyaratan pemrosesan dan memori untuk instans primer dalam DB klaster. Untuk informasi selengkapnya tentang opsi kelas instans DB, lihat Kelas instans DB Aurora.
Deployment Multi-AZ	Pilih Buat Replika di Zona yang Berbeda untuk membuat replika siaga klaster DB baru di Zona Ketersediaan lain di Wilayah AWS target untuk dukungan failover. Untuk informasi selengkapnya

Opsi	Deskripsi
	tentang beberapa Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan.
Pengidentifikasi instans DB	Ketikkan nama untuk instans primer di klaster DB replika baca Aurora Anda. Pengidentifikasi ini digunakan dalam alamat titik akhir untuk instans primer klaster DB baru. Pengidentifikasi instans DB memiliki batasan berikut: • Pengidentifikasi ini harus berisi 1 hingga 63 karakter alfanumerik atau tanda hubung. • Karakter pertamanya harus berupa huruf. • Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut. • Pengidentifikasi ini harus unik untuk semua instans DB untuk setiap akun AWS dan setiap AWS. Karena klaster DB replika baca Aurora dibuat dari snapshot instans DB sumber, nama pengguna master dan kata sandi master untuk replika baca Aurora sama dengan nama pengguna master dan kata sandi master untuk instans DB sumber.
Cloud Privat Virtual (VPC)	Pilih VPC untuk meng-host klaster DB. Pilih Buat VPC baru agar Aurora membuat VPC untuk Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.
Grup subnet DB	Pilih grup subnet DB yang akan digunakan untuk klaster DB. Pilih Buat grup DB subnet baru agar Aurora membuat grup DB subnet untuk Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.

Opsi	Deskripsi
Aksesibilitas publik	Pilih `Yes` untuk memberikan alamat IP publik pada klaster DB; jika tidak, pilih `No`. Instans dalam klaster DB Anda dapat berupa campuran antara instans DB publik dan privat. Untuk informasi selengkapnya tentang menyembunyikan instans dari akses publik, lihat Menyembunyikan instans DB dalam VPC dari internet.
Zona ketersediaan	Tentukan apakah Anda ingin menentukan Zona Ketersediaan tertentu. Untuk informasi selengkapnya tentang Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan.
Grup keamanan VPC (firewall)	Pilih Buat grup keamanan VPC baru agar Aurora membuat grup keamanan VPC untuk Anda. Klik Pilih grup keamanan VPC yang ada untuk menentukan satu atau beberapa grup keamanan VPC untuk mengamankan akses jaringan ke klaster DB. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.
Port basis data	Tentukan port untuk aplikasi dan utilitas yang akan digunakan untuk mengakses basis data. Klaster DB Aurora MySQL ditetapkan secara default ke port MySQL default, 3306. Firewall di beberapa perusahaan memblokir koneksi ke port ini. Jika firewall perusahaan Anda memblokir port default ini, pilih port lain untuk klaster DB baru.
Grup parameter DB	Pilih grup parameter DB untuk klaster DB Aurora MySQL. Aurora memiliki grup parameter DB default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter DB Anda sendiri. Untuk informasi selengkapnya tentang grup parameter DB, lihat Bekerja dengan grup parameter.

Opsi	Deskripsi
Grup parameter klaster DB	Pilih grup parameter klaster DB untuk klaster DB Aurora MySQL. Aurora memiliki grup parameter klaster DB default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter klaster DB Anda sendiri. Untuk informasi selengkapnya tentang grup parameter klaster DB, lihat Bekerja dengan grup parameter.
Enkripsi	Pilih Nonaktifkan enkripsi jika Anda tidak ingin klaster DB Aurora baru Anda dienkripsi. Pilih Aktifkan Enkripsi untuk klaster DB Aurora baru Anda agar dienkripsi saat diam. Jika Anda memilih Aktifkan enkripsi, Anda harus memilih kunci KMS sebagai nilai AWS KMS key. Jika instans DB MySQL Anda tidak dienkripsi, tentukan kunci enkripsi agar klaster DB Anda dienkripsi saat diam. Jika instans DB MySQL Anda dienkripsi, tentukan kunci enkripsi agar klaster DB Anda dienkripsi saat diam menggunakan kunci enkripsi yang ditentuka n. Anda dapat menentukan kunci enkripsi yang digunakan oleh instans DB MySQL atau kunci yang berbeda. Anda tidak dapat membuat klaster DB yang tidak terenkripsi dari instans DB MySQL terenkripsi.
Prioritas	Pilih prioritas failover untuk klaster DB. Jika Anda tidak memilih nilai, nilai default-nya adalah tier-1. Prioritas ini akan menentukan urutan promosi Aurora Replika saat melakukan pemulihan dari kegagalan instans primer. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora.

Opsi	Deskripsi
Periode retensi cadangan	Pilih durasi waktu, dari 1 hingga 35 hari, saat Aurora harus mempertahankan salinan cadangan basis data. Salinan cadangan dapat digunakan untuk point-in-time mengembalikan (PITR) database Anda hingga yang kedua.
Pemantauan yang Ditingkatkan	Pilih Aktifkan pemantauan yang ditingkatkan untuk mengaktifkan metrik pengumpulan secara waktu nyata untuk sistem operasi tempat klaster DB Anda berjalan. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan.
Peran Pemantauan	Hanya tersedia jika Pemantauan yang Ditingkatkan diatur ke Aktifkan pemantauan yang ditingkatkan. Pilih peran IAM yang Anda buat untuk mengizinkan Aurora berkomunikasi dengan Log CloudWatch Amazon untuk Anda, atau pilih Default agar Aurora membuat peran untuk Anda beri nama. `rds-monitoring-role` Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan.
Granularitas	Hanya tersedia jika Pemantauan yang Ditingkatkan diatur ke Aktifkan pemantauan yang ditingkatkan. Atur interval, dalam detik, di antara waktu pengumpulan metrik untuk klaster DB Anda.
Peningkatan versi minor otomatis	Pengaturan ini tidak berlaku untuk klaster DB Aurora MySQL. Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora MySQL, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL.

Opsi	Deskripsi
Jendela pemeliharaan	Klik Pilih jendela dan tentukan rentang waktu mingguan saat pemeliharaan sistem dapat dilakukan . Atau, pilih Tidak ada preferensi bagi Aurora untuk menetapkan periode secara acak.

6. Pilih Buat replika baca.

AWS CLI

Untuk membuat replika baca Aurora dari instans DB RDS for MySQL sumber, gunakan perintah `create-db-cluster` dan `create-db-instance` AWS CLI untuk membuat klaster DB Aurora MySQL baru. Saat Anda memanggil perintah `create-db-cluster`, sertakan parameter `--replication-source-identifier` untuk mengidentifikasi Amazon Resource Name (ARN) untuk instans DB MySQL sumber. Untuk informasi selengkapnya tentang ARN Amazon RDS, lihat Amazon Relational Database Service (Amazon RDS).

Jangan tentukan nama pengguna master, kata sandi master, atau nama basis data karena replika baca Aurora menggunakan nama pengguna master, kata sandi master, dan nama basis data yang sama dengan instans DB MySQL sumber.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 \
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

Untuk Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine
 aurora ^
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 ^
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:primary-
mysql-instance
```

Jika Anda menggunakan konsol untuk membuat replika baca Aurora, Aurora secara otomatis membuat instans primer untuk replika baca Aurora klaster DB Anda. Jika Anda menggunakan AWS

CLI untuk membuat replika baca Aurora, Anda harus secara eksplisit membuat instans primer untuk klaster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB.

Anda dapat membuat instans primer untuk klaster DB Anda dengan menggunakan perintah `create-db-instance` AWS CLI dengan parameter berikut.

- `--db-cluster-identifier`

  Nama klaster DB Anda.

- `--db-instance-class`

  Nama kelas instans DB yang akan digunakan untuk instans primer Anda.

- `--db-instance-identifier`

  Nama instans primer Anda.

- `--engine aurora`

Dalam contoh ini, Anda membuat instans primer bernama *myreadreplicainstance* untuk klaster DB bernama *myreadreplicacluster*, menggunakan kelas instans DB yang ditentukan dalam *myinstanceclass*.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier myreadreplicacluster \
    --db-instance-class myinstanceclass \
    --db-instance-identifier myreadreplicainstance \
    --engine aurora
```

Untuk Windows:

```
aws rds create-db-instance ^
    --db-cluster-identifier myreadreplicacluster ^
    --db-instance-class myinstanceclass ^
    --db-instance-identifier myreadreplicainstance ^
    --engine aurora
```

API RDS

Untuk membuat replika baca Aurora dari instans DB RDS for MySQL sumber, gunakan perintah API Amazon RDS `CreateDBInstance` dan `CreateDBCluster` untuk membuat klaster DB dan instans primer Aurora. Jangan tentukan nama pengguna master, kata sandi master, atau nama basis data karena replika baca Aurora menggunakan nama pengguna master, kata sandi master, dan nama basis data yang sama dengan instans DB RDS for MySQL sumber.

Anda dapat membuat klaster DB Aurora baru untuk replika baca Aurora dari instans DB RDS for MySQL sumber menggunakan perintah API Amazon RDS `CreateDBCluster` dengan parameter berikut:

- `DBClusterIdentifier`

  Nama klaster DB yang akan dibuat.

- `DBSubnetGroupName`

  Nama grup subnet DB yang akan dikaitkan dengan klaster DB ini.

- `Engine=aurora`
- `KmsKeyId`

  AWS KMS key untuk mengenkripsi klaster DB secara opsional, tergantung pada apakah instans DB MySQL Anda dienkripsi atau tidak.

  - Jika instans DB MySQL Anda tidak dienkripsi, tentukan kunci enkripsi agar klaster DB Anda dienkripsi saat diam. Jika tidak, klaster DB Anda akan dienkripsi saat diam menggunakan kunci enkripsi default untuk akun Anda.
  - Jika instans DB MySQL Anda dienkripsi, tentukan kunci enkripsi agar klaster DB Anda dienkripsi saat diam menggunakan kunci enkripsi yang ditentukan. Jika tidak, klaster DB Anda akan dienkripsi saat diam menggunakan kunci enkripsi untuk instans DB MySQL.

  > **Note**
  >
  > Anda tidak dapat membuat klaster DB yang tidak terenkripsi dari instans DB MySQL terenkripsi.

- `ReplicationSourceIdentifier`

Amazon Resource Name (ARN) untuk instans DB MySQL sumber. Untuk informasi selengkapnya tentang ARN Amazon RDS, lihat Amazon Relational Database Service (Amazon RDS).

- `VpcSecurityGroupIds`

  Daftar grup keamanan VPC EC2 yang akan dikaitkan dengan klaster DB ini.

Dalam contoh ini, Anda membuat klaster DB bernama *`myreadreplicacluster`* dari instans DB MySQL sumber dengan ARN yang diatur ke *`mysqlprimaryARN`*, yang terkait dengan grup subnet DB bernama *`mysubnetgroup`* dan grup keamanan VPC bernama *`mysecuritygroup`*.

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBCluster
    &DBClusterIdentifier=myreadreplicacluster
    &DBSubnetGroupName=mysubnetgroup
    &Engine=aurora
    &ReplicationSourceIdentifier=mysqlprimaryARN
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-10-31
    &VpcSecurityGroupIds=mysecuritygroup
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20150927/us-east-1/rds/aws4_request
    &X-Amz-Date=20150927T164851Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=6a8f4bd6a98f649c75ea04a6b3929ecc75ac09739588391cd7250f5280e716db
```

Jika Anda menggunakan konsol untuk membuat replika baca Aurora, Aurora secara otomatis membuat instans primer untuk replika baca Aurora klaster DB Anda. Jika Anda menggunakan AWS CLI untuk membuat replika baca Aurora, Anda harus secara eksplisit membuat instans primer untuk klaster DB Anda. Instans primer adalah instans pertama yang dibuat dalam klaster DB.

Anda dapat membuat instans primer untuk klaster DB Anda dengan menggunakan perintah API Amazon RDS `CreateDBInstance` dengan parameter berikut:

- `DBClusterIdentifier`

  Nama klaster DB Anda.

- `DBInstanceClass`

Nama kelas instans DB yang akan digunakan untuk instans primer Anda.

- `DBInstanceIdentifier`

  Nama instans primer Anda.

- `Engine=aurora`

Dalam contoh ini, Anda membuat instans primer bernama *`myreadreplicainstance`* untuk klaster DB bernama *`myreadreplicacluster`*, menggunakan kelas instans DB yang ditentukan dalam *`myinstanceclass`*.

Example

```
https://rds.us-east-1.amazonaws.com/
    ?Action=CreateDBInstance
    &DBClusterIdentifier=myreadreplicacluster
    &DBInstanceClass=myinstanceclass
    &DBInstanceIdentifier=myreadreplicainstance
    &Engine=aurora
    &SignatureMethod=HmacSHA256
    &SignatureVersion=4
    &Version=2014-09-01
    &X-Amz-Algorithm=AWS4-HMAC-SHA256
    &X-Amz-Credential=AKIADQKE4SARGYLE/20140424/us-east-1/rds/aws4_request
    &X-Amz-Date=20140424T194844Z
    &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
    &X-Amz-Signature=bee4aabc750bf7dad0cd9e22b952bd6089d91e2a16592c2293e532eeaab8bc77
```

Melihat replika baca Aurora

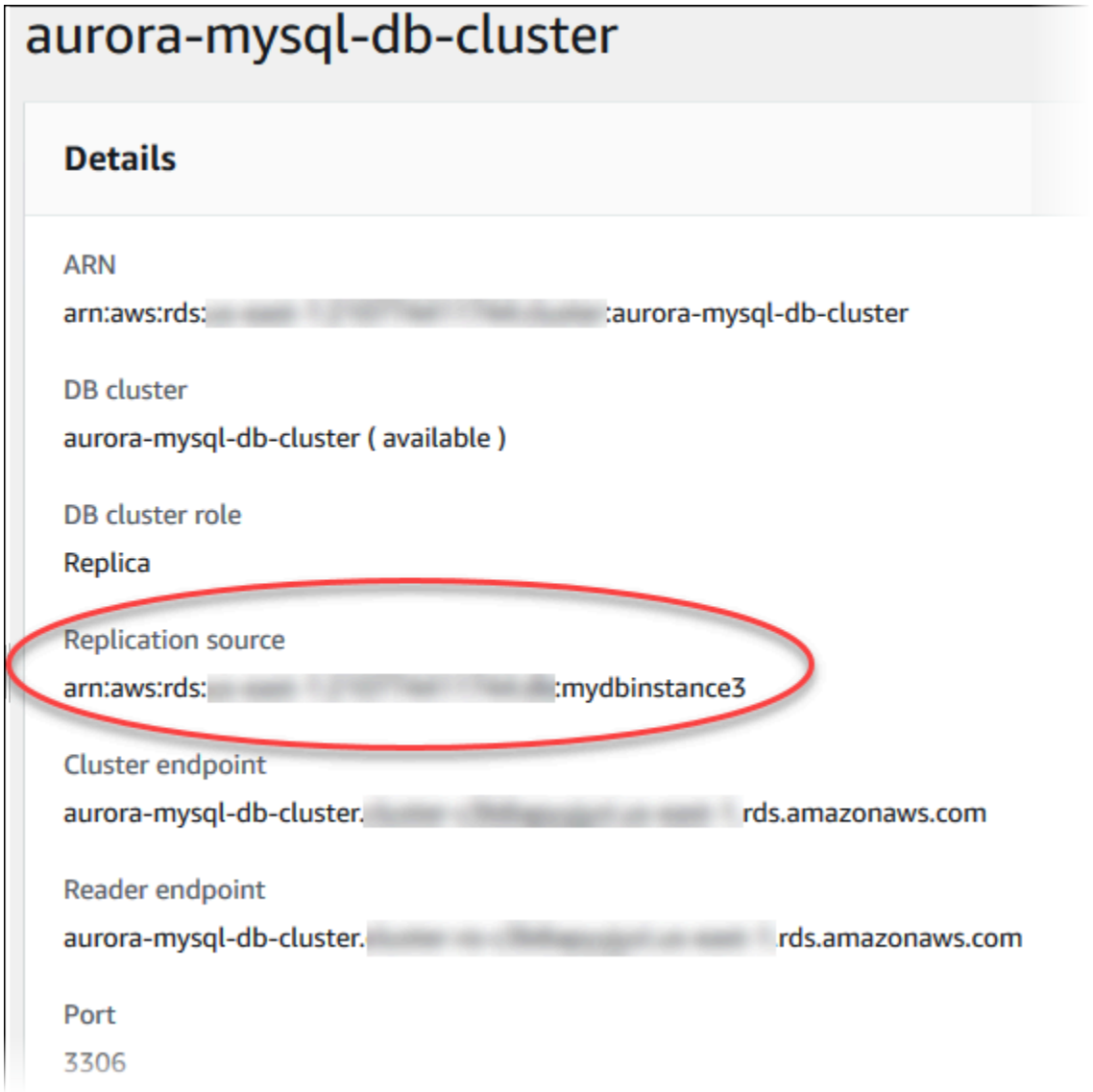
Anda dapat menampilkan relasi replikasi MySQL ke Aurora MySQL untuk klaster DB Aurora MySQL Anda dengan menggunakan AWS Management Console atau AWS CLI.

Konsol

Untuk menampilkan instans DB MySQL primer untuk replika baca Aurora

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.

3. Pilih klaster DB untuk replika baca Aurora untuk menampilkan detailnya. Informasi instans DB MySQL primer ada di bidang Sumber replikasi.



AWS CLI

Untuk menampilkan relasi replikasi MySQL ke Aurora MySQL untuk klaster DB Aurora MySQL Anda dengan menggunakan AWS CLI, gunakan perintah `describe-db-clusters` dan `describe-db-instances`.

Untuk menentukan instans DB MySQL mana yang primer, gunakan `describe-db-clusters` dan tentukan pengidentifikasi klaster replika baca Aurora untuk opsi `--db-cluster-identifier`. Lihat elemen `ReplicationSourceIdentifier` dalam output untuk ARN instans DB yang merupakan replikasi primer.

Untuk menentukan klaster DB mana yang merupakan replika baca Aurora, gunakan `describe-db-instances` dan tentukan pengidentifikasi instans DB MySQL untuk opsi `--db-instance-identifier`. Lihat elemen `ReadReplicaDBClusterIdentifiers` dalam output untuk pengidentifikasi klaster DB dari replika baca Aurora.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-clusters \
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances \
    --db-instance-identifier mysqlprimary
```

Untuk Windows:

```
aws rds describe-db-clusters ^
    --db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances ^
    --db-instance-identifier mysqlprimary
```

Mempromosikan replika baca Aurora

Setelah migrasi selesai, Anda dapat mempromosikan replika baca Aurora ke klaster DB mandiri menggunakan AWS Management Console atau AWS CLI.

Kemudian, Anda dapat mengarahkan aplikasi klien Anda ke titik akhir untuk replika baca Aurora. Untuk informasi selengkapnya tentang titik akhir Aurora, lihat Manajemen koneksi Amazon Aurora. Promosi akan selesai dengan cukup cepat, dan Anda dapat membaca dan menulis ke replika baca Aurora selama promosi. Namun, Anda tidak dapat menghapus instans DB MySQL primer atau membatalkan tautan Instans DB dan replika baca Aurora selama waktu ini.

Sebelum Anda mempromosikan replika baca Aurora, hentikan transaksi apa pun agar tidak ditulis ke instans DB MySQL sumber, lalu tunggu hingga lag replika pada replika baca Aurora mencapai 0. Anda dapat melihat lag replika untuk replika baca Aurora dengan memanggil perintah `SHOW SLAVE STATUS` (Aurora MySQL versi 2) atau `SHOW REPLICA STATUS` (Aurora MySQL versi 3) pada replika baca Aurora Anda. Periksa nilai Detik di belakang master.

Anda dapat mulai menulis ke replika baca Aurora setelah transaksi tulis ke primer berhenti dan lag replika adalah 0. Jika Anda menulis ke replika baca Aurora sebelum hal ini terjadi dan Anda memodifikasi tabel yang juga sedang dimodifikasi di MySQL primer, Anda berisiko merusak replikasi ke Aurora. Jika ini terjadi, Anda harus menghapus dan membuat ulang replika baca Aurora Anda.

Konsol

Untuk mempromosikan replika baca Aurora ke klaster DB Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB untuk replika baca Aurora.
4. Untuk Tindakan, pilih Promosikan.
5. Pilih Promosikan replika baca.

Setelah mempromosikan, konfirmasikan bahwa proses promosinya telah selesai dengan menggunakan prosedur berikut.

Untuk mengonfirmasi bahwa replika baca Aurora telah dipromosikan

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Peristiwa.
3. Di halaman Peristiwa, verifikasi bahwa ada peristiwa `Promoted Read Replica cluster to a stand-alone database cluster` untuk klaster yang Anda promosikan.

Setelah promosi selesai, instans DB MySQL primer dan replika baca Aurora akan dibatalkan tautannya, dan Anda dapat menghapus instans DB dengan aman jika ingin.

AWS CLI

Untuk mempromosikan replika baca Aurora ke klaster DB mandiri, gunakan perintah `promote-read-replica-db-cluster` AWS CLI.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

Untuk Windows:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

Mengelola Amazon Aurora MySQL

Bagian berikut membahas pengelolaan klaster DB Amazon Aurora MySQL.

Topik

- Mengelola performa dan penskalaan untuk Amazon Aurora MySQL
- Melakukan backtracking klaster DB Aurora
- Menguji Amazon Aurora MySQL menggunakan kueri injeksi kesalahan
- Mengubah tabel di Amazon Aurora menggunakan DDL Cepat
- Menampilkan status volume untuk klaster DB Aurora MySQL

Mengelola performa dan penskalaan untuk Amazon Aurora MySQL

Penskalaan instans DB Aurora MySQL

Anda dapat menskalakan instans DB Aurora MySQL dengan dua cara, penskalaan instans dan penskalaan baca. Untuk informasi selengkapnya tentang penskalaan baca, lihat Penskalaan baca.

Anda dapat menskalakan klaster DB Aurora MySQL Anda dengan mengubah kelas instans DB untuk setiap instans DB dalam klaster DB. Aurora MySQL mendukung beberapa kelas instans DB yang dioptimalkan untuk Aurora. Jangan gunakan kelas instans db.t2 atau db.t3 untuk klaster Aurora yang berukuran lebih dari 40 TB. Untuk spesifikasi kelas instans DB yang didukung Aurora MySQL, lihat Kelas instans DB Aurora.

> **Note**
>
> Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Menggunakan kelas instans T untuk pengembangan dan pengujian.

Koneksi maksimum ke instans DB Aurora MySQL

Jumlah maksimum koneksi yang diizinkan untuk instans DB Aurora MySQL ditentukan oleh parameter `max_connections` dalam grup parameter tingkat instans untuk instans DB.

Tabel berikut mencantumkan nilai default yang dihasilkan berupa `max_connections` untuk setiap kelas instans DB yang tersedia untuk Aurora MySQL. Anda dapat meningkatkan jumlah maksimum

koneksi ke instans DB Aurora MySQL Anda dengan menaikkan skala instans ke kelas instans DB dengan lebih banyak memori, atau dengan menetapkan nilai yang lebih besar untuk parameter `max_connections` di grup parameter DB untuk instans Anda, hingga 16.000.

> **Tip**
>
> Jika aplikasi Anda sering membuka dan menutup koneksi, atau membiarkan sejumlah besar koneksi yang lama aktif tetap terbuka, kami menyarankan agar Anda menggunakan Proksi Amazon RDS. Proksi RDS adalah proksi basis data yang sepenuhnya terkelola dan memiliki ketersediaan tinggi yang menggunakan kumpulan koneksi untuk berbagi koneksi basis data dengan aman dan efisien. Untuk mempelajari tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS untuk Aurora.

Untuk detail tentang cara instans Aurora Serverless v2 menangani parameter ini, lihat Koneksi maksimum untuk Aurora Serverless v2.

Kelas instans	Nilai default max_connections		
db.t2.small	45		
db.t2.medium	90		
db.t3.small	45		
db.t3.medium	90		
db.t3.large	135		
db.t4g.medium	90		
db.t4g.large	135		
db.r3.large	1000		
db.r3.xlarge	2000		

Kelas instans	Nilai default max_connections		
db.r3.2xlarge	3000		
db.r3.4xlarge	4000		
db.r3.8xlarge	5000		
db.r4.large	1000		
db.r4.xlarge	2000		
db.r4.2xlarge	3000		
db.r4.4xlarge	4000		
db.r4.8xlarge	5000		
db.r4.16xlarge	6000		
db.r5.large	1000		
db.r5.xlarge	2000		
db.r5.2xlarge	3000		
db.r5.4xlarge	4000		
db.r5.8xlarge	5000		
db.r5.12xlarge	6000		
db.r5.16xlarge	6000		
db.r5.24xlarge	7000		
db.r6g.large	1000		
db.r6g.xlarge	2000		

Kelas instans	Nilai default max_connections		
db.r6g.2xlarge	3000		
db.r6g.4xlarge	4000		
db.r6g.8xlarge	5000		
db.r6g.12xlarge	6000		
db.r6g.16xlarge	6000		
db.r6i.large	1000		
db.r6i.xlarge	2000		
db.r6i.2xlarge	3000		
db.r6i.4xlarge	4000		
db.r6i.8xlarge	5000		
db.r6i.12xlarge	6000		
db.r6i.16xlarge	6000		
db.r6i.24xlarge	7000		
db.r6i.32xlarge	7000		
db.x2g.large	2000		
db.x2g.xlarge	3000		
db.x2g.2xlarge	4000		
db.x2g.4xlarge	5000		
db.x2g.8xlarge	6000		

Kelas instans	Nilai default max_connections		
db.x2g.12xlarge	7000		
db.x2g.16xlarge	7000		

Jika Anda membuat grup parameter baru untuk menyesuaikan nilai default Anda sendiri untuk batas koneksi, Anda akan melihat bahwa batas koneksi default akan diperoleh menggunakan formula berdasarkan nilai `DBInstanceClassMemory`. Seperti ditunjukkan dalam tabel sebelumnya, rumus ini menghasilkan batas koneksi yang bertambah sebanyak 1000 saat memorinya berlipat ganda antara instans R3, R4, dan R5 yang makin besar, dan sebanyak 45 untuk ukuran memori yang berbeda-beda pada instans T2 dan T3.

Lihat Menentukan parameter DB untuk detail selengkapnya tentang cara `DBInstanceClassMemory` dihitung.

instans DB Aurora MySQL dan RDS for MySQL memiliki jumlah overhead memori yang berbeda. Oleh karena itu, nilai `max_connections` dapat berbeda untuk instans DB Aurora MySQL dan RDS for MySQL yang menggunakan kelas instans yang sama. Nilai-nilai dalam tabel ini hanya berlaku untuk instans DB Aurora MySQL.

> **Note**
>
> Batas konektivitas untuk instans T2 dan T3 jauh lebih rendah karena dengan Aurora, kelas instans tersebut hanya ditujukan untuk skenario pengembangan dan pengujian, bukan untuk beban kerja produksi.

Batas koneksi default disesuaikan untuk sistem yang menggunakan nilai default untuk pemakai memori besar lainnya, seperti pool buffer dan cache kueri. Jika Anda mengubah pengaturan lain tersebut untuk klaster Anda, pertimbangkan untuk menyesuaikan batas koneksi agar memperhitungkan peningkatan atau penurunan memori yang tersedia pada instans DB.

Batas penyimpanan sementara untuk Aurora MySQL

Aurora MySQL menyimpan tabel dan indeks dalam subsistem penyimpanan Aurora. Aurora MySQL menggunakan penyimpanan sementara terpisah untuk file sementara nonpersisten. Aurora MySQL menggunakan penyimpanan lokal untuk menyimpan log kesalahan, log umum, log kueri lambat, log audit, dan tabel sementara non-InnoDB. Penyimpanan lokal juga mencakup file yang digunakan untuk tujuan seperti mengurutkan set data besar selama pemrosesan kueri atau untuk operasi pembuatan indeks. Ini tidak termasuk tabel sementara InnoDB. Untuk informasi selengkapnya, lihat artikel Apa yang disimpan di penyimpanan lokal yang kompatibel dengan Aurora MySQL, dan bagaimana saya dapat memecahkan masalah penyimpanan lokal?. Untuk informasi selengkapnya tentang tabel sementara di Aurora MySQL versi 3, lihat Perilaku tabel sementara baru di Aurora MySQL versi 3. Untuk informasi selengkapnya tentang tabel sementara di versi 2, lihat Perilaku ruang tabel sementara di Aurora MySQL versi 2.

Volume penyimpanan lokal ini didukung oleh Amazon Elastic Block Store (EBS) dan dapat diperluas dengan menggunakan kelas instans DB yang lebih besar. Untuk informasi selengkapnya tentang penyimpanan, lihat Penyimpanan dan keandalan Amazon Aurora.

> **Note**
>
> Anda mungkin melihat peristiwa `storage-optimization` saat menskalakan instans DB, misalnya, dari db.r5.2xlarge ke db.r5.4xlarge.

Tabel berikut menunjukkan jumlah maksimum penyimpanan sementara yang tersedia untuk setiap kelas instans DB Aurora MySQL. Untuk informasi selengkapnya tentang dukungan kelas instans DB untuk Aurora, lihat Kelas instans DB Aurora.

Kelas instans DB	Penyimpanan sementara maksimum yang tersedia (GiB)
db.x2g.16xlarge	1280
db.x2g.12xlarge	960
db.x2g.8xlarge	640
db.x2g.4xlarge	320

Kelas instans DB	Penyimpanan sementara maksimum yang tersedia (GiB)
db.x2g.2xlarge	160
db.x2g.xlarge	80
db.x2g.large	40
db.r6i.32xlarge	2560
db.r6i.24xlarge	1920
db.r6i.16xlarge	1280
db.r6i.12xlarge	960
db.r6i.8xlarge	640
db.r6i.4xlarge	320
db.r6i.2xlarge	160
db.r6i.xlarge	80
db.r6i.large	32
db.r6g.16xlarge	1280
db.r6g.12xlarge	960
db.r6g.8xlarge	640
db.r6g.4xlarge	320
db.r6g.2xlarge	160
db.r6g.xlarge	80
db.r6g.large	32
db.r5.24xlarge	1920

Kelas instans DB	Penyimpanan sementara maksimum yang tersedia (GiB)
db.r5.16xlarge	1280
db.r5.12xlarge	960
db.r5.8xlarge	640
db.r5.4xlarge	320
db.r5.2xlarge	160
db.r5.xlarge	80
db.r5.large	32
db.r4.16xlarge	1280
db.r4.8xlarge	640
db.r4.4xlarge	320
db.r4.2xlarge	160
db.r4.xlarge	80
db.r4.large	32
db.t4g.large	32
db.t4g.medium	32
db.t3.large	32
db.t3.medium	32
db.t3.small	32
db.t2.medium	32
db.t2.small	32

> **Important**
>
> Nilai-nilai ini merepresentasikan jumlah maksimum teoretis penyimpanan kosong di setiap instans DB. Penyimpanan lokal sebenarnya yang tersedia untuk Anda mungkin lebih rendah. Aurora menggunakan beberapa penyimpanan lokal untuk proses manajemennya, dan instans DB menggunakan beberapa penyimpanan lokal bahkan sebelum Anda memuat data apa pun. Anda dapat memantau penyimpanan sementara yang tersedia untuk instans DB tertentu dengan `FreeLocalStorage` CloudWatch metrik, yang dijelaskan dalam CloudWatch Metrik Amazon untuk Amazon Aurora. Anda dapat memeriksa jumlah penyimpanan kosong saat ini. Anda juga dapat memetakan jumlah penyimpanan kosong dari waktu ke waktu. Memantau penyimpanan kosong dari waktu ke waktu membantu Anda menentukan apakah nilainya meningkat atau menurun, atau untuk menemukan nilai minimum, maksimum, atau rata-rata.
> (Hal ini tidak berlaku untuk Aurora Serverless v2.)

Melakukan backtracking klaster DB Aurora

Dengan Amazon Aurora Edisi Kompatibel MySQL, Anda dapat melakukan backtracking klaster DB ke waktu tertentu, tanpa memulihkan data dari cadangan.

Daftar Isi

- Gambaran umum backtracking
  - Jendela backtrack
  - Waktu backtracking
  - Batasan backtracking
- Ketersediaan wilayah dan versi
- Pertimbangan peningkatan untuk klaster yang memiliki backtrack aktif
- Mengonfigurasi backtracking
- Melakukan backtrack
- Memantau backtracking
- Berlangganan peristiwa backtrack dengan konsol
- Mengambil backtrack yang ada
- Menonaktifkan backtracking untuk klaster DB

Gambaran umum backtracking

Backtracking “memundurkan” klaster DB ke waktu yang Anda tentukan. Backtracking bukan pengganti untuk mencadangkan klaster DB sehingga Anda dapat memulihkannya ke suatu titik waktu. Namun, backtracking memberikan keuntungan berikut dibandingkan pencadangan dan pemulihan biasa:

- Anda dapat dengan mudah membatalkan kesalahan. Jika Anda tanpa sengaja melakukan tindakan destruktif, seperti DELETE tanpa klausa WHERE, Anda dapat melakukan backtracking klaster DB ke waktu sebelum tindakan destruktif tersebut dilakukan dengan gangguan layanan minimal.
- Anda dapat melakukan backtracking klaster DB dengan cepat. Memulihkan klaster DB ke titik waktu akan meluncurkan klaster DB baru dan memulihkannya dari data cadangan atau snapshot klaster DB, yang dapat memakan waktu berjam-jam. Backtracking klaster DB tidak memerlukan klaster DB baru dan memundurkan klaster DB dalam hitungan menit.
- Anda dapat menjelajahi perubahan data sebelumnya. Anda dapat berulang kali melakukan backtracking klaster DB secara tepat waktu untuk membantu menentukan kapan perubahan data tertentu terjadi. Misalnya, Anda dapat melakukan backtracking klaster DB tiga jam lalu melacak maju dalam waktu satu jam. Dalam hal ini, waktu backtrack adalah dua jam sebelum waktu asli.

> **Note**
>
> Untuk informasi tentang memulihkan klaster DB ke suatu titik waktu, lihat Gambaran umum pencadangan dan pemulihan klaster DB Aurora.

Jendela backtrack

Dengan backtrack, ada jendela backtrack target dan jendela backtrack aktual:

- Jendela backtrack target adalah jumlah waktu yang Anda inginkan untuk dapat melakukan backtracking klaster DB Anda. Saat Anda mengaktifkan backtrack, Anda menentukan jendela backtrack target. Misalnya, Anda dapat menentukan jendela backtrack target selama 24 jam jika Anda ingin dapat melakukan backtracking klaster DB selama satu hari.
- Jendela backtrack aktual adalah jumlah waktu aktual yang Anda inginkan untuk dapat melakukan backtracking klaster DB, yang dapat lebih kecil daripada jendela backtrack target. Jendela backtrack aktual didasarkan pada beban kerja dan penyimpanan yang tersedia untuk menyimpan informasi tentang perubahan basis data, yang disebut catatan perubahan.

Saat Anda membuat pembaruan pada klaster Aurora DB dengan backtracking aktif, Anda menghasilkan catatan perubahan. Aurora menyimpan catatan perubahan untuk jendela backtrack target, dan Anda membayar tarif per jam untuk menyimpannya. Jendela backtrack target dan beban kerja di klaster DB Anda menentukan jumlah catatan perubahan yang Anda simpan. Beban kerja adalah jumlah perubahan yang Anda buat pada klaster DB Anda dalam waktu tertentu. Jika beban kerja Anda berat, Anda menyimpan lebih banyak perubahan catatan di jendela backtrack dibandingkan jika beban kerja Anda ringan.

Anda dapat membayangkan jendela backtrack target sebagai sasaran jumlah waktu yang Anda inginkan untuk dapat melakukan backtracking klaster DB Anda. Dalam kebanyakan kasus, Anda dapat melakukan backtracking dalam jumlah waktu maksimum yang Anda tentukan. Namun, dalam beberapa kasus, klaster DB tidak dapat menyimpan cukup catatan untuk melakukan backtracking dalam jumlah waktu maksimum, dan jendela backtrack aktual Anda lebih kecil daripada target Anda. Biasanya, jendela backtrack aktual lebih kecil daripada target saat Anda memiliki beban kerja yang sangat berat pada klaster DB Anda. Saat jendela backtrack aktual Anda lebih kecil dari target, kami mengirimkan notifikasi kepada Anda.

Saat backtracking diaktifkan untuk klaster DB, dan Anda menghapus tabel yang disimpan dalam klaster DB, Aurora mempertahankan tabel tersebut di catatan perubahan backtrack. Ini dilakukan agar Anda dapat kembali ke waktu sebelum Anda menghapus tabel. Jika Anda tidak memiliki cukup ruang di jendela backtrack Anda untuk menyimpan tabel, pada akhirnya tabel mungkin akan dihapus dari catatan perubahan backtrack.

Waktu backtracking

Aurora selalu melakukan backtracking ke waktu yang konsisten untuk klaster DB. Tindakan ini menghilangkan kemungkinan transaksi yang tidak di-commit saat backtrack selesai. Saat Anda menentukan waktu untuk backtrack, Aurora akan secara otomatis memilih waktu konsisten sedekat mungkin. Pendekatan ini berarti bahwa backtrack yang diselesaikan mungkin tidak sama persis dengan waktu yang Anda tentukan, tetapi Anda dapat menentukan waktu yang tepat untuk backtrack dengan menggunakan perintah CLI describe-db-cluster-backtracks AWS . Untuk informasi selengkapnya, lihat Mengambil backtrack yang ada.

Batasan backtracking

Batasan berikut berlaku untuk backtracking:

- Backtracking hanya tersedia untuk klaster DB yang dibuat dengan fitur Backtrack diaktifkan. Anda tidak dapat memodifikasi cluster DB untuk mengaktifkan fitur Backtrack. Anda dapat mengaktifkan fitur Backtrack saat membuat klaster DB baru atau memulihkan snapshot klaster DB.

- Batasan untuk jendela backtrack adalah 72 jam.
- Backtracking memengaruhi seluruh klaster DB. Misalnya, Anda tidak dapat memilih untuk melakukan backtracking satu tabel atau satu pembaruan data.
- Backtracking tidak didukung dengan replikasi log biner (binlog). Replikasi lintas Wilayah harus dinonaktifkan sebelum Anda dapat mengonfigurasi atau menggunakan backtracking.
- Anda tidak dapat melakukan backtracking klon basis data ke waktu sebelum klon basis data tersebut dibuat. Namun, Anda dapat menggunakan basis data asli untuk melakukan backtracking ke suatu waktu sebelum klon dibuat. Untuk informasi selengkapnya tentang kloning basis data, lihat Mengkloning volume untuk klaster DB Amazon Aurora.
- Backtracking akan menyebabkan gangguan instans DB dalam waktu singkat. Anda harus menghentikan atau menjeda aplikasi sebelum memulai operasi backtrack untuk memastikan bahwa tidak ada permintaan baca atau tulis baru. Selama operasi backtrack, Aurora menjeda basis data, menutup semua koneksi yang terbuka, dan menghentikan operasi baca dan tulis yang tidak di-commit. Aurora kemudian menunggu operasi backtrack selesai.
- Anda tidak dapat memulihkan snapshot Lintas wilayah dari klaster berkemampuan backtrack di AWS Wilayah yang tidak mendukung backtracking.
- Jika Anda melakukan peningkatan di tempat untuk klaster yang memiliki backtrack aktif dari Aurora MySQL versi 2 ke versi 3, Anda tidak dapat melakukan backtracking ke titik waktu sebelum peningkatan terjadi.

Ketersediaan wilayah dan versi

Backtracking tidak tersedia untuk Aurora PostgreSQL.

Berikut ini adalah mesin yang didukung dan ketersediaan Wilayah untuk Backtrack dengan Aurora MySQL.

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AS Timur (Ohio)	Semua versi	Semua versi
AS Timur (Virginia Utara)	Semua versi	Semua versi
AS Barat (California Utara)	Semua versi	Semua versi

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
AS Barat (Oregon)	Semua versi	Semua versi
Afrika (Cape Town)	–	–
Asia Pasifik (Hong Kong)	–	–
Asia Pasifik (Jakarta)	–	–
Asia Pasifik (Melbourne)	–	–
Asia Pasifik (Mumbai)	Semua versi	Semua versi
Asia Pasifik (Osaka)	Semua versi	Versi 2.07.3 dan lebih tinggi
Asia Pasifik (Seoul)	Semua versi	Semua versi
Asia Pasifik (Singapura)	Semua versi	Semua versi
Asia Pasifik (Sydney)	Semua versi	Semua versi
Asia Pasifik (Tokyo)	Semua versi	Semua versi
Kanada (Pusat)	Semua versi	Semua versi
Kanada Barat (Calgary)	–	–

Wilayah	Aurora MySQL versi 3	Aurora MySQL versi 2
China (Beijing)	–	–
Tiongkok (Ningxia)	–	–
Eropa (Frankfurt)	Semua versi	Semua versi
Eropa (Irlandia)	Semua versi	Semua versi
Eropa (London)	Semua versi	Semua versi
Eropa (Milan)	–	–
Eropa (Paris)	Semua versi	Semua versi
Eropa (Spanyol)	–	–
Eropa (Stockholm)	–	–
Eropa (Zürich)	–	–
Israel (Tel Aviv)	–	–
Timur Tengah (Bahrain)	–	–
Timur Tengah (UEA)	–	–
Amerika Selatan (Sao Paulo)	–	–
AWS GovCloud (AS-Timur)	–	–
AWS GovCloud (AS-Barat)	–	–

Pertimbangan peningkatan untuk klaster yang memiliki backtrack aktif

Anda dapat meningkatkan klaster DB yang memiliki backtrack aktif dari Aurora MySQL versi 2 ke versi 3 karena semua versi minor Aurora MySQL versi 3 didukung untuk Backtrack.

Mengonfigurasi backtracking

Untuk menggunakan fitur Backtrack, Anda harus mengaktifkan backtracking dan menentukan jendela backtrack target. Jika tidak, backtracking dinonaktifkan.

Untuk Jendela Backtrack Target, tentukan jumlah waktu yang Anda inginkan untuk dapat memundurkan basis data Anda menggunakan Backtrack. Aurora mencoba mempertahankan catatan perubahan yang cukup untuk mendukung periode waktu tersebut.

Konsol

Anda dapat menggunakan konsol untuk mengonfigurasi backtracking saat Anda membuat klaster DB baru. Anda juga dapat memodifikasi klaster DB untuk mengubah jendela backtrack untuk klaster yang memiliki backtrack aktif. Jika Anda menonaktifkan backtrack sepenuhnya untuk klaster dengan mengatur jendela backtrack ke 0, Anda tidak dapat mengaktifkan backtrack lagi untuk klaster tersebut.

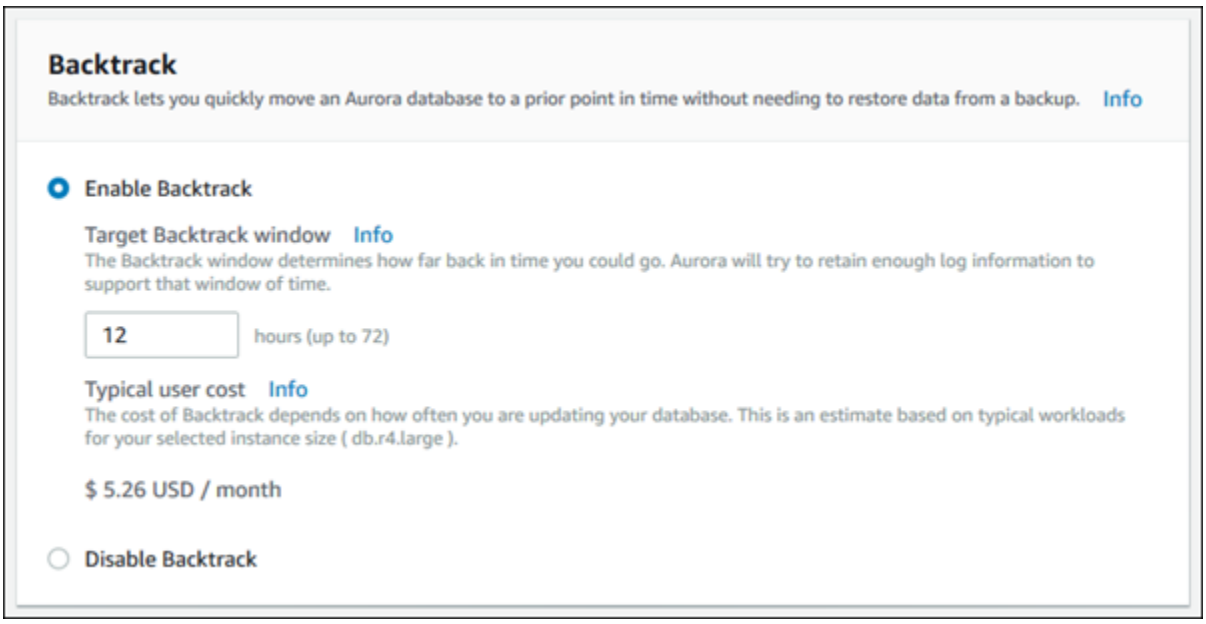
Topik

- Mengonfigurasi backtracking dengan konsol saat membuat klaster DB
- Mengonfigurasi backtrack dengan konsol saat memodifikasi klaster DB

Mengonfigurasi backtracking dengan konsol saat membuat klaster DB

Saat Anda membuat klaster DB Aurora MySQL baru, backtracking akan dikonfigurasi jika Anda memilih Aktifkan Backtrack dan menentukan Jendela Backtrack Target yang lebih besar dari nol di bagian Backtrack.

Untuk membuat klaster DB, ikuti petunjuk dalam Membuat klaster DB Amazon Aurora. Gambar berikut menunjukkan bagian Backtrack.



Ketika Anda membuat klaster DB baru, Aurora tidak memiliki data untuk beban kerja klaster DB. Jadi, Aurora tidak memperkirakan biaya secara khusus untuk klaster DB baru. Sebagai gantinya, konsol akan menampilkan biaya pengguna standar untuk jendela backtrack target yang ditentukan berdasarkan beban kerja standar. Biaya standar dimaksudkan untuk memberikan referensi umum untuk biaya fitur Backtrack.

> **Important**
>
> Biaya Anda sebenarnya mungkin tidak sama dengan biaya standar karena biaya Anda sebenarnya didasarkan pada beban kerja klaster DB Anda.

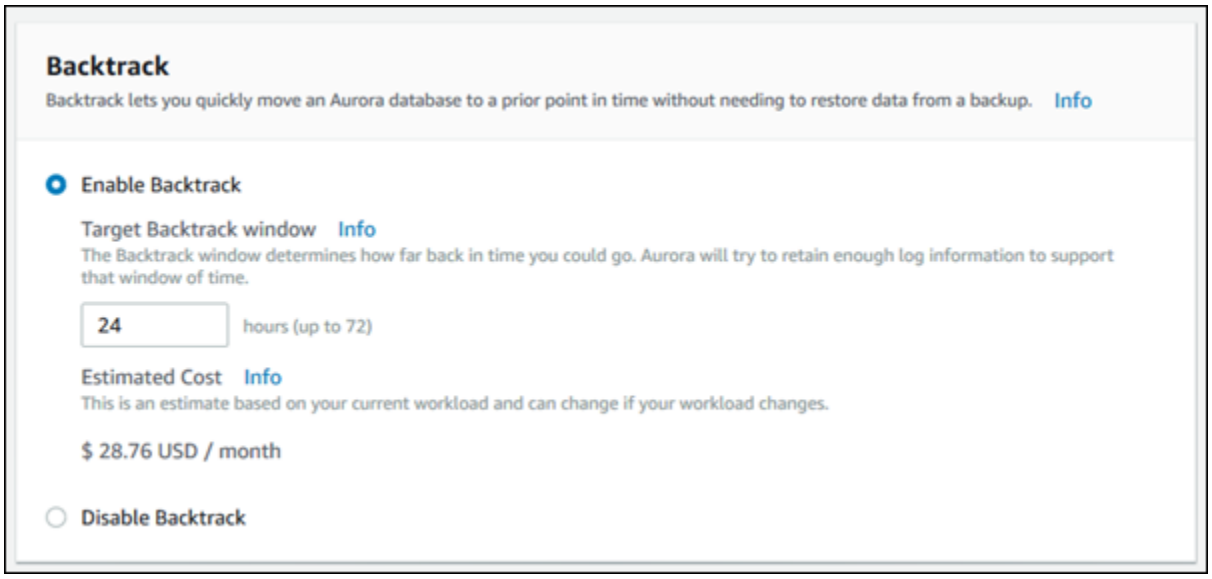
Mengonfigurasi backtrack dengan konsol saat memodifikasi klaster DB

Anda dapat memodifikasi backtrack untuk klaster DB menggunakan konsol.

> **Note**
>
> Saat ini, Anda dapat memodifikasi backtracking hanya untuk klaster DB yang memiliki fitur Backtrack aktif. Bagian Backtrack ini tidak muncul untuk klaster DB yang dibuat dengan fitur Backtrack dinonaktifkan atau jika fitur Backtrack telah dinonaktifkan untuk klaster DB tersebut.

Untuk memodifikasi backtracking untuk klaster DB menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih klaster yang ingin Anda ubah, dan pilih Modifikasi.
4. Untuk Jendela Backtrack Target, ubah jumlah waktu yang Anda inginkan untuk dapat melakukan backtracking. Batasannya adalah 72 jam.



   Konsol menunjukkan perkiraan biaya untuk jumlah waktu yang Anda tentukan berdasarkan beban kerja terdahulu di klaster DB:

   - Jika backtracking dinonaktifkan pada cluster DB, perkiraan biaya didasarkan pada `VolumeWriteIOPS` metrik untuk cluster DB di Amazon CloudWatch.
   - Jika backtracking diaktifkan sebelumnya di cluster DB, perkiraan biaya didasarkan pada `BacktrackChangeRecordsCreationRate` metrik untuk cluster DB di Amazon CloudWatch.
5. Pilih Lanjutkan.
6. Untuk Penjadwalan Modifikasi, pilih salah satu dari berikut ini:

   - Terapkan selama jendela pemeliharaan terjadwal berikutnya – Tunggu untuk menerapkan modifikasi Jendela Backtrack Target hingga periode pemeliharaan berikutnya.
   - Terapkan segera – Terapkan modifikasi Jendela Backtrack Target sesegera mungkin.
7. Pilih Ubah klaster.

AWS CLI

Saat Anda membuat cluster Aurora MySQL DB baru menggunakan perintah create-db-cluster AWS CLI, backtracking dikonfigurasi saat Anda menentukan nilai yang lebih besar dari nol. `--backtrack-window` Nilai `--backtrack-window` menentukan jendela backtrack target. Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.

Anda juga dapat menentukan `--backtrack-window` nilai menggunakan perintah AWS CLI berikut:

- modify-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

Prosedur berikut menjelaskan cara memodifikasi jendela backtrack target untuk klaster DB menggunakan AWS CLI.

Untuk memodifikasi jendela backtrack target untuk cluster DB menggunakan AWS CLI

- Panggil perintah modify-db-cluster AWS CLI dan berikan nilai-nilai berikut:
  - `--db-cluster-identifier` – Nama klaster DB.
  - `--backtrack-window` – Jumlah maksimum detik yang Anda inginkan untuk melakukan backtracking klaster DB.

  Contoh berikut mengatur jendela backtrack target untuk `sample-cluster` ke satu hari (86.400 detik).

  Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 86400
```

  Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 86400
```

> **Note**
>
> Saat ini, Anda dapat mengaktifkan backtracking hanya untuk klaster DB yang dibuat dengan fitur Backtrack aktif.

API RDS

Saat Anda membuat klaster DB Aurora MySQL baru menggunakan operasi CreateDBCluster API Amazon RDS, backtracking dikonfigurasi ketika Anda menentukan nilai `BacktrackWindow` yang lebih besar dari nol. Nilai `BacktrackWindow` menentukan jendela backtrack target untuk klaster DB yang ditentukan dalam nilai `DBClusterIdentifier`. Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.

Anda juga dapat menentukan nilai `BacktrackWindow` yang sesuai dengan operasi API berikut:

- ModifyDBCluster
- Dipulihkan S3 ClusterFrom
- Dipulihkan ClusterFromSnapshot
- Dipulihkan ClusterToPointInTime

> **Note**
>
> Saat ini, Anda dapat mengaktifkan backtracking hanya untuk klaster DB yang dibuat dengan fitur Backtrack aktif.

Melakukan backtrack

Anda dapat melakukan backtracking klaster DB ke stempel waktu backtrack tertentu. Jika stempel waktu backtrack tidak lebih awal dari waktu backtrack yang paling awal, dan tidak di masa depan, klaster DB akan di-backtracking ke stempel waktu tersebut.

Jika tidak, biasanya terjadi kesalahan. Selain itu, jika Anda mencoba untuk melakukan backtracking klaster DB yang memiliki pencatatan log biner aktif, kesalahan biasanya terjadi kecuali jika Anda telah memilih untuk memaksa backtrack untuk terjadi. Memaksa backtrack agar terjadi dapat mengganggu operasi lain yang menggunakan pencatatan log biner.

Important

Backtracking tidak menghasilkan entri binlog untuk perubahan yang dibuatnya. Jika Anda mengaktifkan pencatatan log biner untuk klaster DB, backtracking mungkin tidak kompatibel dengan implementasi binlog Anda.

Note

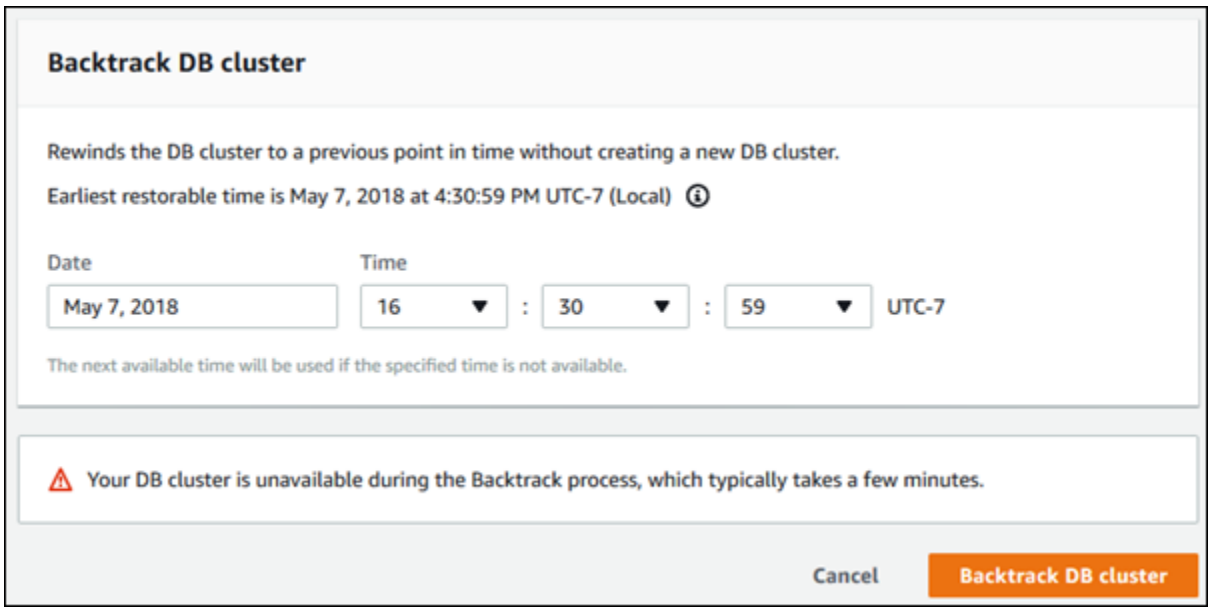
Untuk klon basis data, Anda tidak dapat melakukan backtracking klaster DB lebih awal dari tanggal dan waktu ketika klon dibuat. Untuk informasi selengkapnya tentang kloning basis data, lihat Mengkloning volume untuk klaster DB Amazon Aurora.

Konsol

Prosedur berikut menjelaskan cara melakukan operasi backtrack untuk klaster DB menggunakan konsol.

Untuk melakukan operasi backtrack menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Instans.
3. Pilih instans primer untuk klaster DB yang ingin Anda backtrack.
4. Untuk Tindakan, pilih Klaster DB Backtrack.
5. Di halaman Klaster DB Backtrack, masukkan stempel waktu backtrack untuk melakukan backtracking klaster DB.



6. Pilih Klaster DB Backtrack.

AWS CLI

Prosedur berikut menjelaskan cara melakukan backtracking klaster DB menggunakan AWS CLI.

Untuk mundur cluster DB menggunakan AWS CLI

- Panggil perintah backtrack-db-cluster AWS CLI dan berikan nilai-nilai berikut:
  - `--db-cluster-identifier` – Nama klaster DB.
  - `--backtrack-to` – Stempel waktu backtrack untuk melakukan backtracking klaster DB, yang ditentukan dalam format ISO 8601.

  Contoh berikut melakukan backtracking klaster DB `sample-cluster` ke 19 Maret 2018, pukul 10.00.

  Untuk Linux, macOS, atau Unix:

```
aws rds backtrack-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-to 2018-03-19T10:00:00+00:00
```

  Untuk Windows:

```
aws rds backtrack-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-to 2018-03-19T10:00:00+00:00
```

API RDS

Untuk melakukan backtracking klaster DB menggunakan API Amazon RDS, gunakan operasi BacktrackDBCluster. Operasi ini melakukan backtracking klaster DB yang ditentukan dalam nilai `DBClusterIdentifier` ke waktu yang ditentukan.

Memantau backtracking

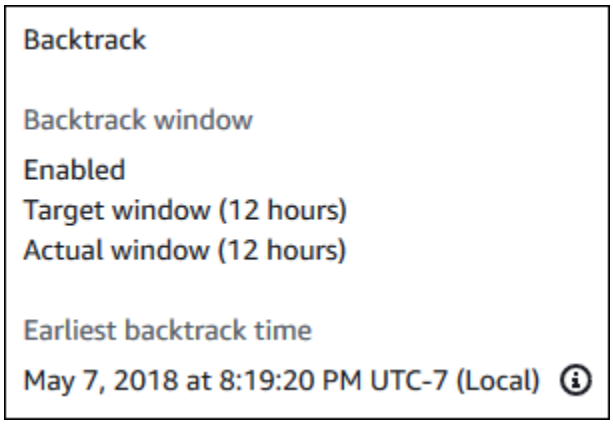
Anda dapat melihat informasi backtracking dan memantau metrik backtracking untuk klaster DB.

Konsol

Untuk melihat informasi backtracking dan memantau metrik backtracking menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih nama klaster DB untuk membuka informasi tentangnya.

   Informasi backtrack ada di bagian Backtrack.

   Saat backtrack diaktifkan, informasi berikut tersedia:

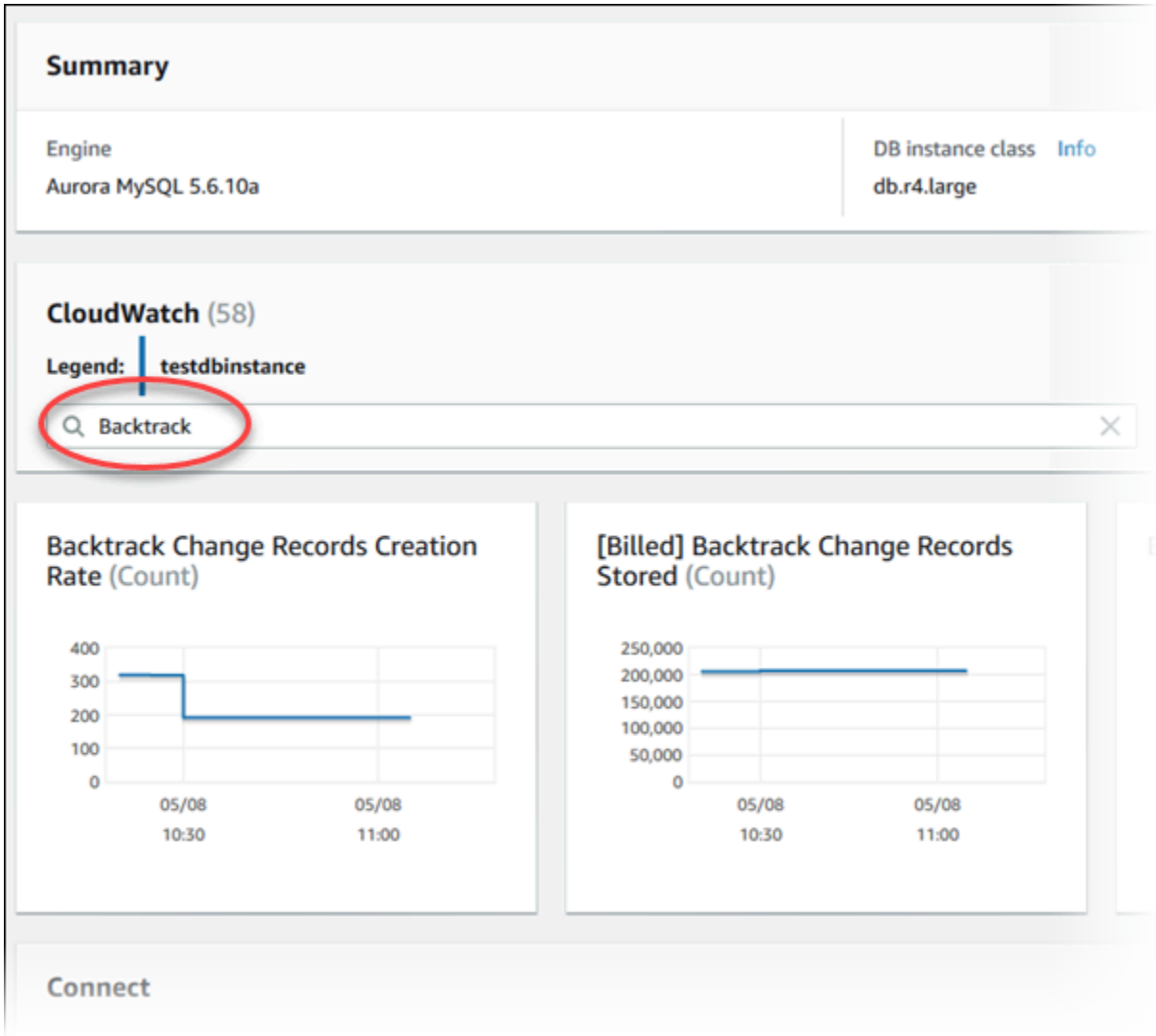
- Jendela target – Jumlah waktu saat ini yang ditentukan untuk jendela backtrack target. Target adalah jumlah waktu maksimum yang Anda inginkan untuk melakukan backtracking jika ada penyimpanan yang memadai.
- Jendela aktual – Jumlah waktu aktual yang Anda inginkan untuk dapat melakukan backtracking, yang dapat lebih kecil daripada jendela backtrack target. Jendela backtrack aktual didasarkan pada beban kerja Anda dan penyimpanan yang tersedia untuk mempertahankan catatan perubahan backtrack.
- Waktu backtrack paling awal – Waktu backtrack paling awal yang memungkinkan untuk klaster DB. Anda tidak dapat melakukan backtracking klaster DB ke waktu sebelum waktu yang ditampilkan.

4. Lakukan hal berikut untuk melihat metrik backtracking untuk klaster DB:

   a. Di panel navigasi, pilih Instans.

   b. Pilih nama instans primer untuk klaster DB untuk menampilkan detailnya.

   c. Di CloudWatchbagian ini, ketik **`Backtrack`** ke dalam CloudWatchkotak untuk hanya menampilkan metrik Backtrack.



Metrik berikut ditampilkan:

- Laju Pembuatan Catatan Perubahan Backtrack (Hitungan) – Metrik ini menunjukkan jumlah catatan perubahan backtrack yang dibuat selama lima menit untuk klaster DB Anda. Anda dapat menggunakan metrik ini untuk memperkirakan biaya backtrack untuk jendela backtrack target Anda.
- [Ditagih] Catatan Perubahan Backtrack Disimpan (Hitungan) – Metrik ini menunjukkan jumlah aktual RDS perubahan backtrack yang digunakan oleh klaster DB Anda.
- Jendela Backtrack Aktual (Menit) – Metrik ini menunjukkan apakah terdapat perbedaan antara jendela backtrack target dan jendela backtrack aktual. Sebagai contoh, jika jendela backtrack target Anda adalah 2 jam (120 menit), dan metrik ini menunjukkan bahwa jendela backtrack aktual adalah 100 menit, berarti jendela backtrack aktual lebih kecil dari target.

- Peringatan Jendela Backtrack (Hitungan) – Metrik ini menunjukkan jumlah berapa kali jendela backtrack aktual lebih kecil daripada jendela backtrack target untuk periode waktu tertentu.

> **Note**
>
> Metrik berikut mungkin mengalami lag dari waktu saat ini:
>
> - Laju Pembuatan Catatan Perubahan Backtrack (Hitungan)
> - [Ditagih] Catatan Perubahan Backtrack Disimpan (Hitungan)

AWS CLI

Prosedur berikut menjelaskan cara menampilkan informasi backtrack untuk klaster DB menggunakan AWS CLI.

Untuk melihat informasi backtrack untuk cluster DB menggunakan AWS CLI

- Panggil perintah describe-db-clusters AWS CLI dan berikan nilai-nilai berikut:
  - `--db-cluster-identifier` – Nama klaster DB.

  Contoh berikut mencantumkan informasi backtrack untuk `sample-cluster`.

  Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-clusters \
    --db-cluster-identifier sample-cluster
```

  Untuk Windows:

```
aws rds describe-db-clusters ^
    --db-cluster-identifier sample-cluster
```

API RDS

Untuk melihat informasi backtrack untuk klaster DB menggunakan API Amazon RDS, gunakan operasi DescribeDBClusters. Operasi ini menampilkan informasi backtrack untuk klaster DB yang ditentukan dalam nilai `DBClusterIdentifier`.

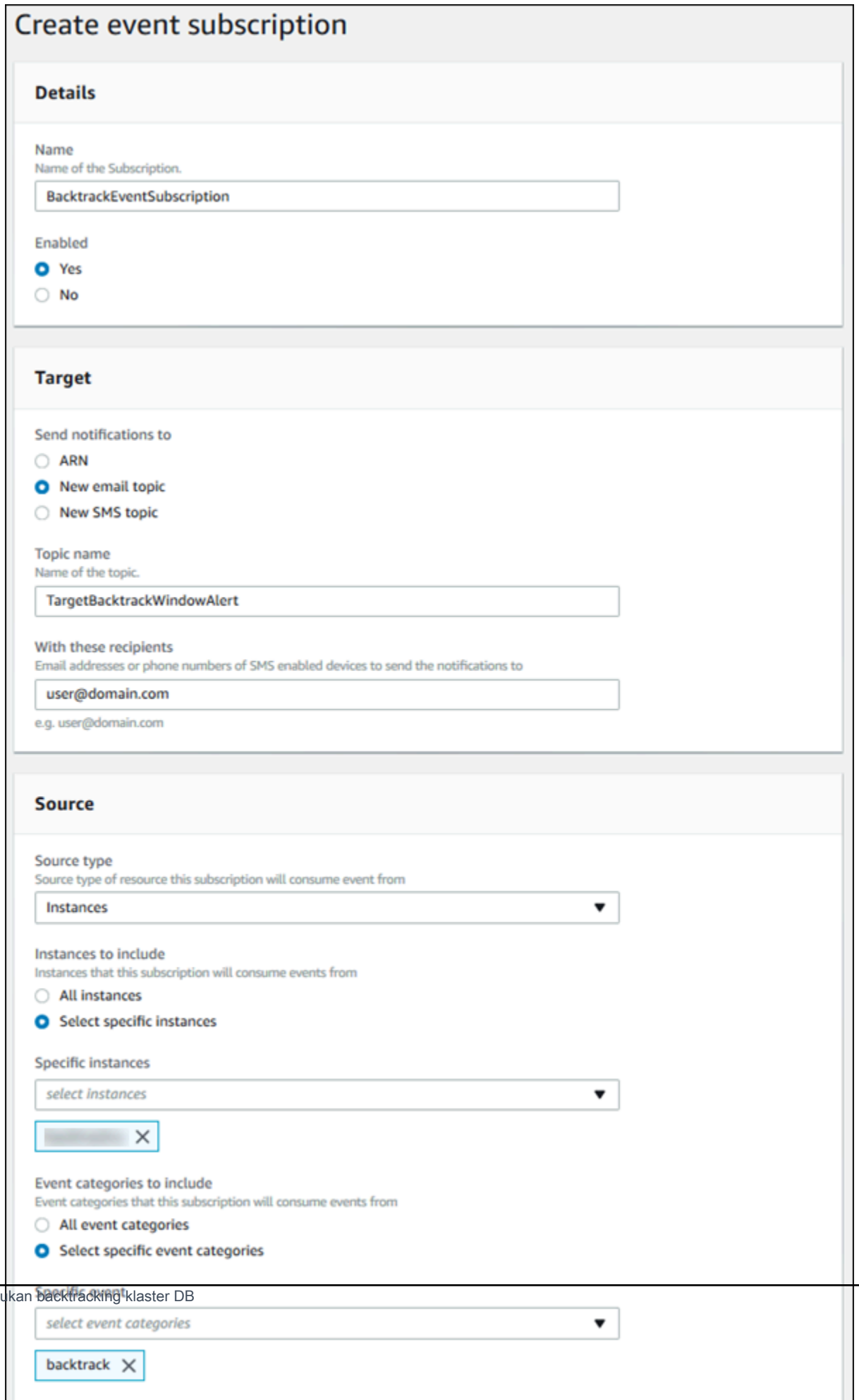
Berlangganan peristiwa backtrack dengan konsol

Prosedur berikut menjelaskan cara berlangganan peristiwa backtrack menggunakan konsol. Peristiwa tersebut mengirimkan notifikasi email atau pesan teks saat jendela backtrack aktual Anda lebih kecil dari jendela backtrack target Anda.

Untuk melihat informasi backtrack menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Langganan peristiwa.
3. Pilih Buat langganan peristiwa.
4. Di kotak Nama, ketik nama untuk langganan peristiwa, dan pastikan bahwa Ya dipilih untuk Diaktifkan.
5. Di bagian Target, pilih Topik email baru.
6. Untuk Nama topik, ketik nama untuk topik, dan untuk Dengan penerima ini, masukkan alamat email atau nomor telepon untuk menerima notifikasi.
7. Di bagian Sumber, pilih Instans untuk Jenis sumber.
8. Untuk Instans untuk disertakan, klik Pilih instans tertentu, dan pilih instans DB Anda.
9. Untuk Kategori peristiwa untuk disertakan, klik Pilih kategori peristiwa tertentu, dan pilih backtrack.

   Halaman Anda akan terlihat serupa dengan halaman berikut.


Create event subscription
Details
Name
Name of the Subscription.
BacktrackEventSubscription
Enabled
Yes
No
Target
Send notifications to
ARN
New email topic
New SMS topic
Topic name
Name of the topic.
TargetBacktrackWindowAlert
With these recipients
Email addresses or phone numbers of SMS enabled devices to send the notifications to
user@domain.com
e.g. user@domain.com
Source
Source type
Source type of resource this subscription will consume event from
Instances
Instances to include
Instances that this subscription will consume events from
All instances
Select specific instances
Specific instances
select instances
Event categories to include
Event categories that this subscription will consume events from
All event categories
Select specific event categories
select event categories
backtrack

10. Pilih Buat.

Mengambil backtrack yang ada

Anda dapat mengambil informasi tentang backtrack yang ada untuk klaster DB. Informasi ini mencakup pengidentifikasi unik backtrack, rentang tanggal dan waktu backtracking, tanggal dan waktu backtrack diminta, dan status backtrack saat ini.

> **Note**
>
> Saat ini, Anda tidak dapat mengambil backtrack yang ada menggunakan konsol.

AWS CLI

Prosedur berikut menjelaskan cara mengambil backtrack yang ada untuk klaster DB menggunakan AWS CLI.

Untuk mengambil backtrack yang ada menggunakan AWS CLI

- Panggil perintah describe-db-cluster-backtracks AWS CLI dan berikan nilai-nilai berikut:
  - `--db-cluster-identifier` – Nama klaster DB.

  Contoh berikut mengambil backtrack yang ada untuk `sample-cluster`.

  Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-cluster-backtracks \
    --db-cluster-identifier sample-cluster
```

  Untuk Windows:

```
aws rds describe-db-cluster-backtracks ^
    --db-cluster-identifier sample-cluster
```

API RDS

Untuk mengambil informasi tentang backtrack untuk cluster DB menggunakan Amazon RDS API, gunakan operasi DescribeDB. ClusterBacktracks Operasi ini menampilkan informasi tentang backtrack untuk klaster DB yang ditentukan dalam nilai `DBClusterIdentifier`.

Menonaktifkan backtracking untuk klaster DB

Anda dapat menonaktifkan fitur Backtrack untuk klaster DB.

Konsol

Anda dapat menonaktifkan backtracking untuk klaster DB menggunakan konsol. Setelah Anda menonaktifkan backtracking sepenuhnya untuk sebuah klaster, Anda tidak dapat mengaktifkannya lagi untuk klaster tersebut.

Untuk menonaktifkan fitur Backtrack untuk klaster DB menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.
3. Pilih klaster yang ingin Anda ubah, dan pilih Modifikasi.
4. Di bagian Backtrack, pilih Nonaktifkan Backtrack.
5. Pilih Lanjutkan.
6. Untuk Penjadwalan Modifikasi, pilih salah satu dari berikut ini:
   - Terapkan selama jendela pemeliharaan terjadwal berikutnya – Tunggu untuk menerapkan modifikasi hingga periode pemeliharaan berikutnya.
   - Terapkan segera – Terapkan modifikasi sesegera mungkin.
7. Pilih Ubah Klaster.

AWS CLI

Anda dapat menonaktifkan fitur Backtrack untuk cluster DB menggunakan AWS CLI dengan mengatur jendela target backtrack ke `0` (nol). Setelah Anda menonaktifkan backtracking sepenuhnya untuk sebuah klaster, Anda tidak dapat mengaktifkannya lagi untuk klaster tersebut.

Untuk memodifikasi jendela backtrack target untuk cluster DB menggunakan AWS CLI

- Panggil perintah modify-db-cluster AWS CLI dan berikan nilai-nilai berikut:
  - `--db-cluster-identifier` – Nama klaster DB.
  - `--backtrack-window` – tentukan `0` untuk menonaktifkan backtrack.

  Contoh berikut menonaktifkan fitur Backtrack untuk `sample-cluster` dengan mengatur `--backtrack-window` ke `0`.

  Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --backtrack-window 0
```

  Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --backtrack-window 0
```

API RDS

Untuk menonaktifkan fitur Backtrack untuk klaster DB menggunakan API Amazon RDS, gunakan operasi ModifyDBCluster. Atur nilai `BacktrackWindow` ke `0` (nol), dan tentukan klaster DB dalam nilai `DBClusterIdentifier`. Setelah Anda menonaktifkan backtracking sepenuhnya untuk sebuah klaster, Anda tidak dapat mengaktifkannya lagi untuk klaster tersebut.

Menguji Amazon Aurora MySQL menggunakan kueri injeksi kesalahan

Anda dapat menguji toleransi kesalahan klaster DB Aurora MySQL Anda menggunakan kueri injeksi kesalahan. Kueri injeksi kesalahan dikeluarkan sebagai perintah SQL ke instans Amazon Aurora. Kueri ini memungkinkan Anda menjadwalkan simulasi kemunculan salah satu peristiwa berikut:

- Crash instans DB penulis atau pembaca

- Kegagalan Replika Aurora
- Kegagalan disk
- Kepadatan disk

Ketika kueri injeksi kesalahan menentukan sebuah crash, kueri ini akan memaksa crash pada instans DB Aurora MySQL. Kueri injeksi kesalahan lainnya menghasilkan simulasi peristiwa kegagalan, tetapi tidak menyebabkan peristiwa terjadi. Jika Anda mengirim kueri injeksi kesalahan, Anda juga perlu menentukan durasi waktu untuk simulasi peristiwa kegagalan yang akan terjadi.

Anda dapat mengirim kueri injeksi kesalahan ke salah satu instans Replika Aurora dengan menghubungkan ke titik akhir untuk Replika Aurora tersebut. Untuk informasi selengkapnya, lihat Manajemen koneksi Amazon Aurora.

Untuk menjalankan kueri injeksi kesalahan, diperlukan semua hak akses pengguna master. Untuk informasi selengkapnya, lihat Hak akses akun pengguna master.

Menguji crash instans

Anda dapat memaksa crash instans Amazon Aurora menggunakan kueri injeksi kesalahan `ALTER SYSTEM CRASH`.

Untuk kueri injeksi kesalahan, failover tidak akan terjadi. Jika Anda ingin menguji failover, maka Anda dapat memilih tindakan instans Failover untuk klaster DB Anda di konsol RDS, atau gunakan perintah failover-db-cluster AWS CLI atau operasi API RDS FailoverDBCluster.

Sintaksis

```
ALTER SYSTEM CRASH [ INSTANCE | DISPATCHER | NODE ];
```

Opsi

Kueri injeksi kesalahan ini menggunakan salah satu dari jenis crash berikut:

- **`INSTANCE`** – Crash basis data yang kompatibel dengan MySQL untuk instans Amazon Aurora disimulasikan.
- **`DISPATCHER`** — Crash dispatcher pada instans penulis untuk klaster DB Aurora disimulasikan. Dispatcher menulis pembaruan ke volume klaster untuk klaster DB Amazon Aurora.
- **`NODE`** – Crash basis data yang kompatibel dengan MySQL dan dispatcher untuk instans Amazon Aurora disimulasikan. Untuk simulasi injeksi kesalahan ini, cache juga dihapus.

Jenis crash default adalah `INSTANCE`.

Menguji kegagalan replika Aurora

Anda dapat menyimulasikan kegagalan Replika Aurora menggunakan kueri injeksi kesalahan `ALTER SYSTEM SIMULATE READ REPLICA FAILURE`.

Kegagalan Replika Aurora akan memblokir semua permintaan dari instans penulis ke Replika Aurora atau semua Replika Aurora dalam klaster DB untuk interval waktu yang ditentukan. Ketika interval waktu selesai, Replika Aurora yang terpengaruh akan secara otomatis disinkronisasi dengan instans master.

Sintaksis

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT READ REPLICA FAILURE
    [ TO ALL | TO "replica name" ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

Opsi

Kueri injeksi kesalahan ini mengambil parameter berikut:

- **`percentage_of_failure`** – Persentase permintaan yang akan diblokir selama peristiwa kegagalan. Nilai ini dapat berupa double antara 0 dan 100. Jika Anda menetapkan 0, tidak ada permintaan yang akan diblokir. Jika Anda menentukan 100, semua permintaan akan diblokir.
- failure_type – Jenis kegagalan yang akan disimulasikan. Tentukan `TO ALL` untuk menyimulasikan kegagalan untuk semua Replika Aurora dalam klaster DB. Tentukan `TO` dan nama Replika Aurora untuk menyimulasikan kegagalan Replika Aurora tunggal. Jenis kegagalan default adalah `TO ALL`.
- **`quantity`** – Jumlah waktu untuk menyimulasikan kegagalan Replika Aurora. Interval adalah jumlah yang diikuti oleh satuan waktu. Simulasi akan terjadi untuk jumlah satuan yang ditentukan tersebut. Misalnya, `20 MINUTE` akan menghasilkan simulasi yang berjalan selama 20 menit.

> **Note**
>
> Berhati-hatilah saat menentukan interval waktu untuk peristiwa kegagalan Replika Aurora Anda. Jika Anda menentukan interval waktu yang terlalu panjang, dan instans penulis Anda menulis data dalam jumlah besar selama peristiwa kegagalan, maka klaster DB Aurora Anda mungkin menganggap bahwa Replika Aurora telah crash lalu menggantinya.

Menguji kegagalan disk

Anda dapat menyimulasikan kegagalan disk untuk klaster DB Aurora menggunakan kueri injeksi kesalahan `ALTER SYSTEM SIMULATE DISK FAILURE`.

Selama simulasi kegagalan disk, klaster DB Aurora secara acak menandai segmen disk sebagai mengalami kesalahan. Permintaan ke segmen tersebut akan diblokir selama durasi simulasi.

Sintaksis

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK FAILURE
    [ IN DISK index | NODE index ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

Opsi

Kueri injeksi kesalahan ini mengambil parameter berikut:

- **`percentage_of_failure`** – Persentase disk yang akan ditandai sebagai mengalami kesalahan selama peristiwa kegagalan. Nilai ini dapat berupa double antara 0 dan 100. Jika Anda menentukan 0, tidak ada disk yang akan ditandai sebagai mengalami kesalahan. Jika Anda menentukan 100, seluruh disk akan ditandai sebagai mengalami kesalahan.
- **`DISK index`** – Blok data logis data tertentu untuk menyimulasikan peristiwa kegagalan. Jika Anda melampaui rentang blok data logis yang tersedia, Anda akan menerima kesalahan yang memberi tahu Anda nilai indeks maksimum yang dapat Anda tentukan. Untuk informasi selengkapnya, lihat Menampilkan status volume untuk klaster DB Aurora MySQL.
- **`NODE index`** – Simpul penyimpanan spesifik untuk menyimulasikan peristiwa kegagalan. Jika Anda melebihi rentang simpul penyimpanan yang tersedia, Anda akan menerima kesalahan yang memberi tahu Anda nilai indeks maksimum yang dapat Anda tentukan. Untuk informasi selengkapnya, lihat Menampilkan status volume untuk klaster DB Aurora MySQL.
- **`quantity`** – Jumlah waktu untuk menyimulasikan kegagalan disk. Interval adalah jumlah yang diikuti oleh satuan waktu. Simulasi akan terjadi untuk jumlah satuan yang ditentukan tersebut. Misalnya, `20 MINUTE` akan menghasilkan simulasi yang berjalan selama 20 menit.

Menguji kepadatan disk

Anda dapat menyimulasikan kegagalan disk untuk klaster DB Aurora menggunakan kueri injeksi kesalahan `ALTER SYSTEM SIMULATE DISK CONGESTION`.

Selama simulasi kepadatan disk, klaster DB Aurora secara acak menandai segmen disk sebagai padat. Permintaan ke segmen tersebut akan ditunda antara waktu penundaan minimum dan maksimum yang ditentukan untuk durasi simulasi.

Sintaksis

```
ALTER SYSTEM SIMULATE percentage_of_failure PERCENT DISK CONGESTION
    BETWEEN minimum AND maximum MILLISECONDS
    [ IN DISK index | NODE index ]
    FOR INTERVAL quantity { YEAR | QUARTER | MONTH | WEEK | DAY | HOUR | MINUTE |
 SECOND };
```

Opsi

Kueri injeksi kesalahan ini mengambil parameter berikut:

- **`percentage_of_failure`** – Persentase disk yang akan ditandai sebagai padat selama peristiwa kegagalan. Nilai ini dapat berupa double antara 0 dan 100. Jika Anda menetapkan 0, tidak ada disk yang ditandai sebagai padat. Jika Anda menetapkan 100, seluruh disk akan ditandai sebagai padat.
- **`DISK index`** Atau **`NODE index`** – Disk atau simpul tertentu untuk menyimulasikan peristiwa kegagalan. Jika Anda melebihi rentang indeks untuk disk atau simpul, Anda akan menerima kesalahan yang memberi tahu Anda nilai indeks maksimum yang dapat Anda tentukan.
- **`minimum`** Dan **`maximum`** – Jumlah penundaan kepadatan minimum dan maksimum, dalam milidetik. Segmen disk yang ditandai sebagai padat akan ditunda selama jumlah waktu acak dalam rentang jumlah minimum dan maksimum milidetik untuk durasi simulasi.
- **`quantity`** – Jumlah waktu untuk menyimulasikan kepadatan disk. Interval adalah jumlah yang diikuti oleh satuan waktu. Simulasi akan terjadi untuk jumlah satuan waktu yang ditentukan. Misalnya, `20 MINUTE` akan menghasilkan simulasi yang berjalan selama 20 menit.

Mengubah tabel di Amazon Aurora menggunakan DDL Cepat

Amazon Aurora menyediakan pengoptimalan untuk menjalankan operasi `ALTER TABLE` di tempat, hampir seketika. Operasi ini akan selesai tanpa mengharuskan tabel disalin dan tanpa memiliki dampak besar pada pernyataan DML lainnya. Karena operasi ini tidak menggunakan penyimpanan sementara untuk salinan tabel, hal ini membuat pernyataan DDL praktis bahkan untuk tabel besar pada kelas instans kecil.

Aurora MySQL versi 3 kompatibel dengan fitur MySQL 8.0 yang disebut DDL instan. Aurora MySQL versi 2 menggunakan implementasi berbeda yang disebut DDL Cepat.

Topik

- DDL instan (Aurora MySQL versi 3)
- DDL Cepat (Aurora MySQL versi 2)

DDL instan (Aurora MySQL versi 3)

Optimisasi yang dilakukan oleh Aurora MySQL versi 3 untuk meningkatkan efisiensi beberapa operasi DDL disebut DDL instan.

Aurora MySQL versi 3 kompatibel dengan DDL instan dari MySQL 8.0 komunitas. Anda melakukan operasi DDL instan dengan menggunakan klausa `ALGORITHM=INSTANT` dengan pernyataan `ALTER TABLE`. Untuk detail sintaksis dan penggunaan DDL instan, lihat ALTER TABLE dan Online DDL Operations dalam dokumentasi MySQL.

Contoh berikut menunjukkan fitur DDL instan. Pernyataan `ALTER TABLE` menambahkan kolom dan mengubah nilai kolom default. Contoh ini mencakup kolom reguler dan virtual, dan tabel reguler dan berpartisi. Pada setiap langkah, Anda dapat melihat hasilnya dengan mengeluarkan pernyataan `SHOW CREATE TABLE` dan `DESCRIBE`.

```
mysql> CREATE TABLE t1 (a INT, b INT, KEY(b)) PARTITION BY KEY(b) PARTITIONS 6;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t1 RENAME TO t2, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b SET DEFAULT 100, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t2 ALTER COLUMN b DROP DEFAULT, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 ADD COLUMN c ENUM('a', 'b', 'c'), ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> ALTER TABLE t2 MODIFY COLUMN c ENUM('a', 'b', 'c', 'd', 'e'), ALGORITHM =
 INSTANT;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> ALTER TABLE t2 ADD COLUMN (d INT GENERATED ALWAYS AS (a + 1) VIRTUAL), ALGORITHM
 = INSTANT;
Query OK, 0 rows affected (0.02 sec)

mysql> ALTER TABLE t2 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t3 (a INT, b INT) PARTITION BY LIST(a)(
    ->   PARTITION mypart1 VALUES IN (1,3,5),
    ->   PARTITION MyPart2 VALUES IN (2,4,6)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE t3 ALTER COLUMN a SET DEFAULT 20, ALTER COLUMN b SET DEFAULT 200,
 ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t4 (a INT, b INT) PARTITION BY RANGE(a)
    ->   (PARTITION p0 VALUES LESS THAN(100), PARTITION p1 VALUES LESS THAN(1000),
    ->   PARTITION p2 VALUES LESS THAN MAXVALUE);
Query OK, 0 rows affected (0.05 sec)

mysql> ALTER TABLE t4 ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)

/* Sub-partitioning example */
mysql> CREATE TABLE ts (id INT, purchased DATE, a INT, b INT)
    ->   PARTITION BY RANGE( YEAR(purchased) )
    ->     SUBPARTITION BY HASH( TO_DAYS(purchased) )
    ->     SUBPARTITIONS 2 (
    ->       PARTITION p0 VALUES LESS THAN (1990),
    ->       PARTITION p1 VALUES LESS THAN (2000),
    ->       PARTITION p2 VALUES LESS THAN MAXVALUE
    ->    );
Query OK, 0 rows affected (0.10 sec)

mysql> ALTER TABLE ts ALTER COLUMN a SET DEFAULT 20,
    ->   ALTER COLUMN b SET DEFAULT 200, ALGORITHM = INSTANT;
Query OK, 0 rows affected (0.01 sec)
```

DDL Cepat (Aurora MySQL versi 2)

Di MySQL, banyak operasi bahasa definisi data (DDL) memiliki dampak performa yang signifikan.

Misalnya, anggaplah bahwa Anda menggunakan operasi `ALTER TABLE` untuk menambahkan kolom ke tabel. Bergantung pada algoritma yang ditentukan untuk operasi, operasi ini dapat memerlukan hal-hal berikut:

- Membuat salinan lengkap tabel
- Membuat tabel sementara untuk memproses operasi bahasa manipulasi data (DML) konkuren
- Membuat kembali semua indeks untuk tabel
- Menerapkan kunci tabel sambil menerapkan perubahan DML konkuren
- Memperlambat throughput DML konkuren

Optimisasi yang dilakukan oleh Aurora MySQL versi 2 untuk meningkatkan efisiensi beberapa operasi DDL disebut DDL Cepat.

Di Aurora MySQL versi 3, Aurora menggunakan fitur MySQL 8.0 yang disebut DDL instan. Aurora MySQL versi 2 menggunakan implementasi berbeda yang disebut DDL Cepat.

> **Important**
>
> Saat ini, mode lab Aurora harus diaktifkan untuk menggunakan DDL Cepat untuk Aurora MySQL. Kami tidak menyarankan penggunaan DDL Cepat untuk klaster DB produksi. Untuk informasi tentang mengaktifkan mode lab Aurora, lihat Mode lab Amazon Aurora MySQL.

Batasan DDL Cepat

Saat ini, DDL Cepat memiliki batasan berikut:

- DDL Cepat hanya mendukung penambahan kolom yang dapat dibuat null, tanpa nilai default, ke akhir tabel yang ada.
- DDL Cepat tidak berfungsi untuk tabel yang dipartisi.
- DDL Cepat tidak berfungsi untuk tabel InnoDB yang menggunakan format baris REDUNDANT.
- DDL Cepat tidak berfungsi untuk tabel dengan indeks pencarian teks lengkap.
- Jika ukuran catatan maksimum yang mungkin untuk operasi DDL terlalu besar, DDL Cepat tidak digunakan. Ukuran catatan terlalu besar jika lebih besar dari setengah ukuran halaman. Ukuran maksimum catatan dihitung dengan menjumlahkan ukuran maksimum semua kolom. Untuk kolom

dengan ukuran bervariasi, yang mengikuti standar InnoDB, byte extern tidak disertakan untuk perhitungan.

Sintaksis DDL Cepat

```
ALTER TABLE tbl_name ADD COLUMN col_name column_definition
```

Pernyataan ini mengambil opsi berikut:

- **`tbl_name`** – Nama tabel yang akan diubah.
- **`col_name`** – Nama kolom yang akan ditambahkan.
- **`col_definition`** – Definisi kolom yang akan ditambahkan.

> **Note**
>
> Anda harus menentukan kolom yang dapat dibuat null tanpa nilai default. Jika tidak, DDL Cepat tidak digunakan.

Contoh DDL Cepat

Contoh berikut menunjukkan percepatan dari operasi DDL Cepat. Contoh SQL pertama menjalankan pernyataan `ALTER TABLE` pada tabel besar tanpa menggunakan DDL Cepat. Operasi ini membutuhkan waktu yang cukup lama. Salah satu contoh CLI ini menunjukkan cara mengaktifkan DDL Cepat untuk klaster. Kemudian, contoh SQL lainnya menjalankan pernyataan `ALTER TABLE` yang sama pada tabel yang identik. Dengan mengaktifkan DDL Cepat, operasinya sangat cepat.

Contoh ini menggunakan tabel `ORDERS` dari tolok ukur TPC-H, yang berisi 150 juta baris. Klaster ini sengaja menggunakan kelas instans yang relatif kecil untuk menunjukkan waktu yang diperlukan pernyataan `ALTER TABLE` saat Anda tidak dapat menggunakan DDL Cepat. Contoh ini membuat klon dari tabel asli yang berisi data identik. Dengan memeriksa pengaturan `aurora_lab_mode`, akan dikonfirmasi bahwa klaster tidak dapat menggunakan DDL Cepat karena mode lab tidak diaktifkan. Kemudian, pernyataan `ALTER TABLE ADD COLUMN` membutuhkan banyak waktu untuk menambahkan kolom baru di akhir tabel.

```
mysql> create table orders_regular_ddl like orders;
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> insert into orders_regular_ddl select * from orders;
Query OK, 150000000 rows affected (1 hour 1 min 25.46 sec)

mysql> select @@aurora_lab_mode;
+------------------+
| @@aurora_lab_mode |
+------------------+
|                0 |
+------------------+

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (40 min 31.41 sec)

mysql> ALTER TABLE orders_regular_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (40 min 44.45 sec)
```

Contoh ini melakukan persiapan tabel besar yang sama seperti contoh sebelumnya. Namun, Anda tidak bisa begitu saja mengaktifkan mode lab dalam sesi SQL interaktif. Pengaturan tersebut harus diaktifkan dalam grup parameter kustom. Tindakan tersebut dapat dilakukan dengan beralih dari sesi `mysql` dan menjalankan beberapa perintah AWS CLI atau menggunakan AWS Management Console.

```
mysql> create table orders_fast_ddl like orders;
Query OK, 0 rows affected (0.02 sec)

mysql> insert into orders_fast_ddl select * from orders;
Query OK, 150000000 rows affected (58 min 3.25 sec)

mysql> set aurora_lab_mode=1;
ERROR 1238 (HY000): Variable 'aurora_lab_mode' is a read only variable
```

Pengaktifan mode lab untuk klaster memerlukan beberapa pekerjaan dengan grup parameter. Contoh AWS CLI ini menggunakan grup parameter klaster untuk memastikan bahwa semua instans DB di klaster menggunakan nilai yang sama untuk pengaturan mode lab.

```
$ aws rds create-db-cluster-parameter-group \
  --db-parameter-group-family aurora5.7 \
    --db-cluster-parameter-group-name lab-mode-enabled-57 --description 'TBD'
$ aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].[ParameterName,ParameterValue]' \
      --output text | grep aurora_lab_mode
```

```
aurora_lab_mode 0
$ aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --parameters ParameterName=aurora_lab_mode,ParameterValue=1,ApplyMethod=pending-
reboot
{
    "DBClusterParameterGroupName": "lab-mode-enabled-57"
}

# Assign the custom parameter group to the cluster that's going to use Fast DDL.
$ aws rds modify-db-cluster --db-cluster-identifier tpch100g \
  --db-cluster-parameter-group-name lab-mode-enabled-57
{
  "DBClusterIdentifier": "tpch100g",
  "DBClusterParameterGroup": "lab-mode-enabled-57",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.2",
  "Status": "available"
}

# Reboot the primary instance for the cluster tpch100g:
$ aws rds reboot-db-instance --db-instance-identifier instance-2020-12-22-5208
{
  "DBInstanceIdentifier": "instance-2020-12-22-5208",
  "DBInstanceStatus": "rebooting"
}

$ aws rds describe-db-clusters --db-cluster-identifier tpch100g \
  --query '*[].[DBClusterParameterGroup]' --output text
lab-mode-enabled-57

$ aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name lab-mode-enabled-57 \
    --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
      --output text | grep aurora_lab_mode
aurora_lab_mode 1
```

Contoh berikut menunjukkan langkah-langkah yang tersisa setelah perubahan grup parameter berlaku. Contoh ini menguji pengaturan `aurora_lab_mode` untuk memastikan bahwa klaster dapat menggunakan DDL Cepat. Contoh ini kemudian menjalankan pernyataan `ALTER TABLE` untuk menambahkan kolom ke akhir tabel besar lainnya. Kali ini, pernyataan selesai dengan sangat cepat.

```
mysql> select @@aurora_lab_mode;
```

```
+-------------------+
| @@aurora_lab_mode |
+-------------------+
|                 1 |
+-------------------+

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_refunded boolean;
Query OK, 0 rows affected (1.51 sec)

mysql> ALTER TABLE orders_fast_ddl ADD COLUMN o_coverletter varchar(512);
Query OK, 0 rows affected (0.40 sec)
```

Menampilkan status volume untuk klaster DB Aurora MySQL

Di Amazon Aurora, volume klaster DB terdiri dari sekumpulan blok logis. Masing-masing blok merepresentasikan 10 gigabyte penyimpanan yang dialokasikan. Blok-blok ini disebut grup perlindungan.

Data di setiap grup perlindungan direplikasi di enam perangkat penyimpanan fisik, yang disebut simpul penyimpanan. Simpul penyimpanan ini dialokasikan di tiga Zona Ketersediaan (AZ) di Wilayah AWS tempat klaster DB berada. Pada gilirannya, setiap simpul penyimpanan berisi satu atau beberapa blok logis data untuk volume klaster DB. Untuk informasi selengkapnya tentang grup perlindungan dan simpul penyimpanan, lihat Memperkenalkan mesin penyimpanan Aurora dalam Blog Basis Data AWS.

Anda dapat menyimulasikan kegagalan seluruh simpul penyimpanan atau satu blok logis data dalam simpul penyimpanan. Untuk melakukannya, Anda menggunakan pernyataan injeksi kegagalan `ALTER SYSTEM SIMULATE DISK FAILURE`. Untuk pernyataan tersebut, Anda menentukan nilai indeks blok logis data atau simpul penyimpanan tertentu. Namun, jika Anda menentukan nilai indeks yang lebih besar dari jumlah blok logis simpul data atau penyimpanan yang digunakan oleh volume klaster DB, pernyataan tersebut akan menampilkan kesalahan. Untuk informasi selengkapnya tentang kueri injeksi kesalahan, lihat Menguji Amazon Aurora MySQL menggunakan kueri injeksi kesalahan.

Anda dapat menghindari kesalahan tersebut dengan menggunakan pernyataan `SHOW VOLUME STATUS`. Pernyataan ini menghasilkan dua variabel status server, `Disks` dan `Nodes`. Variabel ini merepresentasikan jumlah total blok logis simpul data dan penyimpanan, masing-masing, untuk volume klaster DB.

Sintaksis

```
SHOW VOLUME STATUS
```

Contoh

Contoh berikut menggambarkan hasil SHOW VOLUME STATUS yang biasa.

```
mysql> SHOW VOLUME STATUS;
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Disks         | 96    |
| Nodes         | 74    |
+---------------+-------+
```

Menyesuaikan Aurora MySQL

Peristiwa tunggu dan status thread adalah alat penyesuaian penting untuk Aurora MySQL. Jika Anda dapat mengetahui alasan mengapa sesi menunggu sumber daya dan operasi yang sedang dilakukannya, Anda akan lebih mampu mengurangi bottleneck. Anda dapat menggunakan informasi di bagian ini untuk menemukan kemungkinan penyebab dan tindakan korektif.

Amazon DevOps Guru untuk RDS dapat secara proaktif menentukan apakah basis data Aurora MySQL Anda mengalami kondisi bermasalah yang dapat menyebabkan masalah yang lebih besar nantinya. Amazon DevOps Guru for RDS menerbitkan penjelasan dan rekomendasi untuk tindakan korektif dalam wawasan proaktif. Bagian ini berisi wawasan untuk masalah umum.

> **Important**
>
> Peristiwa tunggu dan status thread di bagian ini dikhususkan untuk Aurora MySQL. Gunakan informasi di bagian ini untuk menyesuaikan Amazon Aurora saja, bukan Amazon RDS for MySQL.
> Beberapa peristiwa tunggu di bagian ini tidak memiliki padanan di versi sumber terbuka dari mesin basis data ini. Peristiwa tunggu lainnya memiliki nama yang sama dengan peristiwa di mesin sumber terbuka, tetapi berperilaku berbeda. Misalnya, penyimpanan Amazon Aurora berfungsi secara berbeda dari penyimpanan sumber terbuka, sehingga peristiwa tunggu terkait penyimpanan menunjukkan kondisi sumber daya yang berbeda.

Topik

- Konsep penting untuk penyesuaian Aurora MySQL
- Menyesuaikan Aurora MySQL dengan peristiwa tunggu
- Menyesuaikan Aurora MySQL dengan status thread
- Menyesuaikan Aurora MySQL dengan wawasan proaktif Amazon DevOps Guru

Konsep penting untuk penyesuaian Aurora MySQL

Sebelum Anda menyesuaikan basis data Aurora MySQL Anda, pastikan untuk mempelajari peristiwa tunggu dan status thread dan mengapa itu terjadi. Tinjau juga arsitektur memori dan disk dasar Aurora MySQL saat menggunakan mesin penyimpanan InnoDB. Untuk diagram arsitektur yang bermanfaat, lihat Panduan Referensi MySQL.

Topik

- Peristiwa tunggu Aurora MySQL
- Status thread Aurora MySQL
- Memori Aurora MySQL
- Proses Aurora MySQL

Peristiwa tunggu Aurora MySQL

Peristiwa tunggu menunjukkan sumber daya yang sedang ditunggu oleh sesi. Misalnya, peristiwa tunggu `io/socket/sql/client_connection` menunjukkan bahwa thread sedang dalam proses untuk menangani koneksi baru. Sumber daya biasa yang ditunggu oleh sesi meliputi yang berikut:

- Akses thread tunggal ke buffer, misalnya, saat sesi mencoba memodifikasi buffer
- Baris yang saat ini dikunci oleh sesi lain
- Pembacaan file data
- Penulisan file data

Misalnya, untuk memenuhi kueri, sesi mungkin melakukan pemindaian tabel lengkap. Jika data belum ada dalam memori, sesi akan menunggu I/O disk selesai. Ketika buffer dibaca ke dalam memori, sesi mungkin perlu menunggu karena sesi lain mengakses buffer yang sama. Basis data mencatat peristiwa tunggu dengan menggunakan peristiwa tunggu standar. Peristiwa tersebut dikelompokkan ke dalam kategori.

Peristiwa tunggu tidak dengan sendirinya menunjukkan masalah performa. Misalnya, jika data yang diminta tidak ada dalam memori, data perlu dibaca dari disk. Jika satu sesi mengunci baris untuk pembaruan, sesi lain akan menunggu baris tersebut dibuka sehingga dapat memperbaruinya. Commit perlu menunggu penulisan ke file log selesai. Peristiwa tunggu merupakan bagian integral dari fungsi normal basis data.

Sejumlah besar peristiwa tunggu biasanya menunjukkan masalah performa. Dalam kasus seperti itu, Anda dapat menggunakan data peristiwa tunggu untuk menentukan tempat sesi menghabiskan waktu. Misalnya, jika laporan yang biasanya berjalan dalam hitungan menit sekarang berjalan selama berjam-jam, Anda dapat mengidentifikasi peristiwa tunggu yang berkontribusi paling besar terhadap total waktu tunggu. Jika Anda dapat menentukan penyebab peristiwa tunggu teratas, terkadang Anda dapat membuat perubahan yang meningkatkan performa. Misalnya, jika sesi Anda menunggu baris yang telah dikunci oleh sesi lain, Anda dapat mengakhiri sesi penguncian ini.

Status thread Aurora MySQL

Status thread umum adalah nilai `State` yang terkait dengan pemrosesan kueri umum. Misalnya, status thread `sending data` menunjukkan bahwa thread membaca dan memfilter baris untuk kueri guna menentukan set hasil yang benar.

Anda dapat menggunakan status thread untuk menyesuaikan Aurora MySQL dengan cara yang mirip dengan cara Anda menggunakan peristiwa tunggu. Misalnya, kemunculan `sending data` yang sering biasanya menunjukkan bahwa kueri tidak menggunakan indeks. Untuk informasi selengkapnya tentang status thread, lihat General Thread States dalam Panduan Referensi MySQL.

Saat Anda menggunakan Wawasan Performa, salah satu kondisi berikut ini akan berlaku:

- Skema Performa diaktifkan – Aurora MySQL menunjukkan peristiwa tunggu, bukan status thread.
- Skema Performa tidak diaktifkan – Aurora MySQL menunjukkan status thread.

Sebaiknya konfigurasikan Skema Performa untuk manajemen otomatis. Skema Performa memberikan wawasan tambahan dan alat yang lebih baik untuk menyelidiki potensi masalah performa. Untuk informasi selengkapnya, lihat Mengaktifkan Skema Performa untuk Wawasan Performa di Aurora MySQL.

Memori Aurora MySQL

Di Aurora MySQL, area memori yang paling penting adalah pool buffer dan log buffer.

Topik

- Pool buffer

Pool buffer

Pool buffer adalah area memori bersama tempat Aurora MySQL menyimpan tabel dan data indeks. Kueri dapat mengakses data yang sering digunakan langsung dari memori tanpa membaca dari disk.

Pool buffer disusun sebagai daftar halaman yang ditautkan. Halaman dapat menyimpan beberapa baris. Aurora MySQL menggunakan algoritma least recently used (LRU) untuk mengeluarkan halaman dari pool berdasarkan umurnya.

Untuk informasi selengkapnya, lihat Buffer Pool dalam Panduan Referensi MySQL.

Proses Aurora MySQL

Aurora MySQL menggunakan model proses yang sangat berbeda dari Aurora PostgreSQL.

Topik

- Server MySQL (mysqld)
- Thread
- Pool thread

Server MySQL (mysqld)

Server MySQL adalah proses sistem operasi tunggal bernama mysqld. Server MySQL tidak memunculkan proses tambahan. Dengan demikian, basis data Aurora MySQL menggunakan mysqld untuk melakukan sebagian besar pekerjaannya.

Ketika server MySQL dimulai, server tersebut akan mendengarkan koneksi jaringan dari klien MySQL. Ketika klien terhubung ke basis data, mysqld membuka thread.

Thread

Thread pengelola koneksi mengaitkan setiap koneksi klien dengan thread khusus. Thread ini mengelola autentikasi, menjalankan pernyataan, dan memberikan hasil ke klien. Pengelola koneksi membuat thread baru jika diperlukan.

Cache thread adalah kumpulan thread yang tersedia. Ketika koneksi berakhir, MySQL mengembalikan thread ke cache thread jika cache tidak penuh. Variabel sistem `thread_cache_size` menentukan ukuran cache thread.

Pool thread

Pool thread terdiri dari sejumlah grup thread. Setiap grup mengelola satu set koneksi klien. Ketika klien terhubung ke basis data, pool thread menetapkan koneksi ke grup thread dengan cara round-robin. Pool thread memisahkan koneksi dan thread. Tidak ada relasi tetap antara koneksi dan thread yang menjalankan pernyataan yang diterima dari koneksi tersebut.

Menyesuaikan Aurora MySQL dengan peristiwa tunggu

Tabel berikut merangkum peristiwa tunggu Aurora MySQL yang paling sering menunjukkan masalah performa. Peristiwa tunggu berikut adalah bagian dari daftar dalam Peristiwa tunggu Aurora MySQL.

Peristiwa tunggu	Deskripsi
cpu	Peristiwa ini terjadi ketika thread aktif di CPU atau sedang menunggu CPU.
io/aurora_redo_log_flush	Peristiwa ini terjadi ketika sesi akan menulis data yang persisten ke penyimpanan Aurora.
io/aurora_respond_to_client	Peristiwa ini terjadi ketika thread menunggu untuk memberikan set hasil ke klien.
io/redo_log_flush	Peristiwa ini terjadi ketika sesi akan menulis data yang persisten ke penyimpanan Aurora.
io/socket/sql/client_connection	Peristiwa ini terjadi ketika thread sedang dalam proses untuk menangani koneksi baru.
io/table/sql/handler	Peristiwa ini terjadi ketika pekerjaan telah didelegasikan ke mesin penyimpanan.
synch/cond/innodb/row_lock_wait	Peristiwa ini terjadi ketika satu sesi telah mengunci baris untuk pembaruan, dan sesi lain mencoba memperbarui baris yang sama.
synch/cond/innodb/row_lock_wait_cond	Peristiwa ini terjadi ketika satu sesi telah mengunci baris untuk pembaruan, dan sesi lain mencoba memperbarui baris yang sama.
synch/cond/sql/MDL_context::COND_wait_status	Peristiwa ini terjadi ketika ada thread yang menunggu kunci metadata tabel.
synch/mutex/innodb/aurora_lock_thread_slot_futex	Peristiwa ini terjadi ketika satu sesi telah mengunci baris untuk pembaruan, dan sesi lain mencoba memperbarui baris yang sama.
synch/mutex/innodb/buf_pool_mutex	Peristiwa ini terjadi ketika thread telah mendapat kunci pada pool buffer InnoDB untuk mengakses halaman dalam memori.

Peristiwa tunggu	Deskripsi
synch/mutex/innodb/fil_system_mutex	Peristiwa ini terjadi ketika sesi menunggu untuk mengakses cache memori ruang tabel.
synch/mutex/innodb/trx_sys_mutex	Peristiwa ini terjadi ketika ada aktivitas basis data tinggi dengan sejumlah besar transaksi.
synch/sxlock/innodb/hash_table_locks	Peristiwa ini terjadi ketika halaman yang tidak ditemukan di pool buffer harus dibaca dari file.

cpu

Peristiwa tunggu `cpu` terjadi saat thread aktif di CPU atau sedang menunggu CPU.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora MySQL versi 2 dan 3

Konteks

Untuk setiap vCPU, koneksi dapat berjalan bekerja pada CPU ini. Dalam situasi tertentu, jumlah koneksi aktif yang siap dijalankan lebih tinggi daripada jumlah vCPU. Ketidakseimbangan ini mengakibatkan koneksi menunggu sumber daya CPU. Jika jumlah koneksi aktif secara konsisten tetap lebih tinggi daripada jumlah vCPU, maka instans akan mengalami pertentangan CPU. Pertentangan tersebut menyebabkan peristiwa tunggu `cpu` terjadi.

> **Note**
>
> Metrik Wawasan Performa untuk CPU adalah `DBLoadCPU`. Nilai untuk `DBLoadCPU` dapat berbeda dari nilai untuk CloudWatch metrik`CPUUtilization`. Metrik terakhir dikumpulkan dari HyperVisor untuk instance database.

Metrik OS Wawasan Performa memberikan informasi terperinci tentang pemanfaatan CPU. Misalnya, Anda dapat menampilkan metrik berikut:

- `os.cpuUtilization.nice.avg`
- `os.cpuUtilization.total.avg`
- `os.cpuUtilization.wait.avg`
- `os.cpuUtilization.idle.avg`

Wawasan Performa melaporkan penggunaan CPU oleh mesin basis data sebagai `os.cpuUtilization.nice.avg`.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa ini terjadi lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

- Kueri analitik
- Transaksi bersamaan yang tinggi
- Transaksi yang berjalan lama
- Peningkatan mendadak dalam jumlah koneksi, yang dikenal sebagai login storm
- Peningkatan peralihan konteks

Tindakan

Jika peristiwa tunggu `cpu` mendominasi aktivitas basis data, hal tersebut tidak selalu menunjukkan adanya masalah performa. Tanggapi peristiwa ini hanya saat performa menurun.

Tergantung pada penyebab peningkatan pemanfaatan CPU, pertimbangkan strategi berikut:

- Tingkatkan kapasitas CPU host. Pendekatan ini biasanya hanya memberikan bantuan sementara.

- Identifikasi kueri teratas untuk pengoptimalan potensial.
- Arahkan ulang sebagian beban kerja hanya-baca ke simpul pembaca jika berlaku.

Topik

- Mengidentifikasi sesi atau kueri yang menyebabkan masalah
- Menganalisis dan mengoptimalkan beban kerja CPU yang tinggi

Mengidentifikasi sesi atau kueri yang menyebabkan masalah

Untuk menemukan sesi dan kueri tersebut, lihat tabel SQL Teratas dalam Wawasan Performa untuk pernyataan SQL yang memiliki beban CPU tertinggi. Untuk informasi selengkapnya, lihat Menganalisis metrik dengan dasbor Wawasan Performa.

Biasanya, satu atau dua pernyataan SQL mengonsumsi sebagian besar siklus CPU. Konsentrasikan upaya Anda pada pernyataan ini. Misalnya, instans DB Anda memiliki 2 vCPU dengan beban DB 3,1 sesi aktif rata-rata (AAS), yang semuanya dalam status CPU. Dalam hal ini, instans Anda terikat dengan CPU. Pertimbangkan strategi berikut:

- Lakukan peningkatan ke kelas instans yang lebih besar dengan lebih banyak vCPU.
- Sesuaikan kueri Anda untuk memiliki beban CPU yang lebih rendah.

Dalam contoh ini, kueri SQL teratas memiliki beban DB 1,5 AAS, yang semuanya dalam status CPU. Pernyataan SQL lain memiliki beban 0,1 dalam status CPU. Dalam contoh ini, jika menghentikan pernyataan SQL dengan beban terendah, Anda tidak akan mengurangi beban basis data secara signifikan. Namun, jika mengoptimalkan dua kueri beban tinggi menjadi dua kali lebih efisien, Anda menghilangkan kemacetan CPU. Jika Anda mengurangi beban CPU 1,5 AAS sebesar 50%, AAS untuk setiap pernyataan akan berkurang menjadi 0,75. Total beban DB yang dihabiskan untuk CPU sekarang adalah 1,6 AAS. Nilai ini berada di bawah baris vCPU maksimum 2.0.

Untuk ikhtisar pemecahan masalah yang berguna menggunakan Wawasan Performa, lihat postingan blog Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa. Lihat juga artikel Dukungan AWS, Bagaimana cara memecahkan masalah dan mengatasi pemanfaatan CPU yang tinggi pada instans Amazon RDS for MySQL saya?.

Menganalisis dan mengoptimalkan beban kerja CPU yang tinggi

Setelah mengidentifikasi kueri atau sejumlah kueri yang meningkatkan penggunaan CPU, Anda dapat mengoptimalkannya atau mengakhiri koneksi. Contoh berikut menunjukkan cara mengakhiri koneksi.

```
CALL mysql.rds_kill(processID);
```

Untuk informasi selengkapnya, lihat mysql.rds_kill.

Jika Anda mengakhiri sesi, tindakan tersebut dapat memicu rollback yang panjang.

Mengikuti panduan untuk mengoptimalkan kueri

Untuk mengoptimalkan kueri, pertimbangkan panduan berikut:

- Jalankan pernyataan `EXPLAIN`.

  Perintah ini menunjukkan setiap langkah yang terlibat dalam menjalankan kueri. Untuk informasi selengkapnya, lihat Mengoptimalkan Kueri dengan EXPLAIN dalam dokumentasi MySQL.
- Jalankan pernyataan `SHOW PROFILE`.

  Gunakan pernyataan ini untuk meninjau detail profil yang dapat menunjukkan penggunaan sumber daya untuk pernyataan yang dijalankan selama sesi saat ini. Untuk informasi selengkapnya, lihat Pernyataan SHOW PROFILE dalam dokumentasi MySQL.
- Jalankan pernyataan `ANALYZE TABLE`.

  Gunakan pernyataan ini untuk menyegarkan statistik indeks untuk tabel yang diakses oleh kueri dengan konsumsi CPU tinggi. Dengan menganalisis pernyataan, Anda dapat membantu pengoptimal memilih rencana eksekusi yang sesuai. Untuk informasi selengkapnya, lihat Pernyataan ANALYZE TABLE dalam dokumentasi MySQL.

Mengikuti panduan untuk meningkatkan penggunaan CPU

Untuk meningkatkan penggunaan CPU dalam instans basis data, ikuti panduan berikut:

- Pastikan semua kueri telah menggunakan indeks yang tepat.
- Cari tahu apakah Anda dapat menggunakan kueri paralel Aurora. Anda dapat menggunakan teknik ini untuk mengurangi penggunaan CPU pada simpul kepala dengan menekan pemrosesan fungsi, pemfilteran baris, dan proyeksi kolom untuk klausa `WHERE`.

- Cari tahu apakah jumlah eksekusi SQL per detik memenuhi ambang batas yang diharapkan.
- Cari tahu apakah pemeliharaan indeks atau pembuatan indeks baru membutuhkan siklus CPU yang diperlukan oleh beban kerja produksi Anda. Jadwalkan aktivitas pemeliharaan di luar waktu aktivitas puncak.
- Cari tahu apakah Anda dapat menggunakan partisi untuk membantu mengurangi set data kueri. Untuk informasi selengkapnya, lihat postingan blog Cara merencanakan dan mengoptimalkan Amazon Aurora dengan kompatibilitas MySQL untuk beban kerja terkonsolidasi.

Memeriksa storm koneksi

Jika metrik `DBLoadCPU` tidak terlalu tinggi, tetapi metrik `CPUUtilization` tinggi, penyebab pemanfaatan CPU yang tinggi berasal dari luar mesin basis data. Contoh klasik adalah storm koneksi.

Periksa apakah kondisi berikut benar:

- Ada peningkatan dalam metrik Performance Insights dan `CPUUtilization` metrik Amazon CloudWatch`DatabaseConnections`.
- Jumlah thread dalam CPU lebih besar dari jumlah vCPU.

Jika kondisi di atas benar, coba kurangi jumlah koneksi basis data. Misalnya, Anda dapat menggunakan kumpulan koneksi, seperti Proksi RDS. Untuk mempelajari praktik terbaik penskalaan dan manajemen koneksi yang efektif, lihat laporan resmi Buku Panduan DBA Amazon Aurora MySQL untuk Manajemen Koneksi.

io/aurora_redo_log_flush

Peristiwa `io/aurora_redo_log_flush` terjadi ketika sesi menulis data persisten ke penyimpanan Amazon Aurora.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora MySQL versi 2

Konteks

Peristiwa `io/aurora_redo_log_flush` ditujukan untuk operasi input/output (I/O) tulis di Aurora MySQL.

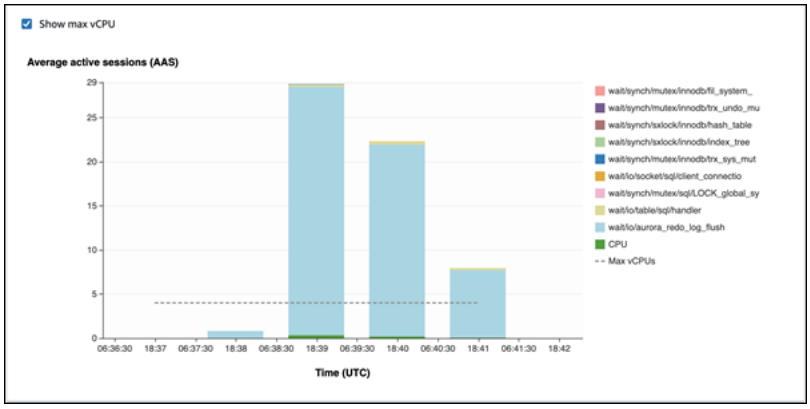
> **Note**
>
> Di Aurora MySQL versi 3, peristiwa tunggu ini disebut io/redo_log_flush.

Kemungkinan penyebab peningkatan peristiwa tunggu

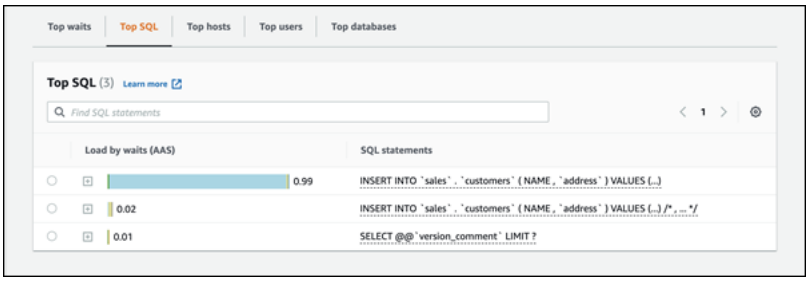
Untuk persistensi data, commit memerlukan penulisan tahan lama ke penyimpanan yang stabil. Jika basis data terlalu banyak melakukan commit, terdapat peristiwa tunggu pada operasi I/O tulis, yakni peristiwa tunggu `io/aurora_redo_log_flush`.

Dalam contoh berikut, sebanyak 50.000 catatan dimasukkan ke dalam klaster DB Aurora MySQL menggunakan kelas instans DB db.r5.xlarge:

- Pada contoh pertama, setiap sesi menyisipkan 10.000 catatan baris demi baris. Secara default, jika perintah bahasa manipulasi data (DML) tidak berada dalam transaksi, Aurora MySQL akan menggunakan commit implisit. Autocommit diaktifkan. Ini berarti terdapat commit untuk setiap penyisipan baris. Wawasan Performa menunjukkan bahwa koneksi menghabiskan sebagian besar waktunya menunggu peristiwa tunggu `io/aurora_redo_log_flush`.

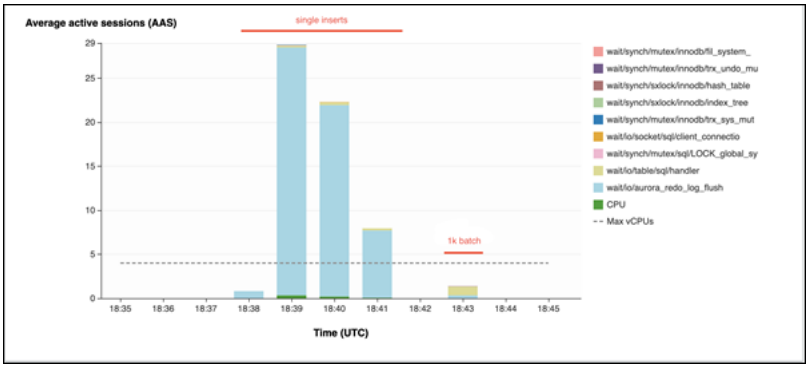


Hal ini disebabkan oleh pernyataan sisipan sederhana yang digunakan.



Diperlukan waktu 3,5 menit untuk menyisipkan 50.000 catatan.

- Dalam contoh kedua, sisipan dibuat dalam 1.000 batch, yakni setiap koneksi melakukan 10 commit, bukan 10.000. Wawasan Performa menunjukkan bahwa koneksi tidak menghabiskan sebagian besar waktunya untuk peristiwa tunggu `io/aurora_redo_log_flush`.



Diperlukan waktu 4 detik untuk menyisipkan 50.000 catatan.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Mengidentifikasi sesi dan kueri yang bermasalah

Jika instans DB Anda mengalami kemacetan, langkah pertama adalah menemukan sesi dan kueri yang menjadi penyebabnya. Untuk postingan Blog Basis Data AWS yang berguna, lihat Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa.

Untuk mengidentifikasi sesi dan kueri yang menyebabkan kemacetan:

1. Masuk ke AWS Management Console, lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.

3. Pilih instans DB Anda.
4. Di Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih SQL Teratas.

   Kueri di bagian atas daftar adalah penyebab beban tertinggi pada basis data.

Mengelompokkan operasi tulis

Contoh berikut memicu peristiwa tunggu `io/aurora_redo_log_flush`. (Autocommit diaktifkan.)

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

Dalam upaya mengurangi waktu yang dihabiskan untuk menunggu peristiwa tunggu `io/aurora_redo_log_flush`, kelompokkan operasi tulis secara logis ke dalam satu commit guna mengurangi panggilan persisten ke penyimpanan.

Menonaktifkan autocommit

Nonaktifkan autocommit sebelum membuat perubahan besar yang tidak ada dalam transaksi, seperti yang ditunjukkan pada contoh berikut.

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
```

```
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
COMMIT;

SET SESSION AUTOCOMMIT=ON;
```

Menggunakan transaksi

Anda dapat menggunakan transaksi, seperti yang ditunjukkan pada contoh berikut.

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;

-- Other DML statements here
END
```

Menggunakan batch

Anda dapat membuat perubahan dalam batch, seperti yang ditunjukkan pada contoh berikut. Namun, menggunakan batch yang terlalu besar dapat menyebabkan masalah kinerja, terutama pada replika baca atau saat melakukan point-in-time pemulihan (PITR).

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

io/aurora_respond_to_client

Peristiwa `io/aurora_respond_to_client` terjadi ketika thread menunggu untuk mengembalikan set hasil ke klien.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora MySQL versi 2

Dalam versi sebelum 2.07.7, 2.09.3, dan 2.10.2, peristiwa tunggu ini secara keliru menyertakan waktu idle.

Konteks

Peristiwa `io/aurora_respond_to_client` menunjukkan bahwa thread menunggu untuk mengembalikan set hasil ke klien.

Pemrosesan kueri selesai, dan hasilnya dikembalikan ke klien aplikasi. Namun, karena bandwidth jaringan pada klaster DB tidak cukup, thread menunggu untuk mengembalikan set hasil.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `io/aurora_respond_to_client` muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

Kelas instans DB tidak cukup untuk beban kerja

> Kelas instans DB yang digunakan oleh klaster DB tidak memiliki bandwidth jaringan yang diperlukan untuk memproses beban kerja secara efisien.

Set hasil besar

Terjadi peningkatan ukuran set hasil yang dikembalikan karena kueri mengembalikan jumlah baris yang lebih tinggi. Set hasil yang lebih besar menghabiskan lebih banyak bandwidth jaringan.

Peningkatan beban pada klien

Mungkin terdapat tekanan CPU, tekanan memori, atau saturasi jaringan pada klien. Peningkatan beban pada klien menunda penerimaan data dari klaster DB Aurora MySQL.

Peningkatan latensi jaringan

Mungkin terdapat peningkatan latensi jaringan antara klaster DB Aurora MySQL dan klien. Latensi jaringan yang lebih tinggi akan menambah waktu yang diperlukan klien untuk menerima data.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengidentifikasi sesi dan kueri penyebab peristiwa
- Menskalakan kelas instans DB
- Memeriksa beban kerja untuk hasil yang tidak terduga
- Mendistribusikan beban kerja dengan instans pembaca
- Menggunakan pengubah SQL_BUFFER_RESULT

Mengidentifikasi sesi dan kueri penyebab peristiwa

Anda dapat menggunakan Wawasan Performa untuk menampilkan kueri yang diblokir oleh peristiwa tunggu `io/aurora_respond_to_client`. Biasanya, basis data dengan beban sedang hingga signifikan memiliki peristiwa tunggu. Peristiwa tunggu ini mungkin dapat diterima jika basis data berperforma optimal. Jika tidak, periksa di mana basis data tersebut menghabiskan waktu terbanyak. Lihat peristiwa tunggu yang berkontribusi ke beban tertinggi, lalu cari tahu apakah Anda dapat mengoptimalkan basis data dan aplikasi untuk mengurangi peristiwa tersebut.

Untuk menemukan kueri SQL yang bertanggung jawab atas beban tinggi:

1. Masuk ke AWS Management Console, lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa ditampilkan untuk instans DB tersebut.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih SQL Teratas.

   Bagan ini mencantumkan kueri SQL yang bertanggung jawab atas beban. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar pemecahan masalah yang berguna menggunakan Wawasan Performa, lihat postingan Blog Basis Data AWS, Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa.

Menskalakan kelas instans DB

Periksa peningkatan nilai metrik Amazon CloudWatch yang terkait dengan throughput jaringan, seperti `NetworkReceiveThroughput` dan `NetworkTransmitThroughput`. Jika bandwidth jaringan kelas instans DB tercapai, Anda dapat menskalakan kelas instans DB yang digunakan oleh klaster DB dengan memodifikasi klaster DB tersebut. Kelas instans DB dengan bandwidth jaringan yang lebih besar mengembalikan data ke klien secara lebih efisien.

Untuk informasi tentang pemantauan metrik Amazon CloudWatch, lihat Melihat metrik di konsol Amazon RDS. Untuk informasi tentang kelas instans DB, lihat Kelas instans DB Aurora. Untuk informasi tentang mengubah klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Memeriksa beban kerja untuk hasil yang tidak terduga

Periksa beban kerja pada klaster DB dan pastikan beban kerja tidak memberikan hasil yang tidak terduga. Misalnya, mungkin terdapat kueri yang mengembalikan jumlah baris lebih tinggi dari yang diperkirakan. Dalam hal ini, Anda dapat menggunakan metrik penghitung Wawasan Performa seperti `Innodb_rows_read`. Untuk informasi selengkapnya, lihat Metrik penghitung Wawasan Performa.

Mendistribusikan beban kerja dengan instans pembaca

Anda dapat mendistribusikan beban kerja hanya-baca dengan Aurora Replicas. Anda dapat menskalakan secara horizontal dengan menambahkan lebih banyak Aurora Replicas. Dengan demikian, Anda dapat memperoleh peningkatan batas throttling untuk bandwidth jaringan. Untuk informasi selengkapnya, lihat Klaster DB Amazon Aurora.

Menggunakan pengubah SQL_BUFFER_RESULT

Anda dapat menambahkan pengubah `SQL_BUFFER_RESULT` ke pernyataan `SELECT` untuk memaksakan hasilnya ke tabel sementara sebelum dikembalikan ke klien. Pengubah ini dapat membantu mengatasi masalah performa saat kunci InnoDB tidak dibebaskan karena kueri berada dalam status tunggu `io/aurora_respond_to_client`. Untuk informasi selengkapnya, lihat Pernyataan SELECT dalam dokumentasi MySQL.

io/redo_log_flush

Peristiwa `io/redo_log_flush` terjadi ketika sesi menulis data persisten ke penyimpanan Amazon Aurora.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora MySQL versi 3

Konteks

Peristiwa `io/redo_log_flush` ditujukan untuk operasi input/output (I/O) tulis di Aurora MySQL.

> **Note**
>
> Di Aurora MySQL versi 2, peristiwa tunggu ini disebut io/aurora_redo_log_flush.

Kemungkinan penyebab peningkatan peristiwa tunggu

Untuk persistensi data, commit memerlukan penulisan tahan lama ke penyimpanan yang stabil. Jika basis data terlalu banyak melakukan commit, terdapat peristiwa tunggu pada operasi I/O tulis, yakni peristiwa tunggu `io/redo_log_flush`.

Untuk contoh perilaku peristiwa tunggu ini, lihat io/aurora_redo_log_flush.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Mengidentifikasi sesi dan kueri yang bermasalah

Jika instans DB Anda mengalami kemacetan, langkah pertama adalah menemukan sesi dan kueri yang menjadi penyebabnya. Untuk postingan Blog Basis Data AWS yang berguna, lihat Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa.

Untuk mengidentifikasi sesi dan kueri yang menyebabkan kemacetan:

1. Masuk ke AWS Management Console, lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB Anda.
4. Di Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih SQL Teratas.

   Kueri di bagian atas daftar adalah penyebab beban tertinggi pada basis data.

Mengelompokkan operasi tulis

Contoh berikut memicu peristiwa tunggu `io/redo_log_flush`. (Autocommit diaktifkan.)

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE id=xx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

```
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

Dalam upaya mengurangi waktu yang dihabiskan untuk menunggu peristiwa tunggu `io/redo_log_flush`, kelompokkan operasi tulis secara logis ke dalam satu commit guna mengurangi panggilan persisten ke penyimpanan.

Menonaktifkan autocommit

Nonaktifkan autocommit sebelum membuat perubahan besar yang tidak ada dalam transaksi, seperti yang ditunjukkan pada contoh berikut.

```
SET SESSION AUTOCOMMIT=OFF;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
....
UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1=xx;
-- Other DML statements here
COMMIT;

SET SESSION AUTOCOMMIT=ON;
```

Menggunakan transaksi

Anda dapat menggunakan transaksi, seperti yang ditunjukkan pada contoh berikut.

```
BEGIN
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');
....
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES ('xxxx','xxxxx');

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
....
DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1=xx;
```

```
-- Other DML statements here
END
```

Menggunakan batch

Anda dapat membuat perubahan dalam batch, seperti yang ditunjukkan pada contoh berikut. Namun, menggunakan batch yang terlalu besar dapat menyebabkan masalah kinerja, terutama pada replika baca atau saat melakukan point-in-time pemulihan (PITR).

```
INSERT INTO `sampleDB`.`sampleTable` (sampleCol2, sampleCol3) VALUES
('xxxx','xxxxx'),('xxxx','xxxxx'),...,('xxxx','xxxxx'),('xxxx','xxxxx');

UPDATE `sampleDB`.`sampleTable` SET sampleCol3='xxxxx' WHERE sampleCol1 BETWEEN xx AND
 xxx;

DELETE FROM `sampleDB`.`sampleTable` WHERE sampleCol1<xx;
```

io/socket/sql/client_connection

Peristiwa `io/socket/sql/client_connection` terjadi saat thread sedang dalam proses penanganan koneksi baru.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora MySQL versi 2 dan 3

Konteks

Peristiwa `io/socket/sql/client_connection` menunjukkan bahwa mysqld sibuk membuat thread untuk menangani koneksi klien baru yang masuk. Dalam skenario ini, pemrosesan layanan

permintaan koneksi klien baru melambat, sementara koneksi menunggu agar thread ditetapkan. Untuk informasi selengkapnya, lihat Server MySQL (mysqld).

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa ini muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

- Terdapat peningkatan mendadak dalam koneksi pengguna baru dari aplikasi ke instans Amazon RDS Anda.
- Instans DB Anda tidak dapat memproses koneksi baru karena jaringan, CPU, atau memori sedang dibatasi.

Tindakan

Jika `io/socket/sql/client_connection` mendominasi aktivitas basis data, hal tersebut tidak selalu menunjukkan adanya masalah performa. Dalam basis data yang tidak idle, peristiwa tunggu selalu berada di atas. Lakukan tindakan hanya bila performa menurun. Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengidentifikasi sesi dan kueri yang bermasalah
- Mengikuti praktik terbaik untuk manajemen koneksi
- Menskalakan instans jika sumber daya sedang dibatasi
- Memeriksa host teratas dan pengguna teratas
- Membuat kueri tabel performance_schema
- Memeriksa status thread kueri
- Mengaudit permintaan dan kueri
- Menggabungkan koneksi basis data

Mengidentifikasi sesi dan kueri yang bermasalah

Jika instans DB Anda mengalami kemacetan, langkah pertama adalah menemukan sesi dan kueri yang menjadi penyebabnya. Untuk postingan blog yang berguna, lihat Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa.

Untuk mengidentifikasi sesi dan kueri yang menyebabkan kemacetan:

1. Masuk ke AWS Management Console, lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB Anda.
4. Di Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih SQL Teratas.

   Kueri di bagian atas daftar adalah penyebab beban tertinggi pada basis data.

Mengikuti praktik terbaik untuk manajemen koneksi

Untuk mengelola koneksi Anda, pertimbangkan strategi berikut:

- Gunakan penggabungan koneksi.

  Anda dapat secara bertahap meningkatkan jumlah koneksi sesuai yang diperlukan. Untuk informasi selengkapnya, lihat laporan resmi Buku Panduan Administrator Basis Data Amazon Aurora MySQL.
- Gunakan simpul pembaca untuk mendistribusikan kembali lalu lintas hanya-baca.

  Lihat informasi yang lebih lengkap di Replika Aurora dan Manajemen koneksi Amazon Aurora.

Menskalakan instans jika sumber daya sedang dibatasi

Cari contoh throttling dalam sumber daya berikut:

- CPU

  Periksa CloudWatch metrik Amazon Anda untuk penggunaan CPU yang tinggi.
- Jaringan

  Periksa peningkatan nilai CloudWatch metrik `network receive throughput` dan`network transmit throughput`. Jika instans Anda telah mencapai batas bandwidth jaringan untuk kelas instans, pertimbangkan untuk menskalakan instans RDS ke jenis kelas instans yang lebih tinggi. Untuk informasi selengkapnya, lihat Kelas instans DB Aurora.
- Memori yang dapat dibebaskan

Periksa penurunan CloudWatch metrik`FreeableMemory`. Pertimbangkan juga untuk mengaktifkan Peningkatan Pemantauan. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan.

Memeriksa host teratas dan pengguna teratas

Gunakan Wawasan Performa untuk memeriksa host teratas dan pengguna teratas. Untuk informasi selengkapnya, lihat Menganalisis metrik dengan dasbor Wawasan Performa.

Membuat kueri tabel performance_schema

Untuk mendapatkan jumlah koneksi total dan saat ini yang akurat, buat kueri tabel `performance_schema`. Dengan teknik ini, Anda mengidentifikasi pengguna sumber atau host yang bertanggung jawab untuk membuat koneksi dalam jumlah besar. Misalnya, buat kueri tabel `performance_schema` sebagai berikut:

```
SELECT * FROM performance_schema.accounts;
SELECT * FROM performance_schema.users;
SELECT * FROM performance_schema.hosts;
```

Memeriksa status thread kueri

Jika masalah performa terus berlanjut, periksa status thread kueri. Di klien `mysql`, keluarkan perintah berikut.

```
show processlist;
```

Mengaudit permintaan dan kueri

Untuk memeriksa sifat permintaan dan kueri dari akun pengguna, gunakan AuroraAurora MySQL Advanced Audit. Untuk mempelajari cara mengaktifkan audit, lihat Menggunakan Audit Lanjutan dengan klaster DB Amazon Aurora MySQL.

Menggabungkan koneksi basis data

Pertimbangkan untuk menggunakan Proksi Amazon RDS untuk manajemen koneksi. Dengan menggunakan Proksi RDS, Anda dapat mengizinkan aplikasi untuk menggabungkan dan berbagi koneksi basis data guna meningkatkan kemampuan penskalaannya. Proksi RDS membuat aplikasi lebih tangguh terhadap kegagalan basis data dengan secara otomatis terhubung ke sebuah instans

DB siaga sekaligus menjaga koneksi aplikasi. Untuk informasi selengkapnya, lihat Menggunakan Proksi Amazon RDS untuk Aurora.

io/table/sql/handler

Peristiwa `io/table/sql/handler` terjadi ketika pekerjaan telah didelegasikan ke mesin penyimpanan.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora MySQL versi 3: 3.01.0 dan 3.01.1
- Aurora MySQL versi 2

Konteks

Peristiwa `io/table` menunjukkan peristiwa tunggu untuk akses ke tabel. Peristiwa ini terjadi terlepas dari apakah data di-cache di pool buffer atau diakses pada disk. Peristiwa `io/table/sql/handler` menunjukkan peningkatan aktivitas beban kerja.

Handler adalah rutinitas yang berspesialisasi dalam jenis data tertentu atau berfokus pada tugas khusus tertentu. Misalnya, handler peristiwa menerima dan mencerna peristiwa serta sinyal dari sistem operasi atau antarmuka pengguna. Handler memori melakukan tugas-tugas yang berkaitan dengan memori. Handler input file adalah fungsi yang menerima input file dan melakukan tugas khusus pada data, sesuai dengan konteksnya.

Tampilan seperti `performance_schema.events_waits_current` sering kali menunjukkan `io/table/sql/handler` ketika peristiwa tunggu sebenarnya adalah peristiwa tunggu bersarang seperti penguncian. Jika peristiwa tunggu sebenarnya bukan `io/table/sql/handler`, Wawasan Performa akan melaporkan peristiwa tunggu bersarang. Saat melaporkan `io/table/sql/handler`, Wawasan Performa mewakili pemrosesan InnoDB atas permintaan I/O, bukan peristiwa

tunggu bersarang yang tersembunyi. Untuk informasi selengkapnya, lihat Peristiwa Atom dan Molekul Skema Performa dalam Panduan Referensi MySQL.

> **Note**
>
> Namun, di Aurora MySQL versi 3.01.0 dan 3.01.1, synch/mutex/innodb/aurora_lock_thread_slot_futex dilaporkan sebagai `io/table/sql/handler`.

Peristiwa `io/table/sql/handler` sering kali muncul dalam peristiwa tunggu teratas dengan peristiwa tunggu I/O seperti `io/aurora_redo_log_flush` dan `io/file/innodb/innodb_data_file`.

Kemungkinan penyebab peningkatan peristiwa tunggu

Dalam Wawasan Performa, lonjakan mendadak dalam peristiwa `io/table/sql/handler` menunjukkan adanya peningkatan aktivitas beban kerja. Peningkatan aktivitas berarti peningkatan I/O.

Wawasan Performa memfilter ID peristiwa bersarang dan tidak melaporkan peristiwa tunggu `io/table/sql/handler` saat peristiwa bersarang yang mendasarinya adalah peristiwa tunggu penguncian. Misalnya, jika peristiwa akar penyebabnya adalah synch/mutex/innodb/aurora_lock_thread_slot_futex, Wawasan Performa akan menampilkan peristiwa tunggu ini dalam peristiwa tunggu teratas, bukan `io/table/sql/handler`.

Dalam tampilan seperti `performance_schema.events_waits_current`, peristiwa tunggu untuk `io/table/sql/handler` sering kali muncul ketika peristiwa tunggu sebenarnya adalah peristiwa tunggu bersarang seperti penguncian. Jika peristiwa tunggu sebenarnya berbeda dengan `io/table/sql/handler`, Wawasan Performa akan mencari peristiwa tunggu bersarang dan melaporkan peristiwa tunggu sebenarnya, bukan `io/table/sql/handler`. Saat Wawasan Performa melaporkan `io/table/sql/handler`, peristiwa tunggu sebenarnya adalah `io/table/sql/handler`, bukan peristiwa tunggu bersarang yang tersembunyi. Untuk informasi selengkapnya, lihat Peristiwa Atom dan Molekul Skema Performa dalam Panduan Referensi MySQL 5.7.

> **Note**
>
> Namun, di Aurora MySQL versi 3.01.0 dan 3.01.1, synch/mutex/innodb/aurora_lock_thread_slot_futex dilaporkan sebagai `io/table/sql/handler`.

Tindakan

Jika peristiwa tunggu ini mendominasi aktivitas basis data, hal tersebut tidak selalu menunjukkan adanya masalah performa. Peristiwa tunggu selalu berada di atas saat basis data aktif. Anda perlu melakukan tindakan hanya bila performa menurun.

Kami merekomendasikan berbagai tindakan, tergantung pada peristiwa tunggu lain yang Anda lihat.

Topik

- Mengidentifikasi sesi dan kueri penyebab peristiwa
- Memeriksa korelasi dengan metrik penghitung Wawasan Performa
- Memeriksa peristiwa tunggu berkorelasi lainnya

Mengidentifikasi sesi dan kueri penyebab peristiwa

Biasanya, basis data dengan beban sedang hingga signifikan memiliki peristiwa tunggu. Peristiwa tunggu ini mungkin dapat diterima jika basis data berperforma optimal. Jika tidak, periksa di mana basis data tersebut menghabiskan waktu terbanyak. Lihat peristiwa tunggu yang berkontribusi ke beban tertinggi, lalu cari tahu apakah Anda dapat mengoptimalkan basis data dan aplikasi untuk mengurangi peristiwa tersebut.

Untuk menemukan kueri SQL yang bertanggung jawab atas beban tinggi:

1. Masuk ke AWS Management Console, lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa ditampilkan untuk instans DB tersebut.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih SQL Teratas.

   Bagan ini mencantumkan kueri SQL yang bertanggung jawab atas beban. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar pemecahan masalah yang berguna menggunakan Wawasan Performa, lihat postingan blog Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa.

Memeriksa korelasi dengan metrik penghitung Wawasan Performa

Periksa metrik penghitung Wawasan Performa seperti `Innodb_rows_changed`. Jika metrik penghitung berkorelasi dengan `io/table/sql/handler`, ikuti langkah-langkah berikut:

1. Dalam Wawasan Performa, cari pernyataan SQL yang memperhitungkan peristiwa tunggu teratas `io/table/sql/handler`. Jika memungkinkan, optimalkan pernyataan ini sehingga mengembalikan lebih sedikit baris.
2. Ambil tabel teratas dari tampilan `schema_table_statistics` dan `x $schema_table_statistics`. Tampilan ini menunjukkan jumlah waktu yang dihabiskan per tabel. Untuk informasi selengkapnya, lihat Tampilan schema_table_statistics dan x $schema_table_statistics dalam Panduan Referensi MySQL.

   Secara default, baris diurutkan berdasarkan total waktu tunggu menurun. Tabel dengan pertentangan terbanyak ditampilkan terlebih dahulu. Output menunjukkan apakah waktu dihabiskan untuk membaca, menulis, mengambil, menyisipkan, memperbarui, atau menghapus. Contoh berikut dijalankan pada instans Aurora MySQL 2.09.1.

```
mysql> select * from sys.schema_table_statistics limit 1\G

*************************** 1. row ***************************
     table_schema: read_only_db
       table_name: sbtest41
    total_latency: 54.11 m
     rows_fetched: 6001557
    fetch_latency: 39.14 m
    rows_inserted: 14833
   insert_latency: 5.78 m
     rows_updated: 30470
   update_latency: 5.39 m
     rows_deleted: 14833
   delete_latency: 3.81 m
 io_read_requests: NULL
          io_read: NULL
  io_read_latency: NULL
io_write_requests: NULL
         io_write: NULL
 io_write_latency: NULL
 io_misc_requests: NULL
  io_misc_latency: NULL
1 row in set (0.11 sec)
```

Memeriksa peristiwa tunggu berkorelasi lainnya

Jika `synch/sxlock/innodb/btr_search_latch` dan `io/table/sql/handler` bersama-sama berkontribusi paling besar terhadap anomali beban DB, periksa apakah variabel `innodb_adaptive_hash_index` diaktifkan. Jika ya, coba tingkatkan nilai parameter `innodb_adaptive_hash_index_parts`.

Jika Adaptive Hash Index dinonaktifkan, coba untuk mengaktifkannya. Untuk mempelajari selengkapnya tentang Adaptive Hash Index MySQL, lihat sumber daya berikut:

- Artikel Is Adaptive Hash Index in InnoDB right for my workload? di situs web Percona
- Adaptive Hash Index dalam Panduan Referensi MySQL
- Artikel Contention in MySQL InnoDB: Useful Info From the Semaphores Section di situs web Percona

> **Note**
>
> Adaptive Hash Index tidak didukung pada instans DB pembaca Aurora.
> Dalam kasus tertentu, performa mungkin buruk pada instans pembaca saat `synch/sxlock/innodb/btr_search_latch` dan `io/table/sql/handler` bersifat dominan. Jika demikian, coba alihkan beban kerja sementara ke instans DB penulis dan aktifkan Adaptive Hash Index.

synch/cond/innodb/row_lock_wait

Peristiwa `synch/cond/innodb/row_lock_wait` terjadi ketika satu sesi telah mengunci baris untuk pembaruan dan sesi lain mencoba memperbarui baris yang sama. Untuk informasi selengkapnya, lihat Penguncian InnoDB dalam Referensi MySQL.

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora MySQL versi 3: 3.02.0, 3.02.1, 3.02.2

Kemungkinan penyebab peningkatan peristiwa tunggu

Beberapa pernyataan bahasa manipulasi data (DML) mengakses baris atau sejumlah baris yang sama secara serentak.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada peristiwa tunggu lain yang Anda lihat.

Topik

- Menemukan dan merespons pernyataan SQL yang bertanggung jawab atas peristiwa tunggu ini
- Menemukan dan merespons sesi pemblokiran

Menemukan dan merespons pernyataan SQL yang bertanggung jawab atas peristiwa tunggu ini

Gunakan Wawasan Performa untuk mengidentifikasi pernyataan SQL yang bertanggung jawab atas peristiwa tunggu ini. Pertimbangkan strategi berikut:

- Jika kunci baris merupakan masalah yang terus-menerus, coba tulis ulang aplikasi untuk menggunakan penguncian optimis.
- Gunakan pernyataan multibaris.
- Bagikan beban kerja ke objek basis data yang berbeda-beda. Anda dapat melakukannya melalui partisi.
- Periksa nilai parameter `innodb_lock_wait_timeout`. Parameter ini mengontrol durasi transaksi menunggu sebelum menghasilkan kesalahan batas waktu.

Untuk ikhtisar pemecahan masalah yang berguna menggunakan Wawasan Performa, lihat postingan blog Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa.

Menemukan dan merespons sesi pemblokiran

Cari tahu apakah sesi pemblokiran idle atau aktif. Cari tahu juga apakah sesi tersebut berasal dari aplikasi atau pengguna aktif.

Untuk mengidentifikasi sesi yang memegang kunci, Anda dapat menjalankan `SHOW ENGINE INNODB STATUS`. Contoh berikut menunjukkan output sampel.

```
mysql> SHOW ENGINE INNODB STATUS;
```

```
---TRANSACTION 1688153, ACTIVE 82 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 2 row lock(s)
MySQL thread id 4244, OS thread handle 70369524330224, query id 4020834 172.31.14.179
 reinvent executing
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 24 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 11 page no 4 n bits 72 index GEN_CLUST_INDEX of table test.t1 trx
 id 1688153 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

Sebagai alternatif, Anda dapat menggunakan kueri berikut untuk mengekstrak detail tentang kunci saat ini.

```
mysql> SELECT p1.id waiting_thread,
    p1.user waiting_user,
    p1.host waiting_host,
    it1.trx_query waiting_query,
    ilw.requesting_engine_transaction_id waiting_transaction,
    ilw.blocking_engine_lock_id blocking_lock,
    il.lock_mode blocking_mode,
    il.lock_type blocking_type,
    ilw.blocking_engine_transaction_id blocking_transaction,
    CASE it.trx_state
        WHEN 'LOCK WAIT'
        THEN it.trx_state
        ELSE p.state end blocker_state,
    concat(il.object_schema,'.', il.object_name) as locked_table,
    it.trx_mysql_thread_id blocker_thread,
    p.user blocker_user,
    p.host blocker_host
FROM performance_schema.data_lock_waits ilw
JOIN performance_schema.data_locks il
ON ilw.blocking_engine_lock_id = il.engine_lock_id
AND ilw.blocking_engine_transaction_id = il.engine_transaction_id
JOIN information_schema.innodb_trx it
ON ilw.blocking_engine_transaction_id = it.trx_id join information_schema.processlist p
ON it.trx_mysql_thread_id = p.id join information_schema.innodb_trx it1
ON ilw.requesting_engine_transaction_id = it1.trx_id join
 information_schema.processlist p1
ON it1.trx_mysql_thread_id = p1.id\G
```

```
*************************** 1. row ***************************
waiting_thread: 4244
waiting_user: reinvent
waiting_host: 123.456.789.012:18158
waiting_query: select id1 from test.t1 where id1=1 for update
waiting_transaction: 1688153
blocking_lock: 70369562074216:11:4:2:70369549808672
blocking_mode: X
blocking_type: RECORD
blocking_transaction: 1688142
blocker_state: User sleep
locked_table: test.t1
blocker_thread: 4243
blocker_user: reinvent
blocker_host: 123.456.789.012:18156
1 row in set (0.00 sec)
```

Saat mengidentifikasi sesi, opsi Anda mencakup yang berikut:

- Menghubungi pemilik aplikasi atau pengguna.
- Jika sesi pemblokiran idle, coba akhiri sesi pemblokiran. Tindakan ini dapat memicu rollback yang panjang. Untuk mempelajari cara mengakhiri sesi, lihat Mengakhiri sesi atau kueri.

Untuk informasi selengkapnya tentang cara mengidentifikasi transaksi pemblokiran, lihat Menggunakan Informasi Penguncian dan Transaksi InnoDB dalam Panduan Referensi MySQL.

synch/cond/innodb/row_lock_wait_cond

Peristiwa `synch/cond/innodb/row_lock_wait_cond` terjadi ketika satu sesi telah mengunci baris untuk pembaruan dan sesi lain mencoba memperbarui baris yang sama. Untuk informasi selengkapnya, lihat Penguncian InnoDB dalam Referensi MySQL.

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora MySQL versi 2

Kemungkinan penyebab peningkatan peristiwa tunggu

Beberapa pernyataan bahasa manipulasi data (DML) mengakses baris atau sejumlah baris yang sama secara serentak.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada peristiwa tunggu lain yang Anda lihat.

Topik

- Menemukan dan merespons pernyataan SQL yang bertanggung jawab atas peristiwa tunggu ini
- Menemukan dan merespons sesi pemblokiran

Menemukan dan merespons pernyataan SQL yang bertanggung jawab atas peristiwa tunggu ini

Gunakan Wawasan Performa untuk mengidentifikasi pernyataan SQL yang bertanggung jawab atas peristiwa tunggu ini. Pertimbangkan strategi berikut:

- Jika kunci baris merupakan masalah yang terus-menerus, coba tulis ulang aplikasi untuk menggunakan penguncian optimis.
- Gunakan pernyataan multibaris.
- Bagikan beban kerja ke objek basis data yang berbeda-beda. Anda dapat melakukannya melalui partisi.
- Periksa nilai parameter `innodb_lock_wait_timeout`. Parameter ini mengontrol durasi transaksi menunggu sebelum menghasilkan kesalahan batas waktu.

Untuk ikhtisar pemecahan masalah yang berguna menggunakan Wawasan Performa, lihat postingan blog Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa.

Menemukan dan merespons sesi pemblokiran

Cari tahu apakah sesi pemblokiran idle atau aktif. Cari tahu juga apakah sesi tersebut berasal dari aplikasi atau pengguna aktif.

Untuk mengidentifikasi sesi yang memegang kunci, Anda dapat menjalankan `SHOW ENGINE INNODB STATUS`. Contoh berikut menunjukkan output sampel.

```
mysql> SHOW ENGINE INNODB STATUS;
```

```
---TRANSACTION 2771110, ACTIVE 112 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 24, OS thread handle 70369573642160, query id 13271336 172.31.14.179
 reinvent Sending data
select id1 from test.t1 where id1=1 for update
------- TRX HAS BEEN WAITING 43 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 11 page no 3 n bits 0 index GEN_CLUST_INDEX of table test.t1 trx
 id 2771110 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 5; compact format; info bits 0
```

Sebagai alternatif, Anda dapat menggunakan kueri berikut untuk mengekstrak detail tentang kunci saat ini.

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
              p1.host waiting_host,
              it1.trx_query waiting_query,
              ilw.requesting_trx_id waiting_transaction,
              ilw.blocking_lock_id blocking_lock,
              il.lock_mode blocking_mode,
              il.lock_type blocking_type,
              ilw.blocking_trx_id blocking_transaction,
              CASE it.trx_state
                WHEN 'LOCK WAIT'
                THEN it.trx_state
                ELSE p.state
              END blocker_state,
              il.lock_table locked_table,
              it.trx_mysql_thread_id blocker_thread,
              p.user blocker_user,
              p.host blocker_host
       FROM information_schema.innodb_lock_waits ilw
       JOIN information_schema.innodb_locks il
         ON ilw.blocking_lock_id = il.lock_id
        AND ilw.blocking_trx_id = il.lock_trx_id
       JOIN information_schema.innodb_trx it
         ON ilw.blocking_trx_id = it.trx_id
       JOIN information_schema.processlist p
         ON it.trx_mysql_thread_id = p.id
       JOIN information_schema.innodb_trx it1
         ON ilw.requesting_trx_id = it1.trx_id
```

```
       JOIN information_schema.processlist p1
         ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

Saat mengidentifikasi sesi, opsi Anda mencakup yang berikut:

- Menghubungi pemilik aplikasi atau pengguna.
- Jika sesi pemblokiran idle, coba akhiri sesi pemblokiran. Tindakan ini dapat memicu rollback yang panjang. Untuk mempelajari cara mengakhiri sesi, lihat Mengakhiri sesi atau kueri.

Untuk informasi selengkapnya tentang cara mengidentifikasi transaksi pemblokiran, lihat Menggunakan Informasi Penguncian dan Transaksi InnoDB dalam Panduan Referensi MySQL.

synch/cond/sql/MDL_context::COND_wait_status

Peristiwa `synch/cond/sql/MDL_context::COND_wait_status` terjadi saat terdapat thread yang menunggu pada kunci metadata tabel.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora MySQL versi 2 dan 3

Konteks

Peristiwa `synch/cond/sql/MDL_context::COND_wait_status` menunjukkan bahwa terdapat thread yang menunggu pada kunci metadata tabel. Dalam kasus tertentu, satu sesi mempertahankan kunci metadata pada tabel, sementara sesi lainnya mencoba mendapatkan kunci yang sama pada tabel yang sama. Jika demikian, sesi kedua akan menunggu pada peristiwa tunggu `synch/cond/sql/MDL_context::COND_wait_status`.

MySQL menggunakan penguncian metadata untuk mengelola akses bersamaan ke objek basis data dan memastikan konsistensi data. Penguncian metadata berlaku untuk tabel, skema, acara terjadwal, tablespace, dan penguncian pengguna yang diperoleh dengan fungsi `get_lock`, serta program yang disimpan. Program yang disimpan mencakup prosedur, fungsi, dan pemicu. Untuk informasi selengkapnya, lihat Penguncian metadata dalam dokumentasi MySQL.

Daftar proses MySQL menunjukkan sesi ini dalam status `waiting for metadata lock`. Dalam Wawasan Performa, jika `Performance_schema` diaktifkan, maka peristiwa `synch/cond/sql/MDL_context::COND_wait_status` akan muncul.

Batas waktu default untuk kueri yang menunggu pada penguncian metadata didasarkan pada nilai parameter `lock_wait_timeout`, dengan nilai default 31.536.000 detik (365 hari).

Untuk detail selengkapnya tentang berbagai kunci InnoDB dan jenis kunci yang dapat menyebabkan konflik, lihat Penguncian InnoDB dalam dokumentasi MySQL.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `synch/cond/sql/MDL_context::COND_wait_status` muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

Transaksi yang berjalan lama

Satu atau lebih transaksi mengubah data dalam jumlah besar data dan menahan kunci pada tabel untuk waktu yang sangat lama.

Transaksi idle

Satu atau lebih transaksi tetap terbuka untuk waktu yang lama, tanpa di-commit atau di-roll back.

Pernyataan DDL pada tabel besar

Satu atau lebih pernyataan definisi data (DDL), seperti perintah `ALTER TABLE`, dijalankan pada tabel yang sangat besar.

Kunci tabel eksplisit

Terdapat kunci eksplisit pada tabel yang tidak dirilis tepat waktu. Misalnya, sebuah aplikasi mungkin menjalankan pernyataan `LOCK TABLE` secara salah.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda dan versi klaster DB Aurora MySQL.

Topik

- Mengidentifikasi sesi dan kueri penyebab peristiwa
- Memeriksa peristiwa masa lalu
- Menjalankan kueri pada Aurora MySQL versi 2
- Merespons sesi pemblokiran

Mengidentifikasi sesi dan kueri penyebab peristiwa

Anda dapat menggunakan Wawasan Performa untuk menampilkan kueri yang diblokir oleh peristiwa tunggu `synch/cond/sql/MDL_context::COND_wait_status`. Namun, untuk mengidentifikasi sesi pemblokiran, buat kueri tabel metadata dari `performance_schema` dan `information_schema` di klaster DB.

Biasanya, basis data dengan beban sedang hingga signifikan memiliki peristiwa tunggu. Peristiwa tunggu ini mungkin dapat diterima jika basis data berperforma optimal. Jika tidak, periksa di mana basis data tersebut menghabiskan waktu terbanyak. Lihat peristiwa tunggu yang berkontribusi ke beban tertinggi, lalu cari tahu apakah Anda dapat mengoptimalkan basis data dan aplikasi untuk mengurangi peristiwa tersebut.

Untuk menemukan kueri SQL yang bertanggung jawab atas beban tinggi:

1. Masuk ke AWS Management Console, lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa untuk instans DB tersebut akan muncul.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih SQL Teratas.

   Bagan ini mencantumkan kueri SQL yang bertanggung jawab atas beban. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar pemecahan masalah yang berguna menggunakan Wawasan Performa, lihat postingan Blog Basis Data AWS, Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa.

Memeriksa peristiwa masa lalu

Anda dapat memperoleh wawasan tentang peristiwa tunggu ini untuk memeriksa kejadian di masa lalu. Untuk melakukannya, selesaikan tindakan berikut:

- Periksa bahasa manipulasi data (DML) dan throughput serta latensi DDL untuk mengetahui apakah terdapat perubahan pada beban kerja.

  Anda dapat menggunakan Wawasan Performa untuk menemukan kueri yang menunggu pada peristiwa ini saat masalah terjadi. Anda juga dapat melihat inti sari kueri yang berjalan menjelang waktu masalah terjadi.
- Jika log audit atau log umum diaktifkan untuk klaster DB, Anda dapat memeriksa semua kueri yang berjalan pada objek (schema.table) yang terlibat dalam transaksi tunggu. Anda juga dapat memeriksa kueri yang selesai berjalan sebelum transaksi.

Informasi yang tersedia untuk memecahkan masalah peristiwa masa lalu terbatas. Melakukan pemeriksaan ini tidak menunjukkan objek yang menunggu informasi. Namun, Anda dapat mengidentifikasi tabel dengan beban berat saat peristiwa terjadi dan kumpulan baris yang sering dioperasikan yang menyebabkan konflik pada saat masalah terjadi. Anda kemudian dapat

menggunakan informasi ini untuk mereproduksi masalah di lingkungan pengujian dan memberikan wawasan tentang penyebabnya.

Menjalankan kueri pada Aurora MySQL versi 2

Di Aurora MySQL versi 2, Anda dapat mengidentifikasi sesi yang diblokir secara langsung dengan membuat kueri tabel `performance_schema` atau tampilan skema `sys`. Sebuah contoh dapat mengilustrasikan cara membuat kueri tabel untuk mengidentifikasi kueri dan sesi pemblokiran.

Dalam output daftar proses berikut, ID koneksi 89 sedang menunggu pada kunci metadata, dan menjalankan perintah `TRUNCATE TABLE`. Dalam kueri pada tabel `performance_schema` atau tampilan skema `sys`, output menunjukkan bahwa sesi pemblokiran adalah 76.

```
MySQL [(none)]> select @@version, @@aurora_version;
+-----------+------------------+
| @@version | @@aurora_version |
+-----------+------------------+
| 5.7.12    | 2.09.0           |
+-----------+------------------+
1 row in set (0.01 sec)

MySQL [(none)]> show processlist;
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+------------------------------+
| Id | User            | Host               | db        | Command | Time | State
                    | Info                           |
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+------------------------------+
|  2 | rdsadmin        | localhost          | NULL      | Sleep   |    0 | NULL
                    | NULL                           |
|  4 | rdsadmin        | localhost          | NULL      | Sleep   |    2 | NULL
                    | NULL                           |
|  5 | rdsadmin        | localhost          | NULL      | Sleep   |    1 | NULL
                    | NULL                           |
| 20 | rdsadmin        | localhost          | NULL      | Sleep   |    0 | NULL
                    | NULL                           |
| 21 | rdsadmin        | localhost          | NULL      | Sleep   |  261 | NULL
                    | NULL                           |
| 66 | auroramysql5712 | 172.31.21.51:52154 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
| 67 | auroramysql5712 | 172.31.21.51:52158 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                           |
```

```
| 68 | auroramysql5712 | 172.31.21.51:52150 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 69 | auroramysql5712 | 172.31.21.51:52162 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 70 | auroramysql5712 | 172.31.21.51:52160 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 71 | auroramysql5712 | 172.31.21.51:52152 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 72 | auroramysql5712 | 172.31.21.51:52156 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 73 | auroramysql5712 | 172.31.21.51:52164 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 74 | auroramysql5712 | 172.31.21.51:52166 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 75 | auroramysql5712 | 172.31.21.51:52168 | sbtest123 | Sleep   |    0 | NULL
                    | NULL                          |
| 76 | auroramysql5712 | 172.31.21.51:52170 | NULL      | Query   |    0 | starting
                    | show processlist              |
| 88 | auroramysql5712 | 172.31.21.51:52194 | NULL      | Query   |   22 | User sleep
                    | select sleep(10000)           |
| 89 | auroramysql5712 | 172.31.21.51:52196 | NULL      | Query   |    5 | Waiting for
 table metadata lock | truncate table sbtest.sbtest1 |
+----+-----------------+--------------------+-----------+---------+------
+---------------------------------+-------------------------------+
18 rows in set (0.00 sec)
```

Selanjutnya, sebuah kueri pada tabel `performance_schema` atau tampilan skema `sys` menunjukkan bahwa sesi pemblokiran adalah 76.

```
MySQL [(none)]> select * from sys.schema_table_lock_waits;

+---------------+-------------+-------------------+-------------
+-----------------------------+-------------------+----------------------
+---------------------------------+--------------------+-----------------------------
+-----------------------------+--------------------+--------------
+-----------------------------+--------------------+-----------------------
+-------------------------+-----------------------------+
| object_schema | object_name | waiting_thread_id | waiting_pid | waiting_account
          | waiting_lock_type | waiting_lock_duration | waiting_query
   | waiting_query_secs | waiting_query_rows_affected | waiting_query_rows_examined |
 blocking_thread_id | blocking_pid | blocking_account             | blocking_lock_type
 | blocking_lock_duration | sql_kill_blocking_query | sql_kill_blocking_connection |
```

```
+---------------+-------------+-------------------+-------------
+------------------------------+-------------------+-----------------------
+------------------------------+--------------------+----------------------------
+------------------------------+--------------------+--------------
+------------------------------+--------------------+-----------------------
+--------------------------+-------------------------------+
| sbtest        | sbtest1     |               121 |          89 |
 auroramysql5712@192.0.2.0    | EXCLUSIVE         | TRANSACTION           | truncate
 table sbtest.sbtest1 |                 10 |                            0 |
               0 |                108 |           76 | auroramysql5712@192.0.2.0    |
 SHARED_READ        | TRANSACTION            | KILL QUERY 76           | KILL 76
              |
+---------------+-------------+-------------------+-------------
+------------------------------+-------------------+-----------------------
+------------------------------+--------------------+----------------------------
+------------------------------+--------------------+--------------
+------------------------------+--------------------+-----------------------
+--------------------------+-------------------------------+
1 row in set (0.00 sec)
```

Merespons sesi pemblokiran

Saat mengidentifikasi sesi, opsi Anda mencakup yang berikut:

- Menghubungi pemilik aplikasi atau pengguna.
- Jika sesi pemblokiran idle, coba akhiri sesi pemblokiran. Tindakan ini dapat memicu rollback yang panjang. Untuk mempelajari cara mengakhiri sesi, lihat Mengakhiri sesi atau kueri.

Untuk informasi selengkapnya tentang cara mengidentifikasi transaksi pemblokiran, lihat Menggunakan Transaksi InnoDB dan Informasi Penguncian dalam dokumentasi MySQL.

synch/mutex/innodb/aurora_lock_thread_slot_futex

Peristiwa `synch/mutex/innodb/aurora_lock_thread_slot_futex` terjadi ketika satu sesi telah mengunci baris untuk pembaruan dan sesi lain mencoba memperbarui baris yang sama. Untuk informasi selengkapnya, lihat Penguncian InnoDB dalam Referensi MySQL.

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora MySQL versi 2

> **Note**
>
> Di Aurora MySQL versi 3.01.0 dan 3.01.1, peristiwa tunggu ini dilaporkan sebagai io/table/sql/handler.

Kemungkinan penyebab peningkatan peristiwa tunggu

Beberapa pernyataan bahasa manipulasi data (DML) mengakses baris atau sejumlah baris yang sama secara serentak.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada peristiwa tunggu lain yang Anda lihat.

Topik

- Menemukan dan merespons pernyataan SQL yang bertanggung jawab atas peristiwa tunggu ini
- Menemukan dan merespons sesi pemblokiran

Menemukan dan merespons pernyataan SQL yang bertanggung jawab atas peristiwa tunggu ini

Gunakan Wawasan Performa untuk mengidentifikasi pernyataan SQL yang bertanggung jawab atas peristiwa tunggu ini. Pertimbangkan strategi berikut:

- Jika kunci baris merupakan masalah yang terus-menerus, coba tulis ulang aplikasi untuk menggunakan penguncian optimis.
- Gunakan pernyataan multibaris.
- Bagikan beban kerja ke objek basis data yang berbeda-beda. Anda dapat melakukannya melalui partisi.
- Periksa nilai parameter `innodb_lock_wait_timeout`. Parameter ini mengontrol durasi transaksi menunggu sebelum menghasilkan kesalahan batas waktu.

Untuk ikhtisar pemecahan masalah yang berguna menggunakan Wawasan Performa, lihat postingan blog Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa.

Menemukan dan merespons sesi pemblokiran

Cari tahu apakah sesi pemblokiran idle atau aktif. Cari tahu juga apakah sesi tersebut berasal dari aplikasi atau pengguna aktif.

Untuk mengidentifikasi sesi yang memegang kunci, Anda dapat menjalankan `SHOW ENGINE INNODB STATUS`. Contoh berikut menunjukkan output sampel.

```
mysql> SHOW ENGINE INNODB STATUS;

---------------------TRANSACTION 302631452, ACTIVE 2 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 376, 1 row lock(s)
MySQL thread id 80109, OS thread handle 0x2ae915060700, query id 938819 10.0.4.12
 reinvent updating
UPDATE sbtest1 SET k=k+1 WHERE id=503
------- TRX HAS BEEN WAITING 2 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 148 page no 11 n bits 30 index `PRIMARY` of table
 `sysbench2`.`sbtest1` trx id 302631452 lock_mode X locks rec but not gap waiting
Record lock, heap no 30 PHYSICAL RECORD: n_fields 6; compact format; info bits 0
```

Sebagai alternatif, Anda dapat menggunakan kueri berikut untuk mengekstrak detail tentang kunci saat ini.

```
mysql> SELECT p1.id waiting_thread,
              p1.user waiting_user,
              p1.host waiting_host,
              it1.trx_query waiting_query,
              ilw.requesting_trx_id waiting_transaction,
              ilw.blocking_lock_id blocking_lock,
              il.lock_mode blocking_mode,
              il.lock_type blocking_type,
              ilw.blocking_trx_id blocking_transaction,
              CASE it.trx_state
                WHEN 'LOCK WAIT'
                THEN it.trx_state
                ELSE p.state
              END blocker_state,
              il.lock_table locked_table,
              it.trx_mysql_thread_id blocker_thread,
              p.user blocker_user,
              p.host blocker_host
       FROM information_schema.innodb_lock_waits ilw
```

```
       JOIN information_schema.innodb_locks il
         ON ilw.blocking_lock_id = il.lock_id
        AND ilw.blocking_trx_id = il.lock_trx_id
       JOIN information_schema.innodb_trx it
         ON ilw.blocking_trx_id = it.trx_id
       JOIN information_schema.processlist p
         ON it.trx_mysql_thread_id = p.id
       JOIN information_schema.innodb_trx it1
         ON ilw.requesting_trx_id = it1.trx_id
       JOIN information_schema.processlist p1
         ON it1.trx_mysql_thread_id = p1.id\G

*************************** 1. row ***************************
      waiting_thread: 3561959471
        waiting_user: reinvent
        waiting_host: 123.456.789.012:20485
       waiting_query: select id1 from test.t1 where id1=1 for update
 waiting_transaction: 312337314
       blocking_lock: 312337287:261:3:2
       blocking_mode: X
       blocking_type: RECORD
blocking_transaction: 312337287
       blocker_state: User sleep
        locked_table: `test`.`t1`
      blocker_thread: 3561223876
        blocker_user: reinvent
        blocker_host: 123.456.789.012:17746
1 row in set (0.04 sec)
```

Saat mengidentifikasi sesi, opsi Anda mencakup yang berikut:

- Menghubungi pemilik aplikasi atau pengguna.
- Jika sesi pemblokiran idle, coba akhiri sesi pemblokiran. Tindakan ini dapat memicu rollback yang panjang. Untuk mempelajari cara mengakhiri sesi, lihat Mengakhiri sesi atau kueri.

Untuk informasi selengkapnya tentang cara mengidentifikasi transaksi pemblokiran, lihat Menggunakan Informasi Penguncian dan Transaksi InnoDB dalam Panduan Referensi MySQL.

synch/mutex/innodb/buf_pool_mutex

Peristiwa `synch/mutex/innodb/buf_pool_mutex` terjadi saat thread telah mendapat kunci pada kumpulan buffer InnoDB untuk mengakses halaman dalam memori.

Topik

- Versi mesin yang relevan
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang relevan

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora MySQL versi 2

Konteks

Mutex `buf_pool` adalah mutex tunggal yang melindungi struktur data kontrol dari pool buffer.

Untuk informasi selengkapnya, lihat Memantau Peristiwa Tunggu Mutex InnoDB Menggunakan Skema Performa dalam dokumentasi MySQL.

Kemungkinan penyebab peningkatan peristiwa tunggu

Ini adalah peristiwa tunggu khusus beban kerja. Penyebab umum `synch/mutex/innodb/buf_pool_mutex` muncul di antara peristiwa tunggu teratas mencakup yang berikut:

- Ukuran pool buffer tidak cukup besar untuk menampung set data kerja.
- Beban kerja lebih spesifik untuk halaman tertentu dari tabel tertentu dalam basis data, yang mengarah ke pertentangan dalam pool buffer.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Mengidentifikasi sesi dan kueri penyebab peristiwa

Biasanya, basis data dengan beban sedang hingga signifikan memiliki peristiwa tunggu. Peristiwa tunggu ini mungkin dapat diterima jika basis data berperforma optimal. Jika tidak, periksa di mana basis data tersebut menghabiskan waktu terbanyak. Lihat peristiwa tunggu yang berkontribusi ke

beban tertinggi, lalu cari tahu apakah Anda dapat mengoptimalkan basis data dan aplikasi untuk mengurangi peristiwa tersebut.

Untuk melihat bagan SQL Teratas di Konsol Manajemen AWS:

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa ditampilkan untuk instans DB tersebut.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bawah bagan Beban basis data, pilih SQL Teratas.

   Bagan ini mencantumkan kueri SQL yang bertanggung jawab atas beban. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar pemecahan masalah yang berguna menggunakan Wawasan Performa, lihat postingan blog Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa.

Menggunakan Wawasan Performa

Peristiwa ini terkait dengan beban kerja. Anda dapat menggunakan Wawasan Performa untuk melakukan hal berikut:

- Mengidentifikasi kapan peristiwa tunggu dimulai dan apakah ada perubahan beban kerja pada saat itu dari log aplikasi atau sumber terkait.
- Mengidentifikasi pernyataan SQL yang bertanggung jawab atas peristiwa tunggu ini. Periksa rencana eksekusi kueri untuk memastikan bahwa kueri ini dioptimalkan dan menggunakan indeks yang sesuai.

  Jika kueri teratas yang bertanggung jawab atas peristiwa tunggu berkaitan dengan objek atau tabel basis data yang sama, coba buat partisi untuk objek atau tabel tersebut.

Membuat Aurora Replicas

Anda dapat membuat Aurora Replicas untuk menyajikan lalu lintas hanya-baca. Anda juga dapat menggunakan Aurora Auto Scaling untuk menangani lonjakan lalu lintas baca. Pastikan untuk menjalankan tugas hanya-baca terjadwal dan pencadangan logis pada Aurora Replicas.

Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Auto Scaling dengan Replika Aurora.

Memeriksa ukuran pool buffer

Periksa apakah ukuran pool buffer cukup untuk beban kerja dengan melihat metrik `innodb_buffer_pool_wait_free`. Jika nilai metrik ini tinggi dan terus bertambah, berarti ukuran pool buffer tidak cukup untuk menangani beban kerja. Jika `innodb_buffer_pool_size` telah diatur dengan benar, nilai `innodb_buffer_pool_wait_free` pasti kecil. Untuk informasi selengkapnya, lihat Innodb_buffer_pool_wait_free dalam dokumentasi MySQL.

Tingkatkan ukuran pool buffer jika instans DB memiliki cukup memori untuk buffer sesi dan tugas sistem operasi. Jika tidak, ubah instans DB ke kelas instans DB yang lebih besar untuk memperoleh memori tambahan yang dapat dialokasikan ke pool buffer.

> **Note**
>
> Aurora MySQL secara otomatis menyesuaikan nilai `innodb_buffer_pool_instances` berdasarkan `innodb_buffer_pool_size` yang dikonfigurasi.

Memantau riwayat status global

Dengan memantau tingkat perubahan variabel status, Anda dapat mendeteksi masalah penguncian atau memori pada instans DB Anda. Aktifkan Global Status History (GoSH) jika belum diaktifkan. Untuk informasi selengkapnya tentang GoSH, lihat Mengelola riwayat status global.

Anda juga dapat membuat metrik Amazon CloudWatch kustom untuk memantau variabel status. Untuk informasi selengkapnya, lihat Memublikasikan metrik kustom.

synch/mutex/innodb/fil_system_mutex

Peristiwa `synch/mutex/innodb/fil_system_mutex` terjadi ketika sesi menunggu untuk mengakses cache memori tablespace.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu

- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora MySQL versi 2 dan 3

Konteks

InnoDB menggunakan tablespace untuk mengelola area penyimpanan untuk tabel dan file log. Cache memori tablespace adalah struktur memori global yang memelihara informasi tentang tablespace. MySQL menggunakan peristiwa tunggu `synch/mutex/innodb/fil_system_mutex` untuk mengontrol akses bersamaan ke cache memori tablespace.

Peristiwa `synch/mutex/innodb/fil_system_mutex` menunjukkan bahwa saat ini ada lebih dari satu operasi yang perlu mengambil dan memanipulasi informasi dalam cache memori tablespace untuk tablespace yang sama.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `synch/mutex/innodb/fil_system_mutex` muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, hal ini biasanya terjadi ketika terdapat semua kondisi berikut:

- Peningkatan operasi bahasa manipulasi data (DML) bersamaan yang memperbarui atau menghapus data dalam tabel yang sama.
- Tablespace untuk tabel ini sangat besar dan memiliki banyak halaman data.
- Faktor pengisian untuk halaman data ini rendah.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengidentifikasi sesi dan kueri penyebab peristiwa
- Mengatur ulang tabel besar di luar jam sibuk

Mengidentifikasi sesi dan kueri penyebab peristiwa

Biasanya, basis data dengan beban sedang hingga signifikan memiliki peristiwa tunggu. Peristiwa tunggu ini mungkin dapat diterima jika basis data berperforma optimal. Jika tidak, periksa di mana basis data tersebut menghabiskan waktu terbanyak. Lihat peristiwa tunggu yang berkontribusi ke beban tertinggi, lalu cari tahu apakah Anda dapat mengoptimalkan basis data dan aplikasi untuk mengurangi peristiwa tersebut.

Untuk menemukan kueri SQL yang bertanggung jawab atas beban tinggi:

1. Masuk ke AWS Management Console, lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa untuk instans DB tersebut akan muncul.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih SQL Teratas.

   Bagan ini mencantumkan kueri SQL yang bertanggung jawab atas beban. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar pemecahan masalah yang berguna menggunakan Wawasan Performa, lihat postingan blog Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa.

Cara lain untuk mengetahui kueri yang menyebabkan tingginya jumlah peristiwa tunggu `synch/mutex/innodb/fil_system_mutex` adalah dengan memeriksa `performance_schema`, seperti pada contoh berikut.

```
mysql> select * from performance_schema.events_waits_current where EVENT_NAME='wait/
synch/mutex/innodb/fil_system_mutex'\G
*************************** 1. row ***************************
            THREAD_ID: 19
             EVENT_ID: 195057
         END_EVENT_ID: 195057
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:6700
          TIMER_START: 1010146190118400
            TIMER_END: 1010146196524000
           TIMER_WAIT: 6405600
```

```
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 2. row ***************************
            THREAD_ID: 23
             EVENT_ID: 5480
         END_EVENT_ID: 5480
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:5906
          TIMER_START: 995269979908800
            TIMER_END: 995269980159200
           TIMER_WAIT: 250400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
*************************** 3. row ***************************
            THREAD_ID: 55
             EVENT_ID: 23233794
         END_EVENT_ID: NULL
           EVENT_NAME: wait/synch/mutex/innodb/fil_system_mutex
               SOURCE: fil0fil.cc:449
          TIMER_START: 1010492125341600
            TIMER_END: 1010494304900000
           TIMER_WAIT: 2179558400
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
```

```
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 47285552262176
     NESTING_EVENT_ID: 23233786
   NESTING_EVENT_TYPE: WAIT
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: NULL
```

Mengatur ulang tabel besar di luar jam sibuk

Atur ulang tabel besar yang Anda identifikasi sebagai sumber tingginya jumlah peristiwa tunggu `synch/mutex/innodb/fil_system_mutex` selama periode pemeliharaan di luar jam produksi. Dengan melakukannya, Anda akan memastikan bahwa pembersihan peta tablespace internal tidak terjadi saat akses cepat ke tabel sangat penting. Untuk informasi tentang cara mengatur ulang tabel, lihat Pernyataan OPTIMIZE TABLE dalam Referensi MySQL.

synch/mutex/innodb/trx_sys_mutex

Peristiwa `synch/mutex/innodb/trx_sys_mutex` terjadi ketika terdapat aktivitas basis data tinggi dengan transaksi dalam jumlah besar.

Topik

- Versi mesin yang relevan
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang relevan

Informasi peristiwa tunggu ini didukung untuk versi mesin berikut:

- Aurora MySQL versi 2 dan 3

Konteks

Secara internal, mesin basis data InnoDB menggunakan tingkat isolasi baca berulang dengan snapshot untuk menghadirkan konsistensi baca. Hal ini memberi Anda gambaran titik waktu basis data pada saat snapshot dibuat.

Di InnoDB, semua perubahan diterapkan ke basis data segera setelah tersedia, terlepas dari apakah perubahan tersebut di-commit. Pendekatan ini berarti bahwa tanpa kontrol konkurensi multiversi (MVCC), semua pengguna yang terhubung ke basis data akan melihat semua perubahan dan baris terbaru. Karena itu, InnoDB memerlukan cara untuk melacak perubahan guna memahami strategi rollback bila diperlukan.

Untuk melakukannya, InnoDB menggunakan sistem transaksi (`trx_sys`) untuk melacak snapshot. Sistem transaksi melakukan hal berikut:

- Melacak ID transaksi untuk setiap baris dalam log pembatalan.
- Menggunakan struktur InnoDB internal, `ReadView`, yang membantu mengidentifikasi ID transaksi yang terlihat untuk snapshot.

Kemungkinan penyebab peningkatan peristiwa tunggu

Setiap operasi basis data yang memerlukan penanganan ID transaksi yang konsisten dan terkontrol (membuat, membaca, memperbarui, dan menghapus) akan menghasilkan panggilan dari `trx_sys` ke mutex.

Panggilan ini terjadi di dalam tiga fungsi:

- `trx_sys_mutex_enter` – Menciptakan mutex.
- `trx_sys_mutex_exit` – Melepaskan mutex.
- `trx_sys_mutex_own` – Menguji apakah mutex dimiliki.

Instrumentasi Skema Performa InnoDB melacak semua panggilan mutex `trx_sys`. Pelacakan mencakup, tetapi tidak terbatas pada, manajemen `trx_sys` pada pengaktifan atau penonaktifan basis data, operasi rollback, pembersihan pembatalan, akses baca baris, dan pemuatan pool buffer. Aktivitas basis data yang tinggi dengan transaksi dalam jumlah besar mengakibatkan kemunculan `synch/mutex/innodb/trx_sys_mutex` di antara peristiwa tunggu teratas.

Untuk informasi selengkapnya, lihat Memantau Peristiwa Tunggu Mutex InnoDB Menggunakan Skema Performa dalam dokumentasi MySQL.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Mengidentifikasi sesi dan kueri penyebab peristiwa

Biasanya, basis data dengan beban sedang hingga signifikan memiliki peristiwa tunggu. Peristiwa tunggu ini mungkin dapat diterima jika basis data berperforma optimal. Jika tidak, periksa di mana basis data tersebut menghabiskan waktu terbanyak. Lihat peristiwa tunggu yang berkontribusi terhadap beban tertinggi. Cari tahu apakah Anda dapat mengoptimalkan basis data dan aplikasi untuk mengurangi peristiwa tersebut.

Untuk melihat bagan SQL Teratas di AWS Management Console:

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa ditampilkan untuk instans DB tersebut.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bawah bagan Beban basis data, pilih SQL Teratas.

   Bagan ini mencantumkan kueri SQL yang bertanggung jawab atas beban. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar pemecahan masalah yang berguna menggunakan Wawasan Performa, lihat postingan blog Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa.

Memeriksa peristiwa tunggu lainnya

Periksa peristiwa tunggu lain yang terkait dengan peristiwa tunggu `synch/mutex/innodb/trx_sys_mutex`. Dengan melakukannya, Anda dapat memperoleh informasi selengkapnya tentang sifat beban kerja. Sejumlah besar transaksi dapat mengurangi throughput, tetapi beban kerja mungkin juga mengharuskannya.

Untuk informasi selengkapnya tentang cara mengoptimalkan transaksi, lihat Mengoptimalkan Manajemen Transaksi InnoDB dalam dokumentasi MySQL.

synch/sxlock/innodb/hash_table_locks

Peristiwa `synch/sxlock/innodb/hash_table_locks` terjadi ketika halaman yang tidak ditemukan di pool buffer harus dibaca dari penyimpanan.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk versi berikut:

- Aurora MySQL versi 2 dan 3

Konteks

Peristiwa `synch/sxlock/innodb/hash_table_locks` menunjukkan bahwa beban kerja sering kali mengakses data yang tidak disimpan dalam pool buffer. Peristiwa tunggu ini dikaitkan dengan penambahan halaman baru dan pengosongan data lama dari pool buffer. Data yang disimpan dalam pool buffer usang dan data baru harus di-cache, jadi halaman usang dikosongkan untuk memungkinkan caching halaman baru. MySQL menggunakan algoritma least recently used (LRU) untuk mengosongkan halaman dari pool buffer. Beban kerja mencoba mengakses data yang belum dimuat ke dalam pool buffer atau data yang telah dikosongkan dari pool buffer.

Peristiwa tunggu ini terjadi ketika beban kerja harus mengakses data dalam file pada disk atau ketika blok dibebaskan dari atau ditambahkan ke daftar LRU pool buffer. Operasi ini menunggu untuk memperoleh kunci yang dikecualikan bersama (SX-lock). SX-lock ini digunakan untuk sinkronisasi melalui tabel hash, yakni tabel dalam memori yang dirancang untuk meningkatkan performa akses pool buffer.

Untuk informasi selengkapnya, lihat Pool Buffer dalam dokumentasi MySQL.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa tunggu `synch/sxlock/innodb/hash_table_locks` muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

Pool buffer berukuran sangat kecil

Ukuran pool buffer terlalu kecil untuk menyimpan semua halaman yang sering diakses dalam memori.

Beban kerja berat

Beban kerja menyebabkan seringnya pengosongan dan halaman data dimuat ulang di cache buffer.

Kesalahan membaca halaman

Terjadi kesalahan saat membaca halaman di pool buffer, yang mungkin mengindikasikan adanya kerusakan data.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Meningkatkan ukuran pool buffer
- Meningkatkan pola akses data
- Mengurangi atau mencegah pemindaian tabel penuh
- Memeriksa log kesalahan untuk kerusakan halaman

Meningkatkan ukuran pool buffer

Pastikan pool buffer memiliki ukuran yang sesuai untuk beban kerja. Untuk melakukannya, Anda dapat memeriksa laju hit cache pool buffer. Biasanya, jika nilainya turun di bawah 95%, coba tingkatkan ukuran pool buffer. Pool buffer yang lebih besar dapat lebih lama menyimpan halaman yang sering diakses dalam memori. Untuk meningkatkan ukuran pool buffer, ubah nilai parameter `innodb_buffer_pool_size`. Nilai default parameter ini didasarkan pada ukuran kelas instans DB. Untuk informasi selengkapnya, lihat Praktik terbaik untuk konfigurasi basis data Amazon Aurora MySQL.

Meningkatkan pola akses data

Periksa kueri yang terpengaruh oleh peristiwa tunggu ini dan rencana eksekusinya. Coba tingkatkan pola akses data. Misalnya, jika menggunakan mysqli_result::fetch_array, Anda dapat mencoba meningkatkan ukuran pengambilan array.

Anda dapat menggunakan Wawasan Performa untuk menampilkan kueri dan sesi yang mungkin menyebabkan peristiwa tunggu `synch/sxlock/innodb/hash_table_locks`.

Untuk menemukan kueri SQL yang bertanggung jawab atas beban tinggi:

1. Masuk ke AWS Management Console, lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Wawasan Performa.
3. Pilih instans DB. Dasbor Wawasan Performa ditampilkan untuk instans DB tersebut.
4. Dalam bagan Beban basis data, pilih Potong berdasarkan masa tunggu.
5. Di bagian bawah halaman, pilih SQL Teratas.

   Bagan ini mencantumkan kueri SQL yang bertanggung jawab atas beban. Kueri di bagian atas daftar memiliki tanggung jawab terbesar. Untuk mengatasi kemacetan, fokus pada pernyataan tersebut.

Untuk ikhtisar pemecahan masalah yang berguna menggunakan Wawasan Performa, lihat postingan Blog Basis Data AWS, Menganalisis Beban Kerja Amazon Aurora MySQL dengan Wawasan Performa.

Mengurangi atau mencegah pemindaian tabel penuh

Pantau beban kerja untuk melihat apakah pemindaian tabel penuh dijalankan, dan jika ya, kurangi atau cegah pemindaian tersebut. Misalnya, Anda dapat memantau variabel status seperti `Handler_read_rnd_next`. Untuk informasi selengkapnya, lihat Variabel Status Server dalam dokumentasi MySQL.

Memeriksa log kesalahan untuk kerusakan halaman

Anda dapat memeriksa mysql-error.log untuk pesan terkait kerusakan yang terdeteksi menjelang waktu terjadinya masalah. Pesan yang dapat Anda tangani untuk menyelesaikan masalah terdapat di log kesalahan. Anda mungkin perlu membuat ulang objek yang dilaporkan rusak.

Menyesuaikan Aurora MySQL dengan status thread

Tabel berikut merangkum status thread paling umum untuk Aurora MySQL.

Status thread umum	Deskripsi
???	Status thread ini menunjukkan bahwa sebuah thread sedang memproses pernyataan `SELECT` yang memerlukan penggunaan tabel sementara internal untuk mengurutkan data.
???	Status thread ini menunjukkan bahwa sebuah thread sedang membaca dan memfilter baris untuk kueri guna menentukan set hasil yang benar.

creating sort index

Status thread `creating sort index` menunjukkan bahwa thread sedang memproses pernyataan `SELECT` yang memerlukan penggunaan tabel sementara internal untuk mengurutkan data.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi status thread ini didukung untuk versi berikut:

- Aurora MySQL versi 2 hingga 2.09.2

Konteks

Status `creating sort index` muncul saat kueri dengan klausa `ORDER BY` atau `GROUP BY` tidak dapat menggunakan indeks yang ada untuk melakukan operasi. Dalam hal ini, MySQL perlu

melakukan operasi `filesort` yang lebih mahal. Operasi ini biasanya dilakukan dalam memori jika set hasil tidak terlalu besar. Jika set hasil besar, operasi akan melibatkan pembuatan file pada disk.

Kemungkinan penyebab peningkatan peristiwa tunggu

Kemunculan `creating sort index` tidak dengan sendirinya menunjukkan adanya suatu masalah. Jika performa buruk, dan Anda sering melihat instans `creating sort index`, penyebab yang paling mungkin adalah kueri lambat dengan operator `ORDER BY` atau `GROUP BY`.

Tindakan

Pedoman umumnya adalah menemukan kueri dengan klausa `ORDER BY` atau `GROUP BY` yang terkait dengan peningkatan dalam status `creating sort index`. Lalu, lihat apakah menambahkan indeks atau meningkatkan ukuran buffer urutan akan memecahkan masalah.

Topik

- Mengaktifkan Skema Performa jika tidak aktif
- Mengidentifikasi kueri masalah
- Memeriksa "jelaskan rencana" untuk penggunaan filesort
- Meningkatkan ukuran buffer urutan

Mengaktifkan Skema Performa jika tidak aktif

Wawasan Performa akan melaporkan status thread hanya jika instrumen Skema Performa tidak aktif. Bila instrumen Skema Performa diaktifkan, Wawasan Performa akan melaporkan peristiwa tunggu. Instrumen Skema Performa memberikan wawasan tambahan dan alat yang lebih baik untuk menyelidiki potensi masalah performa. Oleh karena itu, sebaiknya Anda mengaktifkan Skema Performa. Untuk informasi selengkapnya, lihat Mengaktifkan Skema Performa untuk Wawasan Performa di Aurora MySQL.

Mengidentifikasi kueri masalah

Untuk mengidentifikasi kueri saat ini yang menyebabkan peningkatan dalam status `creating sort index`, jalankan `show processlist` dan lihat apakah kueri memiliki `ORDER BY` atau `GROUP BY`. Cara lainnya, jalankan `explain for connection N`, dengan `N` adalah ID daftar proses kueri dengan `filesort`.

Untuk mengidentifikasi kueri sebelumnya yang menyebabkan peningkatan ini, aktifkan log kueri lambat dan temukan kueri dengan `ORDER BY`. Jalankan `EXPLAIN` pada kueri lambat dan cari

"menggunakan filesort". Untuk informasi selengkapnya, lihat Memeriksa "jelaskan rencana" untuk penggunaan filesort.

Memeriksa "jelaskan rencana" untuk penggunaan filesort

Identifikasi pernyataan dengan klausa `ORDER BY` atau `GROUP BY` yang menghasilkan status `creating sort index`.

Contoh berikut menunjukkan cara menjalankan `explain` pada kueri. Kolom `Extra` menunjukkan bahwa kueri ini menggunakan `filesort`.

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 2064548
     filtered: 100.00
        Extra: Using filesort
1 row in set, 1 warning (0.01 sec)
```

Contoh berikut menunjukkan hasil menjalankan `EXPLAIN` pada kueri yang sama setelah indeks dibuat pada kolom `c1`.

```
mysql> alter table mytable add index (c1);
```

```
mysql> explain select * from mytable order by c1 limit 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: mytable
   partitions: NULL
         type: index
possible_keys: NULL
          key: c1
      key_len: 1023
```

```
          ref: NULL
         rows: 10
     filtered: 100.00
        Extra: Using index
1 row in set, 1 warning (0.01 sec)
```

Untuk informasi tentang penggunaan indeks pada pengoptimalan tata urutan, lihat Pengoptimalan ORDER BY dalam dokumentasi MySQL.

Meningkatkan ukuran buffer urutan

Untuk mengetahui apakah kueri tertentu memerlukan proses `filesort` yang membuat file pada disk, periksa nilai variabel `sort_merge_passes` setelah menjalankan kueri. Berikut adalah contohnya.

```
mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)

--- run query
mysql> select * from mytable order by u limit 10;
--- run status again:

mysql> show session status like 'sort_merge_passes';
+-------------------+-------+
| Variable_name     | Value |
+-------------------+-------+
| Sort_merge_passes | 0     |
+-------------------+-------+
1 row in set (0.01 sec)
```

Jika nilai `sort_merge_passes` tinggi, coba tingkatkan ukuran buffer urutan. Terapkan peningkatan tersebut pada tahap sesi. Menerapkannya secara global dapat secara signifikan meningkatkan jumlah penggunaan RAM MySQL. Contoh berikut menunjukkan cara mengubah ukuran buffer urutan sebelum menjalankan kueri.

```
mysql> set session sort_buffer_size=10*1024*1024;
Query OK, 0 rows affected (0.00 sec)
```

```
-- run query
```

sending data

Status thread `sending data` menunjukkan bahwa thread membaca dan memfilter baris untuk kueri guna menentukan set hasil yang benar. Namanya agak membingungkan karena menyiratkan status mentransfer data, bukan mengumpulkan dan menyiapkan data untuk dikirim nanti.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi status thread ini didukung untuk versi berikut:

- Aurora MySQL versi 2 hingga 2.09.2

Konteks

Banyak status thread tidak bertahan lama. Operasi yang terjadi selama `sending data` cenderung melakukan pembacaan disk atau cache dalam jumlah besar. Oleh karena itu, `sending data` sering kali merupakan status yang paling lama berjalan selama masa kueri tertentu. Status ini muncul saat Aurora MySQL melakukan hal berikut:

- Membaca dan memproses baris untuk pernyataan `SELECT`
- Menjalankan pembacaan dalam jumlah besar baik dari disk atau memori
- Menyelesaikan pembacaan lengkap semua data dari kueri tertentu
- Membaca data dari tabel, indeks, atau pekerjaan prosedur yang disimpan
- Menyortir, mengelompokkan, atau menyusun data

Setelah status `sending data` selesai menyiapkan data, status thread `writing to net` akan menunjukkan pengembalian data ke klien. Biasanya, `writing to net` diambil hanya ketika set hasil sangat besar atau latensi jaringan yang parah memperlambat transfer.

Kemungkinan penyebab peningkatan peristiwa tunggu

Kemunculan `sending data` tidak dengan sendirinya menunjukkan adanya suatu masalah. Jika performa buruk, dan Anda sering melihat instans `sending data`, berikut adalah penyebab yang paling memungkinkan.

Topik

- Kueri yang tidak efisien
- Konfigurasi server suboptimal

Kueri yang tidak efisien

Dalam sebagian besar kasus, status ini terutama disebabkan oleh kueri yang tidak menggunakan indeks yang sesuai untuk menemukan set hasil kueri tertentu. Sebagai contoh, pertimbangkan kueri yang membaca tabel berisi 10 juta data untuk semua pesanan yang ditempatkan di California, dengan kolom status tidak diindeks atau diindeks dengan buruk. Pada contoh terakhir, indeks mungkin ada, tetapi pengoptimal mengabaikannya karena kardinalitas rendah.

Konfigurasi server suboptimal

Jika beberapa kueri muncul dalam status `sending data`, server basis data mungkin dikonfigurasi dengan buruk. Secara khusus, server mungkin memiliki masalah berikut:

- Server basis data tidak memiliki kapasitas komputasi yang cukup: I/O disk, jenis dan kecepatan disk, CPU, atau jumlah CPU.
- Server kekurangan sumber daya yang dialokasikan, seperti pool buffer InnoDB untuk tabel InnoDB atau buffer kunci untuk tabel MyISAM.
- Pengaturan memori per-thread seperti `sort_buffer`, `read_buffer`, dan `join_buffer` menghabiskan lebih banyak RAM dari yang dibutuhkan, sehingga membuat server fisik kekurangan sumber daya memori.

Tindakan

Pedoman umumnya adalah menemukan kueri yang mengembalikan baris dalam jumlah besar dengan memeriksa Skema Performa. Jika pencatatan kueri yang tidak menggunakan indeks diaktifkan, Anda juga dapat memeriksa hasil dari log lambat.

Topik

- Mengaktifkan Skema Performa jika tidak aktif
- Memeriksa pengaturan memori
- Memeriksa "jelaskan rencana" untuk penggunaan indeks
- Memeriksa volume data yang dikembalikan
- Memeriksa masalah konkurensi
- Memeriksa struktur kueri

Mengaktifkan Skema Performa jika tidak aktif

Wawasan Performa akan melaporkan status thread hanya jika instrumen Skema Performa tidak aktif. Bila instrumen Skema Performa diaktifkan, Wawasan Performa akan melaporkan peristiwa tunggu. Instrumen Skema Performa memberikan wawasan tambahan dan alat yang lebih baik untuk menyelidiki potensi masalah performa. Oleh karena itu, sebaiknya Anda mengaktifkan Skema Performa. Untuk informasi selengkapnya, lihat Mengaktifkan Skema Performa untuk Wawasan Performa di Aurora MySQL.

Memeriksa pengaturan memori

Periksa pengaturan memori untuk pool buffer utama. Pastikan pool ini memiliki ukuran yang sesuai untuk beban kerja. Jika basis data Anda menggunakan beberapa instans pool buffer, pastikan instans tersebut tidak dibagi menjadi banyak pool buffer kecil. Thread hanya dapat menggunakan satu pool buffer pada satu waktu.

Pastikan pengaturan memori berikut yang digunakan untuk setiap thread memiliki ukuran yang benar:

- read_buffer
- read_rnd_buffer
- sort_buffer
- join_buffer
- binlog_cache

Gunakan nilai default, kecuali jika Anda memiliki alasan khusus untuk mengubah pengaturan.

Memeriksa "jelaskan rencana" untuk penggunaan indeks

Untuk kueri dalam status thread `sending data`, periksa rencana untuk mengetahui apakah indeks yang sesuai telah digunakan. Jika kueri tidak menggunakan indeks yang berguna, coba

tambahkan petunjuk seperti `USE INDEX` atau `FORCE INDEX`. Petunjuk dapat menambah atau mengurangi banyak waktu yang diperlukan untuk menjalankan kueri, jadi berhati-hatilah sebelum menambahkannya.

Memeriksa volume data yang dikembalikan

Periksa tabel yang ditanyakan dan jumlah data yang ada di dalamnya. Apakah data ini dapat diarsipkan? Dalam banyak kasus, penyebab waktu eksekusi kueri yang buruk bukanlah hasil dari rencana kueri, tetapi volume data yang akan diproses. Banyak developer sangat efisien dalam menambahkan data ke basis data, tetapi jarang mempertimbangkan siklus proses set data dalam fase desain dan pengembangan.

Cari kueri yang berperforma baik dalam basis data volume rendah, tetapi berperforma buruk di sistem Anda saat ini. Terkadang, developer yang mendesain kueri tertentu mungkin tidak menyadari bahwa kueri tersebut mengembalikan 350.000 baris. Developer mungkin telah mengembangkan kueri di lingkungan volume yang lebih rendah, dengan set data lebih kecil dari yang dimiliki lingkungan produksi.

Memeriksa masalah konkurensi

Periksa apakah beberapa kueri dari jenis yang sama berjalan pada waktu yang sama. Format kueri tertentu berjalan secara efisien bila dijalankan sendiri. Namun, jika format kueri serupa berjalan bersama, atau dalam volume tinggi, hal tersebut dapat menimbulkan masalah konkurensi. Masalah ini sering kali muncul bila basis data menggunakan tabel sementara untuk merender hasil. Tingkat isolasi transaksi yang ketat juga dapat menyebabkan masalah konkurensi.

Jika tabel dibaca dan ditulis secara bersamaan, basis data mungkin menggunakan kunci. Untuk membantu mengidentifikasi periode performa yang buruk, periksa penggunaan basis data melalui proses batch skala besar. Untuk melihat kunci dan rollback terbaru, periksa output dari perintah `SHOW ENGINE INNODB STATUS`.

Memeriksa struktur kueri

Periksa apakah kueri yang diambil dari status ini menggunakan subkueri. Jenis kueri ini sering kali menimbulkan performa yang buruk karena basis data mengompilasi hasil secara internal, lalu menggantinya kembali ke kueri untuk merender data. Proses ini merupakan langkah ekstra untuk basis data. Dalam banyak kasus, langkah ini dapat menimbulkan performa yang buruk pada kondisi pemuatan yang sangat bersamaan.

Periksa juga apakah kueri Anda menggunakan klausa `ORDER BY` dan `GROUP BY` dalam jumlah besar. Dalam operasi semacam ini, sering kali basis data harus terlebih dahulu membentuk

set data secara keseluruhan dalam memori. Selanjutnya, basis data harus menyusun atau mengelompokkannya secara khusus sebelum mengembalikannya ke klien.

Menyesuaikan Aurora MySQL dengan wawasan proaktif Amazon DevOps Guru

Wawasan proaktif DevOps Guru mendeteksi kondisi bermasalah yang diketahui pada klaster DB Aurora MySQL sebelum masalah tersebut terjadi. DevOps Guru dapat melakukan hal berikut:

- Mencegah banyak masalah umum pada basis data dengan memeriksa silang konfigurasi basis data terhadap pengaturan umum yang direkomendasikan.
- Memberi tahu Anda tentang masalah kritis dalam armada yang, jika dibiarkan tanpa diperiksa, dapat menyebabkan masalah yang lebih besar di kemudian hari.
- Memberi tahu Anda tentang masalah yang baru ditemukan.

Setiap wawasan proaktif berisi analisis penyebab masalah dan rekomendasi untuk tindakan korektif.

Topik

- Panjang daftar riwayat InnoDB meningkat secara signifikan
- Basis data membuat tabel sementara pada disk

Panjang daftar riwayat InnoDB meningkat secara signifikan

Mulai tanggal *`date`*, daftar riwayat Anda untuk perubahan baris meningkat secara signifikan, hingga *`length`* pada *`db-instance`*. Peningkatan ini memengaruhi performa penonaktifan kueri dan basis data.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab masalah ini
- Tindakan
- Metrik terkait

Versi mesin yang didukung

Informasi wawasan ini didukung untuk semua versi Aurora MySQL.

Konteks

Sistem transaksi InnoDB mempertahankan kontrol konkurensi multiversi (MVCC). Ketika baris diubah, versi pra-modifikasi dari data yang diubah disimpan sebagai catatan undo dalam log undo. Setiap data undo memiliki referensi ke data redo sebelumnya, yang membentuk daftar tertaut.

Daftar riwayat InnoDB adalah daftar global log undo untuk transaksi yang dilakukan. MySQL menggunakan daftar riwayat untuk membersihkan data dan halaman log saat transaksi tidak lagi memerlukan riwayat. Panjang daftar riwayat adalah jumlah total log undo yang berisi perubahan dalam daftar riwayat. Setiap log berisi satu atau beberapa modifikasi. Jika panjang daftar riwayat InnoDB bertambah terlalu besar, yang menunjukkan sejumlah besar versi baris lama, kueri dan penonaktifan basis data menjadi lebih lambat.

Kemungkinan penyebab masalah ini

Penyebab umum dari daftar riwayat panjang meliputi:

- Transaksi yang berjalan lama, baik baca atau tulis
- Beban tulis yang berat

Tindakan

Kami merekomendasikan tindakan yang berbeda bergantung pada penyebab wawasan Anda.

Topik

- Jangan memulai operasi apa pun yang melibatkan penonaktifan basis data hingga daftar riwayat InnoDB berkurang
- Mengidentifikasi dan mengakhiri transaksi yang berjalan lama
- Mengidentifikasi host teratas dan pengguna teratas dengan menggunakan Wawasan Performa.

Jangan memulai operasi apa pun yang melibatkan penonaktifan basis data hingga daftar riwayat InnoDB berkurang

Karena daftar riwayat InnoDB yang panjang memperlambat penonaktifan basis data, kurangi ukuran daftar sebelum memulai operasi yang melibatkan penonaktifan basis data. Operasi ini meliputi upgrade basis data versi utama.

Mengidentifikasi dan mengakhiri transaksi yang berjalan lama

Anda dapat menemukan transaksi yang berjalan lama dengan mengkueri `information_schema.innodb_trx`.

> **Note**
>
> Pastikan juga untuk mencari transaksi jangka panjang pada replika baca.

Untuk mengidentifikasi dan mengakhiri transaksi yang berjalan lama

1. Di klien SQL Anda, jalankan kueri berikut:

```
SELECT a.trx_id,
      a.trx_state,
      a.trx_started,
      TIMESTAMPDIFF(SECOND,a.trx_started, now()) as "Seconds Transaction Has Been
 Open",
      a.trx_rows_modified,
      b.USER,
      b.host,
      b.db,
      b.command,
      b.time,
      b.state
FROM  information_schema.innodb_trx a,
      information_schema.processlist b
WHERE a.trx_mysql_thread_id=b.id
  AND TIMESTAMPDIFF(SECOND,a.trx_started, now()) > 10
ORDER BY trx_started
```

2. Akhiri setiap transaksi yang berjalan lama dengan perintah `ROLLBACK` atau `COMMIT`.

Mengidentifikasi host teratas dan pengguna teratas dengan menggunakan Wawasan Performa.

Optimalkan transaksi agar sejumlah besar baris yang dimodifikasi segera dilakukan.

Metrik terkait

Metrik berikut terkait dengan wawasan ini:

- trx_rseg_history_len

  Untuk informasi selengkapnya, lihat Tabel Metrik INFORMATION_SCHEMA InnoDB di Panduan Referensi MySQL 5.7.

Basis data membuat tabel sementara pada disk

Penggunaan tabel sementara pada disk Anda baru-baru ini meningkat secara signifikan, hingga *`percentage`*. Basis data membuat sekitar *`number`* tabel sementara per detik. Hal ini dapat memengaruhi performa dan meningkatkan operasi disk pada *`db-instance`*.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab masalah ini
- Tindakan
- Metrik terkait

Versi mesin yang didukung

Informasi wawasan ini didukung untuk semua versi Aurora MySQL.

Konteks

Terkadang server MySQL perlu membuat tabel sementara internal saat memproses kueri. Aurora MySQL dapat menyimpan tabel sementara internal dalam memori, tempat tabel dapat diproses oleh mesin penyimpanan TempTable atau MEMORY, atau disimpan pada disk oleh InnoDB. Untuk informasi selengkapnya, lihat Penggunaan Tabel Sementara Internal di MySQL di Panduan Referensi MySQL.

Kemungkinan penyebab masalah ini

Peningkatan tabel sementara pada disk menunjukkan penggunaan kueri yang kompleks. Jika memori yang dikonfigurasi tidak cukup untuk menyimpan tabel sementara dalam memori, Aurora MySQL akan membuat tabel pada disk. Hal ini dapat memengaruhi performa dan meningkatkan operasi disk.

Tindakan

Kami merekomendasikan tindakan yang berbeda bergantung pada penyebab wawasan Anda.

- Untuk Aurora MySQL versi 3, sebaiknya Anda menggunakan mesin penyimpanan TempTable.
- Optimalkan kueri Anda untuk menampilkan lebih sedikit data dengan memilih kolom yang diperlukan saja.

  Jika Anda mengaktifkan Skema Performa dengan semua `statement` instrumen diaktifkan dan waktunya diatur, Anda dapat mengkueri `SYS.statements_with_temp_tables` untuk mengambil daftar kueri yang menggunakan tabel sementara. Untuk informasi selengkapnya, lihat Prasyarat untuk Menggunakan Skema sys dalam dokumentasi MySQL.
- Pertimbangkan untuk mengindeks kolom yang digunakan dalam operasi pengurutan dan pengelompokan.
- Tulis ulang kueri Anda untuk menghindari kolom `BLOB` dan `TEXT`. Kolom ini selalu menggunakan disk.
- Setel parameter basis data berikut: `tmp_table_size` dan`max_heap_table_size`.

  Nilai default untuk parameter ini adalah 16 MiB. Saat menggunakan mesin penyimpanan MEMORY untuk tabel sementara dalam memori, ukuran maksimumnya ditentukan oleh nilai `tmp_table_size` atau `max_heap_table_size`, mana yang lebih kecil. Jika ukuran maksimum ini tercapai, MySQL secara otomatis mengubah tabel sementara internal dalam memori ke tabel sementara internal pada disk InnoDB. Untuk informasi selengkapnya, lihat Menggunakan mesin penyimpanan TempTable di Amazon RDS for MySQL dan Amazon Aurora MySQL.

  > **Note**
  >
  > Jika tabel MEMORY dibuat secara eksplisit dengan CREATE TABLE, hanya variabel `max_heap_table_size` yang menentukan seberapa besar tabel bisa berkembang. Konversi ke format pada disk juga tidak terjadi.

Metrik terkait

Metrik Wawasan Performa berikut terkait dengan wawasan ini:

- Created_tmp_disk_tables
- Created_tmp_tables

Untuk informasi selengkapnya, lihat Created_tmp_disk_tables dalam dokumentasi MySQL.

Bekerja dengan kueri paralel untuk Amazon Aurora MySQL

Topik ini menjelaskan pengoptimalan kinerja kueri paralel untuk Edisi yang Kompatibel dengan Amazon Aurora MySQL. Fitur ini menggunakan jalur pemrosesan khusus untuk kueri tertentu yang sarat data, dengan memanfaatkan arsitektur penyimpanan bersama Aurora. Kueri paralel sangat cocok digunakan dengan klaster DB Aurora MySQL yang memiliki tabel dengan jutaan baris dan kueri analitik yang memerlukan waktu beberapa menit atau jam untuk diselesaikan.

Daftar Isi

- Ikhtisar kueri paralel untuk Aurora MySQL
  - Manfaat
  - Arsitektur
  - Prasyarat
  - Pembatasan
  - Biaya I/O dengan kueri paralel
- Merencanakan klaster kueri paralel
  - Memeriksa kompatibilitas versi Aurora MySQL untuk kueri paralel
- Membuat klaster DB yang bekerja dengan kueri paralel
  - Membuat klaster kueri paralel menggunakan konsol
  - Membuat klaster kueri paralel menggunakan CLI
- Mengaktifkan dan menonaktifkan kueri paralel
  - Mengaktifkan hash join untuk klaster kueri paralel
  - Mengaktifkan dan menonaktifkan kueri paralel menggunakan konsol
  - Mengaktifkan dan menonaktifkan kueri paralel menggunakan CLI
  - Menimpa pengoptimal kueri paralel
- Pertimbangan peningkatan untuk kueri paralel
  - Meningkatkan klaster kueri paralel ke Aurora MySQL versi 3
  - Peningkatan ke Aurora MySQL 2.09 dan yang lebih tinggi
- Penyempurnaan kinerja untuk kueri paralel
- Membuat objek skema untuk memanfaatkan kueri paralel
- Memverifikasi pernyataan mana yang menggunakan kueri paralel
- Memantau kueri paralel

- Cara kerja kueri paralel dengan konsep SQL
  - Pernyataan EXPLAIN
  - Klausa WHERE
  - Bahasa definisi data (DDL)
  - Jenis data kolom
  - Tabel yang dipartisi
  - Fungsi agregat, klausa GROUP BY, dan klausa HAVING
  - Panggilan fungsi dalam klausa WHERE
  - Klausa LIMIT
  - Operator perbandingan
  - Gabungan
  - Subkueri
  - UNION
  - Tampilan
  - Pernyataan bahasa manipulasi data (DML)
  - Transaksi dan penguncian
  - Indeks pohon B
  - Indeks pencarian teks penuh (FTS)
  - Kolom virtual
  - Mekanisme caching bawaan
  - Petunjuk pengoptimal
  - Tabel sementara MyISAM

Ikhtisar kueri paralel untuk Aurora MySQL

Kueri paralel Aurora MySQL adalah pengoptimalan yang memaralelkan beberapa I/O dan komputasi yang terlibat dalam pemrosesan kueri sarat data. Pekerjaan yang diparalelkan mencakup pengambilan baris dari penyimpanan, ekstraksi nilai kolom, dan penentuan baris yang cocok dengan ketentuan dalam klausa `WHERE` dan klausa sambungan. Pekerjaan sarat data ini didelegasikan (dalam istilah pengoptimalan basis data, ditekan) ke beberapa simpul di lapisan penyimpanan terdistribusi Aurora. Tanpa kueri paralel, setiap kueri membawa semua data yang dipindai ke simpul

tunggal dalam klaster Aurora MySQL (simpul kepala) dan melakukan semua pemrosesan kueri di sana.

> **Tip**
>
> Mesin basis data PostgreSQL juga memiliki fitur yang disebut “kueri paralel.” Fitur tersebut tidak terkait dengan kueri paralel Aurora.

Saat fitur kueri paralel diaktifkan, mesin Aurora MySQL secara otomatis menentukan kapan kueri dapat menguntungkan, tanpa memerlukan perubahan SQL seperti petunjuk atau atribut tabel. Pada bagian berikut, Anda dapat menemukan penjelasan tentang kapan kueri paralel diterapkan pada kueri. Anda juga dapat menemukan cara memastikan bahwa kueri paralel diterapkan di tempat yang memberi banyak keuntungan.

> **Note**
>
> Pengoptimalan kueri paralel sangat menguntungkan bagi kueri yang berlangsung lama yang memerlukan waktu beberapa menit atau jam untuk diselesaikan. Aurora MySQL umumnya tidak melakukan pengoptimalan kueri paralel untuk kueri yang murah. Aurora juga umumnya tidak melakukan pengoptimalan kueri paralel jika teknik pengoptimalan lain lebih masuk akal, seperti query caching, buffer pool caching, atau pencarian indeks. Jika Anda mendapati kueri paralel tidak digunakan seperti yang Anda harapkan, lihat Memverifikasi pernyataan mana yang menggunakan kueri paralel.

Topik

- Manfaat
- Arsitektur
- Prasyarat
- Pembatasan
- Biaya I/O dengan kueri paralel

Manfaat

Dengan kueri paralel, Anda dapat menjalankan kueri analitik sarat data pada tabel Aurora MySQL. Dalam banyak kasus, Anda dapat memperoleh peningkatan kinerja yang jauh lebih besar melalui pemisahan tugas untuk pemrosesan kueri.

Manfaat kueri paralel antara lain:

- Meningkatkan kinerja I/O, karena paralelisasi permintaan baca fisik di beberapa simpul penyimpanan.
- Mengurangi lalu lintas jaringan. Aurora tidak mengirim keseluruhan halaman data dari simpul penyimpanan ke simpul kepala lalu memfilter baris dan kolom yang tidak diperlukan sesudahnya. Sebaliknya, Aurora mengirim urutan ringkas yang hanya berisi nilai kolom yang diperlukan untuk set hasil.
- Mengurangi penggunaan CPU pada simpul kepala, karena menekan pemrosesan fungsi, penyaringan baris, dan proyeksi kolom untuk klausa `WHERE`.
- Menurunkan tekanan memori pada pool buffer. Halaman yang diproses oleh kueri paralel tidak ditambahkan ke dalam pool buffer. Pendekatan ini mengurangi peluang pemindaian sarat data mengosongkan data yang sering digunakan dari pool buffer.
- Berpotensi mengurangi duplikasi data dalam alur ekstrak, transformasi, muat (ETL) Anda, dengan memudahkan pelaksanaan kueri analitik yang berlangsung lama pada data yang sudah ada.

Arsitektur

Fitur kueri paralel menggunakan prinsip arsitektur utama Aurora MySQL: memisahkan mesin basis data dari subsistem penyimpanan, dan mengurangi lalu lintas jaringan dengan menyederhanakan protokol komunikasi. Aurora MySQL menggunakan teknik ini untuk mempercepat operasi sarat penulisan seperti pemrosesan log redo. Kueri paralel menerapkan prinsip yang sama untuk operasi baca.

> **Note**
>
> Arsitektur kueri paralel Aurora MySQL berbeda dari arsitektur dengan nama fitur yang mirip di sistem basis data lain. Kueri paralel Aurora MySQL tidak mencakup multipemrosesan simetris (SMP) sehingga tidak bergantung pada kapasitas CPU server basis data. Pemrosesan paralel terjadi di lapisan penyimpanan, terpisah dari server Aurora MySQL yang berfungsi sebagai koordinator kueri.

Secara default, tanpa kueri paralel, pemrosesan untuk kueri Aurora mencakup pengiriman data mentah ke satu simpul di dalam klaster Aurora (simpul kepala). Aurora kemudian melakukan semua pemrosesan lebih lanjut untuk kueri tersebut di dalam thread tunggal pada simpul tunggal tersebut. Dengan kueri paralel, banyak pekerjaan yang sarat I/O dan CPU ini didelegasikan ke simpul dalam lapisan penyimpanan. Hanya baris ringkas set hasil yang dikirim kembali ke simpul kepala, dengan baris yang sudah difilter, dan nilai kolom sudah diekstraksi dan ditransformasikan. Kinerja yang baik berasal dari pengurangan lalu lintas jaringan, pengurangan penggunaan CPU di simpul kepala, dan paralelisasi I/O di seluruh simpul penyimpanan. Jumlah I/O paralel, pemfilteran, dan proyeksi tidak berhubungan dengan jumlah instans DB dalam klaster Aurora yang menjalankan kueri.

Prasyarat

Untuk menggunakan semua fitur kueri paralel diperlukan klaster DB Aurora MySQL yang menjalankan versi 2.09 atau lebih tinggi. Jika Anda sudah memiliki klaster yang ingin Anda gunakan dengan kueri paralel, Anda dapat meningkatkannya ke versi yang kompatibel dan mengaktifkan kueri paralel sesudahnya. Dalam hal ini, pastikan untuk mengikuti prosedur peningkatan dalam Pertimbangan peningkatan untuk kueri paralel karena nama pengaturan konfigurasi dan nilai default berbeda di versi yang lebih baru ini.

Instans DB dalam klaster Anda harus menggunakan kelas instans `db.r*`.

Pastikan bahwa pengoptimalan hash join diaktifkan untuk klaster Anda. Untuk mempelajari caranya, lihat Mengaktifkan hash join untuk klaster kueri paralel.

Untuk menyesuaikan parameter seperti `aurora_parallel_query` dan `aurora_disable_hash_join`, Anda harus memiliki grup parameter kustom yang Anda gunakan untuk klaster Anda. Anda dapat menentukan parameter ini secara terpisah untuk setiap instans DB menggunakan grup parameter DB. Namun, kami menyarankan Anda untuk menetapkannya dalam grup parameter klaster DB. Dengan demikian, semua instans DB di klaster Anda menerima pengaturan yang sama untuk parameter ini.

Pembatasan

Batasan berikut berlaku untuk fitur kueri paralel:

- Kueri paralel tidak didukung dengan konfigurasi penyimpanan klaster DB Aurora I/O-Optimized.
- Anda tidak dapat menggunakan kueri paralel dengan kelas instans db.t2 atau db.t3. Batasan ini berlaku bahkan jika Anda meminta kueri paralel menggunakan variabel session `aurora_pq_force`.

- Kueri paralel tidak berlaku pada tabel yang menggunakan format baris `COMPRESSED` atau `REDUNDANT`. Gunakan format baris `COMPACT` atau `DYNAMIC` untuk tabel yang akan Anda gunakan dengan kueri paralel.
- Aurora menggunakan algoritma berbasis biaya untuk menentukan apakah akan menggunakan mekanisme kueri paralel untuk setiap pernyataan SQL. Menggunakan konsep SQL tertentu dalam pernyataan dapat mencegah kueri paralel atau menjadikan kueri paralel tidak memungkinkan untuk pernyataan tersebut. Untuk informasi tentang kompatibilitas konsep SQL dengan kueri paralel, lihat Cara kerja kueri paralel dengan konsep SQL.
- Setiap instans DB Aurora dapat menjalankan sesi kueri paralel hanya dalam jumlah tertentu pada satu waktu. Jika kueri memiliki beberapa bagian yang menggunakan kueri paralel, seperti subkueri, sambungan, atau operator `UNION`, fase-fase tersebut berjalan secara berurutan. Pernyataan ini hanya dihitung sebagai satu sesi kueri paralel pada satu waktu. Anda dapat memantau jumlah sesi aktif menggunakan variabel status kueri paralel. Anda dapat memeriksa batas sesi serentak untuk instans DB tertentu dengan melakukan kueri variabel status `Aurora_pq_max_concurrent_requests`.
- Kueri paralel tersedia di semua Wilayah AWS yang didukung Aurora. Untuk sebagian besar Wilayah AWS, versi Aurora MySQL minimum yang diperlukan untuk menggunakan kueri paralel adalah 2.09.
- Kueri paralel dirancang untuk meningkatkan kinerja kueri sarat data. Ini tidak dirancang untuk kueri ringan.
- Kami menyarankan Anda menggunakan simpul pembaca untuk pernyataan SELECT, terutama yang sarat data.

Biaya I/O dengan kueri paralel

Jika klaster Aurora MySQL Anda menggunakan kueri paralel, Anda mungkin melihat peningkatan nilai `VolumeReadIOPS`. Kueri paralel tidak menggunakan kumpulan buffer. Jadi, meskipun kueri cepat, pemrosesan yang dioptimalkan ini dapat menghasilkan peningkatan operasi baca dan biaya terkait.

Biaya I/O kueri paralel untuk kueri Anda diukur pada lapisan penyimpanan, dan akan sama atau lebih besar dengan saat kueri paralel diaktifkan. Anda akan mendapat manfaat berupa peningkatan kinerja kueri. Ada dua alasan untuk biaya I/O yang berpotensi lebih tinggi dengan kueri paralel:

- Bahkan jika beberapa data dalam tabel berada di pool buffer, kueri paralel mengharuskan semua data dipindai di lapisan penyimpanan, sehingga menimbulkan biaya I/O.

- Menjalankan kueri paralel tidak menyiapkan pool buffer. Akibatnya, menjalankan kueri paralel yang sama secara berurutan menimbulkan biaya I/O penuh.

Merencanakan klaster kueri paralel

Perencanaan untuk klaster DB dengan kueri paralel yang diaktifkan mengharuskan beberapa pilihan. Termasuk di antaranya adalah melakukan langkah penyiapan (baik membuat maupun memulihkan klaster Aurora MySQL penuh) dan memutuskan seberapa luas pengaktifan kueri paralel di seluruh klaster DB Aurora Anda.

Pertimbangkan hal berikut sebagai bagian dari perencanaan:

- Jika Anda menggunakan Aurora MySQL yang kompatibel dengan MySQL 5.7, Anda harus memilih Aurora MySQL 2.09 atau versi lebih tinggi. Dalam hal ini, Anda selalu membuat klaster yang disediakan. Kemudian Anda mengaktifkan kueri paralel menggunakan parameter `aurora_parallel_query`.

  Jika Anda memiliki klaster Aurora MySQL yang menjalankan versi 2.09 atau lebih tinggi, Anda tidak perlu membuat klaster baru untuk menggunakan kueri paralel. Anda dapat mengaitkan klaster Anda, atau instans DB tertentu dalam klaster, dengan grup parameter yang mengaktifkan parameter `aurora_parallel_query`. Dengan melakukannya, Anda dapat mengurangi waktu dan upaya untuk mengatur data yang relevan untuk digunakan dengan kueri paralel.
- Rencanakan setiap tabel besar yang perlu Anda atur ulang agar Anda dapat menggunakan kueri paralel saat mengaksesnya. Anda mungkin perlu membuat versi baru dari beberapa tabel besar di mana kueri paralel akan bermanfaat. Misalnya, Anda mungkin perlu menghapus indeks pencarian teks penuh. Untuk detailnya, lihat Membuat objek skema untuk memanfaatkan kueri paralel.

Memeriksa kompatibilitas versi Aurora MySQL untuk kueri paralel

Untuk memeriksa Aurora MySQL versi mana yang kompatibel dengan klaster kueri paralel, gunakan perintah `describe-db-engine-versions` AWS CLI dan periksa nilai bidang `SupportsParallelQuery`. Contoh kode berikut menunjukkan cara memeriksa kombinasi mana yang tersedia untuk klaster kueri paralel di Wilayah AWS tertentu. Pastikan untuk menentukan string parameter `--query` secara lengkap pada satu baris.

```
aws rds describe-db-engine-versions --region us-east-1 --engine aurora-mysql \
--query '*[]|[?SupportsParallelQuery == `true`].[EngineVersion]' --output text
```

Perintah sebelumnya menghasilkan output yang serupa dengan yang berikut ini. Output dapat bervariasi tergantung pada Aurora MySQL versi mana yang tersedia di Wilayah AWS tertentu.

```
5.7.mysql_aurora.2.11.1
8.0.mysql_aurora.3.01.0
8.0.mysql_aurora.3.01.1
8.0.mysql_aurora.3.02.0
8.0.mysql_aurora.3.02.1
8.0.mysql_aurora.3.02.2
8.0.mysql_aurora.3.03.0
```

Setelah mulai menggunakan kueri paralel dengan suatu klaster, Anda dapat memantau kinerja dan menghilangkan kendala pada penggunaan kueri paralel. Untuk instruksi langkah demi langkah, lihat Penyempurnaan kinerja untuk kueri paralel.

Membuat klaster DB yang bekerja dengan kueri paralel

Untuk membuat klaster Aurora MySQL dengan kueri paralel, menambahkan instans baru ke dalamnya, atau melakukan operasi administratif lainnya, Anda menggunakan teknik AWS Management Console dan AWS CLI yang sama seperti yang Anda lakukan dengan klaster Aurora MySQL lainnya. Anda dapat membuat klaster baru untuk bekerja dengan kueri paralel. Anda juga dapat membuat klaster DB untuk bekerja dengan kueri paralel dengan memulihkan dari snapshot klaster DB Aurora yang kompatibel dengan MySQL. Jika Anda tidak memahami proses pembuatan klaster Aurora MySQL, Anda dapat menemukan informasi latar belakang dan prasyaratnya dalam Membuat klaster DB Amazon Aurora.

Saat Anda memilih versi mesin Aurora MySQL, sebaiknya Anda memilih versi terbaru. Saat ini, Aurora MySQL versi 2.09 dan lebih tinggi mendukung kueri paralel. Anda memiliki fleksibilitas lebih besar untuk mengaktifkan dan menonaktifkan kueri paralel, atau menggunakan kueri paralel dengan klaster yang ada, jika Anda menggunakan Aurora MySQL 2.09 dan versi lebih tinggi.

Dengan membuat klaster baru atau memulihkan dari snapshot, Anda menggunakan teknik yang sama untuk menambahkan instans DB baru yang Anda lakukan dengan klaster Aurora MySQL lainnya.

Membuat klaster kueri paralel menggunakan konsol

Anda dapat membuat klaster kueri paralel baru dengan konsol seperti dijelaskan berikut ini.

Untuk membuat klaster kueri paralel dengan AWS Management Console

1. Ikuti prosedur AWS Management Console umum dalam Membuat klaster DB Amazon Aurora.
2. Pada layar Pilih mesin, pilih Aurora MySQL.

   Untuk Versi mesin, pilih Aurora MySQL 2.09 atau lebih tinggi. Dengan versi ini, Anda memiliki batasan paling sedikit pada penggunaan kueri paralel. Versi tersebut juga memiliki fleksibilitas terbaik untuk mengaktifkan atau menonaktifkan kueri paralel kapan saja.

   Jika tidak memungkinkan untuk menggunakan Aurora MySQL versi terbaru untuk klaster ini, pilih Tampilkan versi yang mendukung fitur kueri paralel. Melakukan hal tersebut akan memfilter menu Versi untuk menampilkan hanya versi Aurora MySQL khusus yang kompatibel dengan kueri paralel.
3. Untuk Konfigurasi tambahan, pilih grup parameter yang Anda buat untuk grup parameter klaster DB. Penggunaan grup parameter kustom tersebut diperlukan untuk Aurora MySQL 2.09 dan yang lebih tinggi. Di grup parameter klaster DB Anda, tentukan pengaturan parameter `aurora_parallel_query=ON` dan `aurora_disable_hash_join=OFF`. Tindakan tersebut akan mengaktifkan kueri paralel untuk klaster, dan mengaktifkan pengoptimalan hash join yang bekerja bersama kueri paralel.

Untuk memverifikasi bahwa klaster baru dapat menggunakan kueri paralel

1. Buat klaster menggunakan teknik sebelumnya.
2. (Untuk Aurora MySQL versi 2 atau 3) Pastikan pengaturan konfigurasi `aurora_parallel_query` true.

```
mysql> select @@aurora_parallel_query;
+-------------------------+
| @@aurora_parallel_query |
+-------------------------+
|                       1 |
+-------------------------+
```

3. (Untuk Aurora MySQL versi 2) Pastikan pengaturan `aurora_disable_hash_join` false.

```
mysql> select @@aurora_disable_hash_join;
+---------------------------+
| @@aurora_disable_hash_join |
+---------------------------+
```

```
|                           0 |
+-----------------------------+
```

4. Dengan beberapa tabel besar dan kueri sarat data, periksa rencana kueri untuk mengonfirmasi bahwa beberapa kueri Anda menggunakan pengoptimalan kueri paralel. Untuk melakukannya, ikuti prosedur dalam Memverifikasi pernyataan mana yang menggunakan kueri paralel.

Membuat klaster kueri paralel menggunakan CLI

Anda dapat membuat klaster kueri paralel baru dengan CLI seperti yang dijelaskan berikut ini.

Untuk membuat klaster kueri paralel dengan AWS CLI

1. (Opsional) Perika versi Aurora MySQL mana yang kompatibel dengan klaster kueri paralel. Untuk melakukannya, gunakan perintah `describe-db-engine-versions` dan periksa nilai bidang `SupportsParallelQuery`. Sebagai contoh, lihat Memeriksa kompatibilitas versi Aurora MySQL untuk kueri paralel.
2. (Opsional) Buat grup parameter klaster DB kustom dengan pengaturan `aurora_parallel_query=ON` dan `aurora_disable_hash_join=OFF`. Gunakan perintah seperti berikut ini.

```
aws rds create-db-cluster-parameter-group --db-parameter-group-family aurora-
mysql5.7 --db-cluster-parameter-group-name pq-enabled-57-compatible
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-57-compatible \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name pq-
enabled-57-compatible \
  --parameters
 ParameterName=aurora_disable_hash_join,ParameterValue=OFF,ApplyMethod=pending-
reboot
```

   Jika Anda melakukan langkah ini, tentukan opsi `--db-cluster-parameter-group-name` *`my_cluster_parameter_group`* dalam pernyataan `create-db-cluster` selanjutnya. Ganti nama grup parameter Anda. Jika menghapus langkah ini, Anda membuat grup parameter dan mengaitkannya dengan klaster di lain waktu, seperti dijelaskan di Mengaktifkan dan menonaktifkan kueri paralel.
3. Ikuti prosedur AWS CLI umum dalam Membuat klaster DB Amazon Aurora.

4. Tentukan rangkaian opsi berikut:

   - Untuk opsi `--engine`, gunakan `aurora-mysql`. Nilai ini menghasilkan klaster kueri paralel yang kompatibel dengan MySQL 5.7 atau 8.0.
   - Untuk pilihan `--db-cluster-parameter-group-name`, tentukan nama grup parameter klaster DB yang Anda buat dan tentukan nilai parameter `aurora_parallel_query=ON`. Jika Anda menghilangkan pilihan ini, Anda dapat membuat klaster dengan kelompok parameter default, kemudian memodifikasinya untuk menggunakan grup parameter khusus tersebut.
   - Untuk pilihan `--engine-version`, gunakan versi Aurora MySQL yang kompatibel dengan kueri paralel. Gunakan prosedur dari Merencanakan klaster kueri paralel untuk mendapatkan daftar versi jika diperlukan. Gunakan setidaknya versi 2.09.0. Versi ini dan semua versi yang lebih tinggi berisi peningkatan substansial untuk kueri paralel.

     Contoh kode berikut menunjukkan caranya. Ganti nilai Anda sendiri untuk setiap variabel lingkungan seperti *`$CLUSTER_ID`*. Contoh ini juga menentukan opsi `--manage-master-user-password` untuk menghasilkan kata sandi pengguna master dan mengelolanya di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. Alternatifnya, Anda dapat menggunakan opsi `--master-password` untuk menentukan dan mengelola kata sandi sendiri.

```
aws rds create-db-cluster --db-cluster-identifier $CLUSTER_ID \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --master-username $MASTER_USER_ID --manage-master-user-password \
  --db-cluster-parameter-group-name $CUSTOM_CLUSTER_PARAM_GROUP

aws rds create-db-instance --db-instance-identifier ${INSTANCE_ID}-1 \
  --engine same_value_as_in_create_cluster_command \
  --db-cluster-identifier $CLUSTER_ID --db-instance-class $INSTANCE_CLASS
```

5. Verifikasi bahwa klaster yang Anda buat atau pulihkan memiliki fitur kueri paralel yang tersedia.

   Periksa apakah pengaturan konfigurasi `aurora_parallel_query` ada. Jika pengaturan ini memiliki nilai 1, kueri paralel siap Anda gunakan. Jika pengaturan ini memiliki nilai 0, atur menjadi 1 sebelum Anda dapat menggunakan kueri paralel. Berapa pun nilainya, klaster ini mampu melakukan kueri paralel.

```
mysql> select @@aurora_parallel_query;
+------------------------+
| @@aurora_parallel_query|
```

```
+------------------------+
|                      1 |
+------------------------+
```

Untuk memulihkan snapshot ke klaster kueri paralel dengan AWS CLI

1. Periksa versi Aurora MySQL yang mana yang sesuai dengan klaster kueri paralel. Untuk melakukannya, gunakan perintah `describe-db-engine-versions` dan periksa nilai bidang `SupportsParallelQuery`. Sebagai contoh, lihat Memeriksa kompatibilitas versi Aurora MySQL untuk kueri paralel. Tentukan versi yang akan digunakan untuk klaster yang dipulihkan. Pilih Aurora MySQL 2.09.0 atau lebih tinggi untuk klaster yang kompatibel dengan MySQL 5.7.
2. Temukan snapshot klaster yang kompatibel dengan Aurora MySQL.
3. Ikuti prosedur AWS CLI umum dalam Memulihkan dari snapshot klaster DB.

   ```
   aws rds restore-db-cluster-from-snapshot \
     --db-cluster-identifier mynewdbcluster \
     --snapshot-identifier mydbclustersnapshot \
     --engine aurora-mysql
   ```

4. Verifikasi bahwa klaster yang Anda buat atau pulihkan memiliki fitur kueri paralel yang tersedia. Gunakan prosedur verifikasi yang sama seperti di Membuat klaster kueri paralel menggunakan CLI.

Mengaktifkan dan menonaktifkan kueri paralel

Saat kueri paralel diaktifkan, Aurora MySQL menentukan apakah akan menggunakannya saat runtime untuk setiap kueri. Dalam hal sambungan, paduan, subkueri, dan sebagainya, Aurora MySQL menentukan apakah akan menggunakan kueri paralel pada runtime untuk setiap blok kueri. Untuk detailnya, lihat Memverifikasi pernyataan mana yang menggunakan kueri paralel dan Cara kerja kueri paralel dengan konsep SQL.

Anda dapat mengaktifkan dan menonaktifkan kueri paralel secara dinamis pada tingkat global dan sesi untuk instans DB menggunakan opsi aurora_parallel_query. Anda dapat mengubah pengaturan `aurora_parallel_query` di grup klaster DB Anda untuk mengaktifkan atau menonaktifkan kueri paralel secara default.

```
mysql> select @@aurora_parallel_query;
```

```
+------------------------+
| @@aurora_parallel_query|
+------------------------+
|                      1 |
+------------------------+
```

Untuk mengganti parameter `aurora_parallel_query` pada tingkat sesi, gunakan metode standar untuk mengubah pengaturan konfigurasi klien. Contohnya, Anda dapat melakukannya melalui baris perintah `mysql` atau dalam aplikasi JDBC atau ODBC. Perintah pada klien MySQL standar adalah `set session aurora_parallel_query = {'ON'/'OFF'}`. Anda juga dapat menambahkan parameter tingkat sesi ke konfigurasi JDBC atau dalam kode aplikasi Anda untuk mengaktifkan atau menonaktifkan kueri paralel secara dinamis.

Anda dapat secara permanen mengubah pengaturan untuk parameter `aurora_parallel_query`, baik untuk instans DB tertentu atau untuk seluruh klaster Anda. Jika Anda menetapkan nilai parameter dalam grup parameter DB, nilai tersebut hanya berlaku untuk instans DB tertentu dalam klaster Anda. Jika Anda menetapkan nilai parameter dalam grup parameter klaster DB, semua instans DB di klaster tersebut menerima pengaturan yang sama. Untuk mengganti parameter `aurora_parallel_query`, gunakan teknik untuk bekerja dengan grup parameter, seperti dijelaskan dalam Bekerja dengan grup parameter. Ikuti langkah-langkah ini:

1. Buat grup parameter klaster kustom (disarankan) atau grup parameter DB kustom.
2. Dalam grup parameter ini, perbarui `parallel_query` dengan nilai yang Anda inginkan.
3. Bergantung pada apakah Anda membuat grup parameter klaster DB atau grup parameter DB, pasang grup parameter ke klaster Aurora Anda atau ke instans DB spesifik tempat Anda berencana menggunakan fitur kueri paralel.

> **Tip**
>
> Karena `aurora_parallel_query` adalah parameter dinamis, parameter ini tidak memerlukan mulai ulang klaster setelah pengaturan ini diubah. Namun, koneksi apa pun yang menggunakan kueri paralel sebelum beralih opsi akan terus melakukannya sampai koneksi ditutup, atau instans di-boot ulang.

Anda dapat mengubah parameter kueri paralel dengan menggunakan operasi API ModifyDBClusterParameterGroup atau ModifyDBParameterGroup atau AWS Management Console.

Mengaktifkan hash join untuk klaster kueri paralel

Kueri paralel biasanya digunakan untuk jenis kueri sarat sumber daya yang diuntungkan dari pengoptimalan hash join. Oleh karena itu, sebaiknya pastikan hash join diaktifkan untuk klaster yang Anda rencanakan akan menggunakan kueri paralel. Untuk informasi tentang cara menggunakan hash join secara efektif, lihat Mengoptimalkan kueri join MySQL Aurora besar dengan hash join.

Mengaktifkan dan menonaktifkan kueri paralel menggunakan konsol

Anda dapat mengaktifkan atau menonaktifkan kueri paralel pada tingkat instans DB atau tingkat klaster DB dengan menggunakan grup parameter.

Untuk mengaktifkan atau menonaktifkan kueri paralel untuk klaster DB dengan AWS Management Console

1. Buat grup parameter khusus, seperti yang dijelaskan di Bekerja dengan grup parameter.
2. Perbarui aurora_parallel_query ke 1 (diaktifkan) atau 0 (dinonaktifkan). Untuk klaster dengan fitur kueri paralel yang tersedia, aurora_parallel_query dinonaktifkan secara default.
3. Jika Anda menggunakan grup parameter klaster kustom, lampirkan grup tersebut ke klaster Aurora DB tempat Anda berencana menggunakan fitur kueri paralel. Jika Anda menggunakan grup parameter DB kustom, lampirkan grup tersebut ke satu atau lebih instans DB dalam klaster. Kami menyarankan menggunakan grup parameter klaster. Melakukan hal tersebut dapat memastikan semua instans DB di klaster memiliki pengaturan yang sama untuk kueri paralel dan fitur terkait seperti hash join.

Mengaktifkan dan menonaktifkan kueri paralel menggunakan CLI

Anda dapat memodifikasi parameter kueri paralel dengan menggunakan perintah `modify-db-cluster-parameter-group` atau `modify-db-parameter-group`. Pilih perintah yang sesuai, tergantung pada apakah Anda menentukan nilai `aurora_parallel_query` melalui grup parameter klaster DB atau grup parameter DB.

Untuk mengaktifkan atau menonaktifkan kueri paralel untuk klaster DB dengan CLI

- Ubah parameter kueri paralel menggunakan perintah `modify-db-cluster-parameter-group`. Gunakan perintah seperti berikut. Ganti dengan nama yang sesuai untuk grup parameter khusus Anda. Ganti `ON` atau `OFF` untuk bagian `ParameterValue` dari opsi `--parameters`.

```
$ aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters
 ParameterName=aurora_parallel_query,ParameterValue=ON,ApplyMethod=pending-reboot
{
    "DBClusterParameterGroupName": "cluster_param_group_name"
}

aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-
name cluster_param_group_name \
  --parameters ParameterName=aurora_pq,ParameterValue=ON,ApplyMethod=pending-reboot
```

Anda juga dapat mengaktifkan atau menonaktifkan kueri paralel pada tingkat sesi, misalnya melalui baris perintah `mysql` atau dalam aplikasi JDBC atau ODBC. Untuk melakukannya, gunakan metode standar untuk mengubah pengaturan konfigurasi klien. Misalnya, perintah pada klien MySQL standar adalah `set session aurora_parallel_query = {'ON'/'OFF'}` untuk Aurora MySQL.

Anda juga dapat menambahkan parameter tingkat sesi ke konfigurasi JDBC atau dalam kode aplikasi Anda untuk mengaktifkan atau menonaktifkan kueri paralel secara dinamis.

Menimpa pengoptimal kueri paralel

Anda dapat menggunakan variabel sesi `aurora_pq_force` untuk menimpa pengoptimal kueri paralel dan meminta kueri paralel untuk setiap kueri. Kami menyarankan Anda melakukan tindakan ini hanya untuk tujuan pengujian. Contoh berikut ini menunjukkan cara menggunakan `aurora_pq_force` dalam sebuah sesi.

```
set SESSION aurora_parallel_query = ON;
set SESSION aurora_pq_force = ON;
```

Untuk mematikan penimpaan, lakukan hal berikut:

```
set SESSION aurora_pq_force = OFF;
```

Pertimbangan peningkatan untuk kueri paralel

Tergantung versi asal dan tujuan saat Anda meningkatkan versi klaster kueri paralel, Anda mungkin menemukan penyempurnaan dalam jenis kueri yang dapat dioptimalkan oleh kueri paralel. Anda mungkin juga menemukan bahwa Anda tidak perlu menentukan parameter mode mesin khusus untuk

kueri paralel. Bagian berikutnya menjelaskan pertimbangan saat Anda meningkatkan klaster dengan kueri paralel yang diaktifkan.

Meningkatkan klaster kueri paralel ke Aurora MySQL versi 3

Beberapa pernyataan, klausa, dan tipe data SQL memiliki dukungan kueri paralel baru atau yang lebih baik dimulai di Aurora MySQL versi 3. Saat Anda melakukan peningkatan dari rilis sebelum versi 3, periksa apakah kueri tambahan dapat memperoleh manfaat dari pengoptimalan kueri paralel. Untuk informasi tentang penyempurnaan kueri paralel ini, lihat Jenis data kolom, Tabel yang dipartisi, dan Fungsi agregat, klausa GROUP BY, dan klausa HAVING.

Jika Anda meningkatkan klaster kueri paralel dari Aurora MySQL 2.08 atau yang lebih rendah, pelajari juga tentang perubahan cara mengaktifkan kueri paralel. Untuk mempelajarinya, baca Peningkatan ke Aurora MySQL 2.09 dan yang lebih tinggi.

Di Aurora MySQL versi 3, pengoptimalan hash join diaktifkan secara default. Opsi konfigurasi `aurora_disable_hash_join` dari versi sebelumnya tidak digunakan.

Peningkatan ke Aurora MySQL 2.09 dan yang lebih tinggi

Di Aurora MySQL versi 2.09 dan versi lebih tinggi, kueri paralel bekerja untuk klaster yang telah disediakan dan tidak memerlukan parameter mode mesin `parallelquery`. Dengan demikian, Anda tidak perlu membuat klaster baru atau memulihkan dari snapshot yang sudah ada untuk menggunakan kueri paralel dengan versi ini. Anda dapat menggunakan prosedur peningkatan yang dijelaskan dalam Meningkatkan versi minor atau tingkat patch klaster DB Aurora MySQL untuk meningkatkan klaster Anda ke versi tersebut. Anda dapat meningkatkan klaster yang lebih lama terlepas dari apakah itu berupa klaster kueri paralel atau klaster yang tersedia. Untuk mengurangi jumlah pilihan dalam Versi mesin Anda dapat memilih Tampilkan versi yang mendukung fitur kueri paralel untuk memfilter entri di menu tersebut. Lalu pilih Aurora MySQL 2.09 atau lebih tinggi.

Setelah Anda meningkatkan klaster kueri paralel sebelumnya ke Aurora MySQL versi 2.09 atau lebih tinggi, Anda mengaktifkan kueri paralel dalam klaster yang ditingkatkan. Kueri paralel dimatikan secara default dalam versi ini, dan prosedur untuk mengaktifkannya berbeda. Pengoptimalan hash join juga dinonaktifkan secara default dan harus diaktifkan secara terpisah. Oleh karena itu, pastikan Anda mengaktifkan pengaturan ini lagi setelah peningkatan. Untuk petunjuk tentang melakukannya, lihat Mengaktifkan dan menonaktifkan kueri paralel dan Mengaktifkan hash join untuk klaster kueri paralel.

Secara khusus, Anda mengaktifkan kueri paralel dengan menggunakan parameter konfigurasi `aurora_parallel_query=ON` dan `aurora_disable_hash_join=OFF`, bukan

`aurora_pq_supported` dan `aurora_pq`. Parameter `aurora_pq_supported` dan `aurora_pq` diusangkan dalam versi Aurora MySQL yang lebih baru.

Dalam klaster yang ditingkatkan, atribut `EngineMode` memiliki nilai `provisioned`, bukan `parallelquery`. Untuk memeriksa apakah kueri paralel tersedia untuk versi mesin tertentu, sekarang Anda memeriksa nilai bidang `SupportsParallelQuery` di output perintah `describe-db-engine-versions` AWS CLI. Di versi Aurora MySQL sebelumnya, Anda memeriksa keberadaan `parallelquery` dalam daftar `SupportedEngineModes`.

Setelah meningkatkan ke Aurora MySQL versi 2.09 atau lebih tinggi, Anda dapat memanfaatkan fitur-fitur berikut. Fitur ini tidak tersedia untuk klaster kueri paralel yang menjalankan Aurora MySQL versi lama.

- Wawasan Performa. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.
- Pelacakan Mundur. Untuk informasi selengkapnya, lihat Melakukan backtracking klaster DB Aurora.
- Menghentikan dan memulai klaster. Untuk informasi selengkapnya, lihat Menghentikan dan memulai klaster DB Amazon Aurora.

Penyempurnaan kinerja untuk kueri paralel

Untuk mengelola kinerja beban kerja dengan kueri paralel, pastikan kueri paralel digunakan untuk kueri di mana pengoptimalan ini paling membantu.

Untuk melakukannya, Anda dapat melakukan hal berikut:

- Pastikan tabel terbesar Anda kompatibel dengan kueri paralel. Anda mungkin mengubah properti tabel atau membuat ulang beberapa tabel sehingga kueri untuk tabel tersebut dapat memanfaatkan pengoptimalan kueri paralel. Untuk mempelajari caranya, lihat Membuat objek skema untuk memanfaatkan kueri paralel.
- Pantau kueri mana yang menggunakan kueri paralel. Untuk mempelajari caranya, lihat Memantau kueri paralel.
- Verifikasi bahwa kueri paralel digunakan untuk kueri sarat data dan berlangsung lama, dan dengan tingkat keserentakan yang tepat untuk beban kerja Anda. Untuk mempelajari caranya, lihat Memverifikasi pernyataan mana yang menggunakan kueri paralel.

- Sesuaikan kode SQL Anda untuk mengaktifkan kueri paralel agar diterapkan pada kueri yang Anda harapkan. Untuk mempelajari caranya, lihat Cara kerja kueri paralel dengan konsep SQL.

Membuat objek skema untuk memanfaatkan kueri paralel

Sebelum Anda membuat atau memodifikasi tabel yang akan Anda gunakan untuk kueri paralel, pastikan untuk memahami persyaratan yang dijelaskan dalam Prasyarat dan Pembatasan.

Karena kueri paralel memerlukan tabel untuk menggunakan pengaturan `ROW_FORMAT=Compact` atau `ROW_FORMAT=Dynamic`, periksa pengaturan konfigurasi Aurora Anda untuk setiap perubahan pada opsi konfigurasi `INNODB_FILE_FORMAT`. Keluarkan pernyataan `SHOW TABLE STATUS` untuk mengonfirmasi format baris untuk semua tabel dalam basis data.

Sebelum mengubah skema Anda untuk mengaktifkan kueri paralel agar dapat bekerja dengan lebih banyak tabel, pastikan untuk melakukan pengujian. Pengujian Anda harus mengonfirmasi apakah kueri paralel menghasilkan peningkatan kinerja bersih untuk tabel tersebut. Selain itu, pastikan bahwa persyaratan skema untuk kueri paralel kompatibel dengan sasaran Anda.

Sebagai contoh, sebelum beralih dari `ROW_FORMAT=Compressed` ke `ROW_FORMAT=Compact` atau `ROW_FORMAT=Dynamic`, uji kinerja beban kerja untuk tabel asal dan baru. Selain itu, pertimbangkan potensi efek lainnya seperti peningkatan volume data.

Memverifikasi pernyataan mana yang menggunakan kueri paralel

Dalam operasi biasa, Anda tidak perlu melakukan tindakan khusus apa pun untuk memanfaatkan kueri paralel. Setelah kueri memenuhi persyaratan penting untuk kueri paralel, pengoptimal kueri secara otomatis memutuskan apakah akan menggunakan kueri paralel untuk setiap kueri tertentu.

Jika menjalankan percobaan di lingkungan pengembangan atau pengujian, Anda mungkin menemukan bahwa kueri paralel tidak digunakan karena tabel Anda terlalu kecil dalam jumlah baris atau volume data keseluruhan. Data untuk tabel mungkin juga sepenuhnya berada di dalam buffer pool, terutama untuk tabel yang baru Anda buat untuk melakukan percobaan.

Saat Anda memantau atau menyesuaikan kinerja klaster, pastikan untuk memutuskan apakah kueri paralel digunakan dalam konteks yang sesuai. Anda dapat menyesuaikan skema basis data, pengaturan, kueri SQL, atau bahkan klaster topologi dan pengaturan koneksi aplikasi untuk memanfaatkan fitur ini.

Untuk memeriksa apakah suatu kueri menggunakan kueri paralel, periksa rencana kueri (juga dikenal sebagai "rencana menjelaskan") dengan menjalankan pernyataan EXPLAIN. Sebagai contoh

bagaimana pernyataan, klausa, dan ekspresi SQL memengaruhi output `EXPLAIN` untuk kueri paralel, lihat Cara kerja kueri paralel dengan konsep SQL.

Contoh berikut menunjukkan perbedaan antara rencana kueri tradisional dan rencana kueri paralel. Rencana explain ini dari Query 3 dari tolok ukur TPC-H. Banyak sampel kueri di seluruh bagian ini menggunakan tabel dari set data TPC-H. Anda dapat memperoleh definisi tabel, kueri, dan program `dbgen` yang menghasilkan data sampel dari situs web TPC-h.

```
EXPLAIN SELECT l_orderkey,
  sum(l_extendedprice * (1 - l_discount)) AS revenue,
  o_orderdate,
  o_shippriority
FROM customer,
  orders,
  lineitem
WHERE c_mktsegment = 'AUTOMOBILE'
AND c_custkey = o_custkey
AND l_orderkey = o_orderkey
AND o_orderdate < date '1995-03-13'
AND l_shipdate > date '1995-03-13'
GROUP BY l_orderkey,
  o_orderdate,
  o_shippriority
ORDER BY revenue DESC,
  o_orderdate LIMIT 10;
```

Secara default, kueri tersebut mungkin memiliki rencana seperti berikut. Jika Anda tidak melihat hash join digunakan dalam rencana kueri, pastikan pengoptimalan diaktifkan terlebih dahulu.

```
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows     | filtered | Extra                                              |
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
|  1 | SIMPLE      | customer | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |  1480234 |    10.00 | Using where; Using temporary; Using filesort       |
|  1 | SIMPLE      | orders   | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 14875240 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
|  1 | SIMPLE      | lineitem | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL | 59270573 |     3.33 | Using where; Using join buffer (Block Nested Loop) |
```

```
+----+-------------+----------+------------+------+---------------+------+---------
+------+----------+----------+----------------------------------------------------+
```

Untuk Aurora MySQL versi 3, Anda mengaktifkan hash join pada tingkat sesi dengan mengeluarkan pernyataan berikut.

```
SET optimizer_switch='block_nested_loop=on';
```

Untuk Aurora MySQL versi 2.09 dan yang lebih tinggi, Anda mengatur parameter DB `aurora_disable_hash_join` atau parameter klaster DB ke `0` (off). Jika `aurora_disable_hash_join` dinonaktifkan, nilai `optimizer_switch` berubah menjadi `hash_join=on`.

Setelah Anda mengaktifkan hash join, coba jalankan pernyataan `EXPLAIN` lagi. Untuk informasi tentang cara menggunakan hash join secara efektif, lihat Mengoptimalkan kueri join MySQL Aurora besar dengan hash join.

Dengan hash join diaktifkan tetapi kueri paralel dinonaktifkan, kueri mungkin memiliki rencana seperti berikut, yang menggunakan hash join, tetapi tidak menggunakan kueri paralel.

```
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
| id | select_type | table    |...| rows      | Extra
                          |
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
|  1 | SIMPLE      | customer |...|   5798330 | Using where; Using index; Using
 temporary; Using filesort       |
|  1 | SIMPLE      | orders   |...| 154545408 | Using where; Using join buffer (Hash
 Join Outer table orders)   |
|  1 | SIMPLE      | lineitem |...| 606119300 | Using where; Using join buffer (Hash
 Join Outer table lineitem) |
+----+-------------+----------+...+-----------
+-------------------------------------------------------------------+
```

Setelah kueri paralel diaktifkan, dua langkah dalam rencana kueri paralel, seperti ditunjukkan pada kolom `Extra` dalam output `EXPLAIN`. Pemrosesan sarat I/O dan CPU untuk langkah-langkah tersebut diturunkan ke lapisan penyimpanan.

```
+----+...
+-----------------------------------------------------------------------------------------------
+
| id |...| Extra
                                                    |
+----+...
+-----------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using index; Using temporary; Using filesort
                                                    |
|  1 |...| Using where; Using join buffer (Hash Join Outer table orders); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra)   |
|  1 |...| Using where; Using join buffer (Hash Join Outer table lineitem); Using
 parallel query (4 columns, 1 filters, 1 exprs; 0 extra) |
+----+...
+-----------------------------------------------------------------------------------------------
+
```

Untuk informasi tentang cara menginterpretasi output `EXPLAIN` untuk kueri paralel dan bagian pernyataan SQL yang dapat diterapkan oleh kueri paralel, lihat Cara kerja kueri paralel dengan konsep SQL.

Contoh output berikut menunjukkan hasil dari menjalankan kueri sebelumnya pada instans db.r4.2xlarge dengan IT buffer pool dingin. Kueri berjalan jauh lebih cepat saat menggunakan kueri paralel.

> **Note**
>
> Karena waktu ditentukan oleh banyak faktor lingkungan, hasil Anda mungkin berbeda. Selalu lakukan uji kinerja Anda sendiri untuk mengonfirmasi temuan dengan lingkungan dan beban kerja Anda sendiri, dan sebagainya.

```
-- Without parallel query
+------------+-------------+-------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+-------------+----------------+
|   92511430 | 514726.4896 | 1995-03-06  |              0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |              0 |
```

```
+------------+-------------+-------------+----------------+
10 rows in set (24 min 49.99 sec)
```

```
-- With parallel query
+------------+-------------+-------------+----------------+
| l_orderkey | revenue     | o_orderdate | o_shippriority |
+------------+-------------+-------------+----------------+
|   92511430 | 514726.4896 | 1995-03-06  |              0 |
.
.
|   28840519 | 454748.2485 | 1995-03-08  |              0 |
+------------+-------------+-------------+----------------+
10 rows in set (1 min 49.91 sec)
```

Banyak kueri sampel di seluruh bagian ini menggunakan tabel dari set data TPC-H ini, khususnya tabel PART, yang memiliki 20 juta baris dan definisi berikut.

```
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| p_partkey     | int(11)       | NO   | PRI | NULL    |       |
| p_name        | varchar(55)   | NO   |     | NULL    |       |
| p_mfgr        | char(25)      | NO   |     | NULL    |       |
| p_brand       | char(10)      | NO   |     | NULL    |       |
| p_type        | varchar(25)   | NO   |     | NULL    |       |
| p_size        | int(11)       | NO   |     | NULL    |       |
| p_container   | char(10)      | NO   |     | NULL    |       |
| p_retailprice | decimal(15,2) | NO   |     | NULL    |       |
| p_comment     | varchar(23)   | NO   |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
```

Lakukan eksperimen dengan beban kerja Anda untuk mengetahui apakah pernyataan SQL individual dapat memanfaatkan kueri paralel. Lalu gunakan teknik pemantauan berikut untuk membantu memverifikasi frekuensi penggunaan kueri paralel dalam beban kerja nyata dari waktu ke waktu. Untuk beban kerja nyata, faktor tambahan seperti batas keserentakan berlaku.

Memantau kueri paralel

Jika klaster Aurora MySQL Anda menggunakan kueri paralel, Anda mungkin melihat peningkatan nilai `VolumeReadIOPS`. Kueri paralel tidak menggunakan kumpulan buffer. Jadi, meskipun kueri

cepat, pemrosesan yang dioptimalkan ini dapat menghasilkan peningkatan operasi baca dan biaya terkait.

Selain metrik Amazon CloudWatch yang dijelaskan dalam Melihat metrik di konsol Amazon RDS, Aurora menyediakan variabel status global lainnya. Anda dapat menggunakan variabel status global ini untuk membantu memantau eksekusi kueri paralel. Variabel tersebut dapat memberi Anda wawasan tentang alasan pengoptimal dapat menggunakan atau tidak menggunakan kueri paralel dalam situasi tertentu. Untuk mengakses variabel ini, Anda dapat menggunakan perintah `SHOW GLOBAL STATUS`. Anda juga dapat menemukan variabel ini tercantum sebagai berikut.

Sesi kueri paralel tidak selalu berupa pemetaan satu-ke-satu dengan kueri yang dilakukan oleh basis data. Contohnya, anggaplah bahwa rencana kueri Anda memiliki dua langkah yang menggunakan kueri paralel. Dalam hal ini, kueri tersebut mencakup dua sesi paralel dan penghitung untuk permintaan yang telah diupayakan dan permintaan yang berhasil ditambah sebanyak dua.

Saat Anda bereksperimen dengan kueri paralel dengan mengeluarkan pernyataan `EXPLAIN`, bersiaplah untuk melihat kenaikan di penghitung yang ditetapkan sebagai "tidak dipilih" meskipun kueri tersebut sebenarnya tidak berjalan. Saat bekerja dengan kueri paralel dalam produksi, Anda dapat memeriksa apakah penghitung yang "tidak dipilih" meningkat lebih cepat dari yang Anda perkirakan. Pada titik ini, Anda dapat menyesuaikan sehingga kueri paralel berjalan untuk kueri yang Anda harapkan. Untuk melakukannya, Anda dapat mengubah pengaturan klaster, campuran kueri, instans DB tempat kueri paralel diaktifkan, dan sebagainya.

Penghitung ini dilacak di tingkat instans DB. Saat terhubung ke titik akhir yang berbeda, Anda mungkin melihat metrik yang berbeda karena setiap instans DB menjalankan serangkaian kueri paralelnya sendiri. Anda mungkin juga melihat metrik yang berbeda saat titik akhir pembaca terhubung ke instans DB yang berbeda untuk setiap sesi.

Nama	Deskripsi
`Aurora_pq_bytes_returned`	Jumlah byte untuk struktur data tuple yang ditransmisikan ke simpul head selama kueri paralel. Bagi dengan 16.384 untuk membandingkan dengan `Aurora_pq_pages_pushed_down` .
`Aurora_pq_max_concurrent_requests`	Jumlah maksimum sesi kueri paralel yang dapat berjalan secara konkuren di instans

Nama	Deskripsi
	Aurora DB ini. Ini adalah jumlah tetap yang bergantung pada kelas instans DB AWS.
`Aurora_pq_pages_pushed_down`	Jumlah halaman data (masing-masing dengan ukuran tetap 16 KiB) tempat kueri paralel menghindari transmisi jaringan ke simpul head.
`Aurora_pq_request_attempted`	Jumlah sesi kueri paralel yang diminta. Nilai ini dapat menunjukkan lebih dari satu sesi per kueri, bergantung pada konsep SQL seperti subkueri dan penggabungan.
`Aurora_pq_request_executed`	Jumlah sesi kueri paralel yang berhasil berjalan.
`Aurora_pq_request_failed`	Jumlah sesi kueri paralel yang menampilk an kesalahan kepada klien. Dalam beberapa kasus, permintaan untuk kueri paralel mungkin gagal, misalnya karena masalah dalam lapisan penyimpanan. Dalam kasus ini, bagian kueri yang gagal dicoba kembali menggunakan mekanisme kueri nonparalel. Jika kueri yang dicoba kembali juga gagal, kesalahan ditampilk an ke klien dan penghitung ini ditambah.
`Aurora_pq_request_in_progress`	Jumlah sesi kueri paralel yang sedang berlangsung. Angka ini berlaku untuk instans Aurora DB tertentu yang Anda hubungkan, bukan seluruh klaster Aurora DB. Untuk melihat apakah instans DB mendekati batas konkurens inya, bandingkan nilai ini dengan `Aurora_pq _max_concurrent_requests` .

Nama	Deskripsi
`Aurora_pq_request_not_chosen`	Berapa kali kueri paralel tidak dipilih untuk memenuhi suatu kueri. Nilai ini adalah jumlah dari beberapa penghitung granular lainnya. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-ben ar dilakukan.
`Aurora_pq_request_not_chose n_below_min_rows`	Berapa kali kueri paralel tidak dipilih karena jumlah baris dalam tabel. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan.
`Aurora_pq_request_not_chose n_column_bit`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena jenis data yang tidak didukung dalam daftar kolom yang diproyeks ikan.
`Aurora_pq_request_not_chose n_column_geometry`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki kolom dengan jenis data `GEOMETRY`. Untuk informasi tentang versi Aurora MySQL yang menghapus batasan ini, lihat Meningkatkan klaster kueri paralel ke Aurora MySQL versi 3.
`Aurora_pq_request_not_chose n_column_lob`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki kolom dengan jenis data `LOB`, atau kolom `VARCHAR` yang disimpan secara eksternal karena panjang yang dinyatakan. Untuk informasi tentang versi Aurora MySQL yang menghapus batasan ini, lihat Meningkatkan klaster kueri paralel ke Aurora MySQL versi 3.

Nama	Deskripsi
`Aurora_pq_request_not_chosen_column_virtual`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel berisi kolom virtual.
`Aurora_pq_request_not_chosen_custom_charset`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki kolom dengan set karakter kustom.
`Aurora_pq_request_not_chosen_fast_ddl`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel saat ini sedang diubah oleh pernyataan `ALTER` DDL cepat.
`Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool`	Berapa kali kueri paralel tidak dipilih, meskipun kurang dari 95 persen data tabel berada di dalam pool buffer, karena data tabel yang tidak di-buffer tidak cukup untuk membuat kueri paralel layak dijalankan.
`Aurora_pq_request_not_chosen_full_text_index`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki indeks teks penuh.
`Aurora_pq_request_not_chosen_high_buffer_pool_pct`	Berapa kali kueri paralel tidak dipilih karena persentase tinggi data tabel (saat ini, lebih dari 95 persen) sudah berada di dalam pool buffer. Dalam kasus ini, pengoptimal menentukan bahwa membaca data dari pool buffer akan lebih efisien. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan.

Nama	Deskripsi
`Aurora_pq_request_not_chosen_index_hint`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri mencakup petunjuk indeks.
`Aurora_pq_request_not_chosen_innodb_table_format`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel menggunakan format baris InnoDB yang tidak didukung. Kueri paralel Aurora hanya berlaku untuk format baris `COMPACT`, `REDUNDANT` , dan `DYNAMIC`.
`Aurora_pq_request_not_chosen_long_trx`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri dimulai di dalam transaksi yang berjalan lama. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan.
`Aurora_pq_request_not_chosen_no_where_clause`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri tidak mencakup klausa `WHERE` apa pun.
`Aurora_pq_request_not_chosen_range_scan`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri menggunakan pemindaian rentang pada indeks.
`Aurora_pq_request_not_chosen_row_length_too_long`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena total panjang gabungan semua kolom terlalu panjang.

Nama	Deskripsi
Aurora_pq_request_not_chose n_small_table	Berapa kali kueri paralel tidak dipilih karena ukuran keseluruhan tabel, sebagaimana ditentukan oleh jumlah baris dan rata-rata panjang baris. Pernyataan EXPLAIN dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan.
Aurora_pq_request_not_chose n_temporary_table	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri mengacu pada tabel sementara yang menggunakan jenis tabel MyISAM atau memory yang tidak didukung.
Aurora_pq_request_not_chose n_tx_isolation	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri menggunakan tingkat isolasi transaksi yang tidak didukung. Pada instans pembaca DB, kueri paralel hanya berlaku untuk tingkat isolasi REPEATABLE READ dan READ COMMITTED .
Aurora_pq_request_not_chose n_update_delete_stmts	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri adalah bagian dari pernyataan UPDATE atau DELETE.
Aurora_pq_request_not_chose n_unsupported_access	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena klausa WHERE tidak memenuhi kriteria untuk kueri paralel. Hasil ini dapat muncul jika kueri tidak memerlukan pemindaian sarat data, atau jika kueri adalah pernyataan DELETE atau UPDATE.

Nama	Deskripsi
`Aurora_pq_request_not_chosen_unsupported_storage_type`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena klaster Aurora MySQL DB tidak menggunakan konfigurasi penyimpanan klaster Aurora MySQL yang didukung. Parameter ini tersedia di Aurora MySQL versi 3.04 dan lebih tinggi. Untuk informasi selengkapnya, lihat Pembatasan.
`Aurora_pq_request_throttled`	Berapa kali kueri paralel tidak dipilih karena jumlah maksimum kueri paralel konkuren sudah berjalan pada instans Aurora DB tertentu.

Cara kerja kueri paralel dengan konsep SQL

Pada bagian berikut, Anda dapat menemukan detail lebih lanjut tentang alasan pernyataan SQL tertentu menggunakan atau tidak menggunakan kueri paralel. Bagian ini juga memperinci cara fitur Aurora MySQL berinteraksi dengan kueri paralel. Informasi ini dapat membantu Anda mendiagnosis masalah kinerja untuk klaster yang menggunakan kueri paralel atau memahami cara kueri paralel berlaku untuk beban kerja tertentu Anda.

Keputusan untuk menggunakan kueri paralel bergantung pada banyak faktor yang terjadi pada saat pernyataan itu berjalan. Dengan demikian, kueri paralel dapat digunakan untuk kueri tertentu selalu, tidak pernah, atau hanya dalam kondisi tertentu.

> **Tip**
>
> Saat melihat contoh ini di HTML, Anda dapat menggunakan widget Salin di sudut kanan atas setiap daftar kode guna menyalin kode SQL untuk mencoba sendiri. Dengan menggunakan widget Salin, Anda tidak perlu menyalin karakter tambahan di sekitar prompt `mysql>` dan baris lanjutan `->`.

Topik

- Pernyataan EXPLAIN

- Klausa WHERE
- Bahasa definisi data (DDL)
- Jenis data kolom
- Tabel yang dipartisi
- Fungsi agregat, klausa GROUP BY, dan klausa HAVING
- Panggilan fungsi dalam klausa WHERE
- Klausa LIMIT
- Operator perbandingan
- Gabungan
- Subkueri
- UNION
- Tampilan
- Pernyataan bahasa manipulasi data (DML)
- Transaksi dan penguncian
- Indeks pohon B
- Indeks pencarian teks penuh (FTS)
- Kolom virtual
- Mekanisme caching bawaan
- Petunjuk pengoptimal
- Tabel sementara MyISAM

Pernyataan EXPLAIN

Seperti yang ditunjukkan dalam contoh di seluruh bagian ini, pernyataan `EXPLAIN` menunjukkan apakah setiap tahap kueri saat ini memenuhi syarat untuk kueri paralel. Pernyataan itu juga menunjukkan aspek mana dari kueri yang dapat diturunkan ke lapisan penyimpanan. Berikut ini adalah item terpenting dalam rencana kueri:

- Nilai selain `NULL` untuk kolom `key` menunjukkan bahwa kueri dapat dilakukan secara efisien menggunakan pencarian indeks, dan kueri paralel tidak memungkinkan.
- Nilai kecil untuk kolom `rows` (nilai bukan dalam jutaan) menunjukkan bahwa kueri tersebut tidak mengakses cukup data untuk membuat kueri paralel yang bermanfaat. Ini berarti kueri paralel tidak memungkinkan.

- Kolom `Extra` menunjukkan jika kueri paralel yang diharapkan akan digunakan. Output ini terlihat seperti contoh berikut.

  ```
  Using parallel query (A columns, B filters, C exprs; D extra)
  ```

  Angka `columns` menunjukkan jumlah kolom yang dimaksud dalam blok kueri.

  Angka `filters` menunjukkan jumlah predikat `WHERE` yang menunjukkan perbandingan sederhana antara nilai kolom dan konstanta. Perbandingannya dapat berupa kesetaraan, ketidaksetaraan, atau rentang. Aurora dapat memaralelkan jenis predikat ini dengan sangat efektif.

  Angka `exprs` menunjukkan jumlah ekspresi seperti panggilan fungsi, operator, atau ekspresi lainnya yang juga dapat diparalelkan, meskipun tidak sama efektifnya dengan ketentuan filter.

  Angka `extra` menunjukkan jumlah ekspresi yang tidak dapat diturunkan dan dilakukan oleh simpul kepala.

Contohnya, pertimbangkan output `EXPLAIN` berikut ini.

```
mysql> explain select p_name, p_mfgr from part
    -> where p_brand is not null
    -> and upper(p_type) is not null
    -> and round(p_retailprice) is not null;
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type | table |...| rows     | Extra
                                |
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
|  1 | SIMPLE      | part  |...| 20427936 | Using where; Using parallel query (5
 columns, 1 filters, 2 exprs; 0 extra) |
+----+-------------+-------+...+----------
+-------------------------------------------------------------------------------+
```

Informasi dari kolom `Extra` menunjukkan bahwa lima kolom diekstraksi dari setiap baris untuk mengevaluasi ketentuan kueri dan menyusun set hasil. Satu predikat `WHERE` mencakup sebuah filter, yaitu, kolom yang langsung diuji dalam klausa `WHERE`. Dua klausa `WHERE` memerlukan evaluasi ekspresi yang lebih rumit, dalam hal ini mencakup panggilan fungsi. Bidang `0 extra` mengonfirmasi bahwa semua operasi dalam klausa `WHERE` diturunkan ke lapisan penyimpanan sebagai bagian dari pemrosesan kueri paralel.

Dalam kasus di mana kueri paralel tidak dipilih, Anda biasanya dapat menyimpulkan alasan dari output kolom EXPLAIN lain. Contohnya, nilai rows mungkin terlalu kecil, atau kolom possible_keys mungkin menunjukkan bahwa kueri dapat menggunakan pencarian indeks alih-alih pemindaian sarat data. Contoh berikut menunjukkan sebuah kueri di mana pengoptimal dapat memperkirakan bahwa kueri tersebut hanya akan memindai sejumlah kecil baris. Hal itu dilakukan berdasarkan karakteristik kunci primer. Dalam hal ini, kueri paralel tidak diperlukan.

```
mysql> explain select count(*) from part where p_partkey between 1 and 100;
+----+-------------+-------+------+---------------+---------+---------+------+------
+--------------------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref  | rows |
 Extra                    |
+----+-------------+-------+------+---------------+---------+---------+------+------
+--------------------------+
|  1 | SIMPLE      | part  | range | PRIMARY       | PRIMARY | 4       | NULL |   99 |
 Using where; Using index |
+----+-------------+-------+------+---------------+---------+---------+------+------
+--------------------------+
```

Output yang menunjukkan apakah kueri paralel akan digunakan mempertimbangkan semua faktor yang tersedia saat pernyataan EXPLAIN dijalankan. Pengoptimal mungkin mengambil pilihan lain saat kueri benar-benar dijalankan, jika situasi berubah pada saat itu. Contohnya, EXPLAIN dapat melaporkan bahwa suatu pernyataan akan menggunakan kueri paralel. Namun saat kueri benar-benar dijalankan nantinya, kueri tersebut mungkin tidak menggunakan kueri paralel berdasarkan kondisi pada saat itu. Kondisi tersebut dapat mencakup beberapa kueri paralel lainnya yang berjalan secara serentak. Kondisi lainnya juga dapat mencakup baris yang dihapus dari tabel, indeks baru yang dibuat, terlalu banyak waktu yang terlewati dalam transaksi terbuka, dan sebagainya.

Klausa WHERE

Agar menggunakan pengoptimalan kueri paralel, suatu kueri harus mencakup klausa WHERE.

Pengoptimalan kueri paralel mempercepat berbagai jenis ekspresi yang digunakan dalam klausa WHERE:

- Perbandingan sederhana antara nilai kolom dengan konstanta, yang disebut filter. Perbandingan ini sangat diuntungkan ketika diturunkan ke lapisan penyimpanan. Jumlah ekspresi filter dalam sebuah kueri dilaporkan dalam output EXPLAIN.

- Jenis ekspresi lain dalam klausa `WHERE` juga diturunkan ke lapisan penyimpanan jika memungkinkan. Jumlah ekspresi tersebut dalam sebuah kueri dilaporkan dalam output `EXPLAIN`. Ekspresi ini dapat berupa panggilan fungsi, operator `LIKE`, ekspresi `CASE`, dan sebagainya.
- Fungsi dan operator tertentu saat ini tidak diturunkan oleh kueri paralel. Jumlah ekspresi tersebut dalam sebuah kueri dilaporkan sebagai penghitung `extra` dalam output `EXPLAIN`. Sisa kueri tersebut masih dapat menggunakan kueri paralel.
- Meskipun ekspresi dalam daftar pilihan tidak diturunkan, kueri yang berisi fungsi tersebut masih dapat diuntungkan dari pengurangan lalu lintas jaringan untuk hasil perantara dari kueri paralel. Contohnya, kueri yang memanggil fungsi agregat dalam daftar pilihan dapat diuntungkan dari kueri paralel, meskipun fungsi agregasi tidak diturunkan.

Contohnya, kueri berikut ini melakukan pemindaian tabel lengkap dan memproses semua nilai untuk kolom `P_BRAND`. Akan tetapi, kueri tersebut tidak menggunakan kueri paralel karena tidak mencakup klausa `WHERE` apa pun.

```
mysql> explain select count(*), p_brand from part group by p_brand;
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows     |
 Extra                           |
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
|  1 | SIMPLE      | part  | ALL  | NULL          | NULL | NULL    | NULL | 20427936 |
 Using temporary; Using filesort |
+----+-------------+-------+------+---------------+------+---------+------+----------
+---------------------------------+
```

Sebaliknya, kueri berikut mencakup predikat `WHERE` yang memfilter hasil, sehingga kueri paralel dapat diterapkan:

```
mysql> explain select count(*), p_brand from part where p_name is not null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
    -> group by p_brand;
+----+...+----------
+-------------------------------------------------------------------------------------------
+
| id |...| rows     | Extra
                                          |
```

```
+----+...+----------
+-------------------------------------------------------------------------------------------------
+
|  1 |...| 20427936 | Using where; Using temporary; Using filesort; Using parallel
 query (5 columns, 1 filters, 2 exprs; 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------------------------
+
```

Jika pengoptimal memperkirakan bahwa baris yang dikembalikan untuk suatu blok kueri jumlahnya kecil, kueri paralel tidak digunakan untuk blok kueri tersebut. Contoh berikut menunjukkan kasus di mana operator lebih-besar-daripada di kolom kunci primer berlaku untuk jutaan baris, yang menyebabkan kueri paralel digunakan. Pengujian kurang-dari yang berkebalikan diperkirakan berlaku hanya pada beberapa baris dan tidak menggunakan kueri paralel.

```
mysql> explain select count(*) from part where p_partkey > 10;
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+

mysql> explain select count(*) from part where p_partkey < 10;
+----+...+------+--------------------------+
| id |...| rows | Extra                    |
+----+...+------+--------------------------+
|  1 |...|    9 | Using where; Using index |
+----+...+------+--------------------------+
```

Bahasa definisi data (DDL)

Di Aurora MySQL versi 2, kueri paralel hanya tersedia untuk tabel yang tidak memiliki operasi bahasa definisi data (DDL) cepat yang tertunda. Di Aurora MySQL versi 3, Anda dapat menggunakan kueri paralel untuk suatu tabel pada saat yang sama dengan operasi DDL instan.

DDL instan di Aurora MySQL versi 3 menggantikan fitur DDL cepat di Aurora MySQL versi 2. Untuk informasi tentang DDL, lihat DDL instan (Aurora MySQL versi 3).

Jenis data kolom

Di Aurora MySQL versi 3, kueri paralel dapat bekerja dengan tabel yang berisi kolom dengan tipe data `TEXT`, `BLOB`, `JSON`, dan `GEOMETRY`. Ini juga dapat bekerja dengan kolom `VARCHAR` dan `CHAR` dengan panjang maksimum yang dinyatakan melebihi 768 byte. Jika kueri Anda mengacu pada kolom yang berisi jenis objek besar seperti itu, tugas tambahan untuk mengambilnya akan memunculkan overhead ke dalam pemrosesan kueri. Dalam hal ini, periksa apakah kueri dapat menghilangkan referensi ke kolom-kolom tersebut. Jika tidak, jalankan tolok ukur untuk mengonfirmasi apakah kueri tersebut lebih cepat dengan kueri paralel diaktifkan atau dinonaktifkan.

Di Aurora MySQL versi 2, kueri paralel memiliki batasan-batasan tersebut untuk jenis objek besar:

- Jenis data `TEXT`, `BLOB`, `JSON`, dan `GEOMETRY` tidak didukung dengan kueri paralel. Kueri yang mengacu pada kolom apa pun dari jenis ini tidak dapat menggunakan kueri paralel.
- Kolom dengan panjang variabel (jenis data `VARCHAR` dan `CHAR`) kompatibel dengan kueri paralel hingga panjang maksimum yang dinyatakan 768 byte. Kueri yang mengacu pada kolom apa pun dari jenis yang dinyatakan dengan panjang maksimum yang lebih panjang tidak dapat menggunakan kueri paralel. Untuk kolom yang menggunakan set karakter multibyte, batas byte mempertimbangkan jumlah maksimum byte dalam set karakter. Contohnya, untuk set karakter `utf8mb4` (yang memiliki panjang karakter maksimum 4 byte), kolom `VARCHAR(192)` kompatibel dengan kueri paralel tetapi kolom `VARCHAR(193)` tidak.

Tabel yang dipartisi

Anda dapat menggunakan tabel yang dipartisi dengan kueri paralel di Aurora MySQL versi 3. Karena tabel yang dipartisi direpresentasikan secara internal sebagai beberapa tabel yang lebih kecil, kueri yang menggunakan kueri paralel pada tabel yang tidak dipartisi mungkin tidak menggunakan kueri paralel pada tabel yang dipartisi identik. Aurora MySQL mempertimbangkan apakah setiap partisi cukup besar untuk memenuhi syarat optimasi kueri paralel, bukan mengevaluasi ukuran seluruh tabel. Periksa apakah variabel status `Aurora_pq_request_not_chosen_small_table` bertambah jika kueri pada tabel yang dipartisi tidak menggunakan kueri paralel padahal Anda mengharapkannya.

Misalnya, pertimbangkan satu tabel yang dipartisi dengan `PARTITION BY HASH (`*`column`*`) PARTITIONS 2` dan tabel lain dipartisi dengan `PARTITION BY HASH (`*`column`*`) PARTITIONS 10`. Dalam tabel dengan dua partisi, partisinya lima kali lebih besar daripada tabel dengan sepuluh partisi. Dengan demikian, kueri paralel lebih memungkinkan untuk digunakan untuk mengkueri tabel dengan lebih sedikit partisi. Dalam contoh berikut, tabel `PART_BIG_PARTITIONS` memiliki dua partisi

dan PART_SMALL_PARTITIONS memiliki sepuluh partisi. Dengan data yang identik, kueri paralel lebih memungkinkan untuk digunakan untuk tabel dengan lebih sedikit partisi besar.

```
mysql> explain select count(*), p_brand from part_big_partitions where p_name is not
 null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+---------------------+------------
+-------------------------------------------------------------------------------------------
+
| id | select_type | table               | partitions | Extra
                                                                                           |
+----+-------------+---------------------+------------
+-------------------------------------------------------------------------------------------
+
|  1 | SIMPLE      | part_big_partitions | p0,p1      | Using where; Using temporary;
 Using parallel query (4 columns, 1 filters, 1 exprs; 0 extra; 1 group-bys, 1 aggrs) |
+----+-------------+---------------------+------------
+-------------------------------------------------------------------------------------------
+

mysql> explain select count(*), p_brand from part_small_partitions where p_name is not
 null
    ->   and p_mfgr in ('Manufacturer#1', 'Manufacturer#3') and p_retailprice > 1000
 group by p_brand;
+----+-------------+-----------------------+------------------------------
+------------------------------+
| id | select_type | table                 | partitions                   | Extra
                   |
+----+-------------+-----------------------+------------------------------
+------------------------------+
|  1 | SIMPLE      | part_small_partitions | p0,p1,p2,p3,p4,p5,p6,p7,p8,p9 | Using
 where; Using temporary |
+----+-------------+-----------------------+------------------------------
+------------------------------+
```

Fungsi agregat, klausa GROUP BY, dan klausa HAVING

Kueri yang mencakup fungsi agregat sering kali merupakan kandidat yang baik untuk kueri paralel, karena kueri tersebut memindai baris dalam jumlah besar dalam tabel besar.

Di Aurora MySQL 3, kueri paralel dapat mengoptimalkan panggilan fungsi agregat dalam daftar pilih dan klausa `HAVING`.

Sebelum Aurora MySQL 3, panggilan fungsi agregat dalam daftar pilih atau klausa `HAVING` tidak ditekan ke lapisan penyimpanan. Akan tetapi, kueri paralel tetap dapat meningkatkan kinerja kueri dengan fungsi agregat tersebut. Hal itu dilakukan pertama-tama dengan mengekstraksi nilai kolom dari halaman data mentah secara paralel pada lapisan penyimpanan. Kueri paralel tersebut kemudian mengirim kembali nilai itu ke simpul kepala dalam format urutan yang padat, bukan sebagai keseluruhan halaman data. Seperti biasa, kueri memerlukan setidaknya satu predikat `WHERE` agar kueri paralel dapat diaktifkan.

Contoh sederhana berikut mengilustrasikan jenis kueri agregat yang dapat diuntungkan dari kueri paralel. Hal tersebut dilakukan dengan mengembalikan hasil menengah dalam bentuk ringkas ke simpul kepala, memfilter baris yang tidak cocok dari hasil menengah, atau keduanya.

```
mysql> explain select sql_no_cache count(distinct p_brand) from part where p_mfgr =
 'Manufacturer#5';
+----+...+-------------------------------------------------------------------------+
| id |...| Extra                                                                   |
+----+...+-------------------------------------------------------------------------+
|  1 |...| Using where; Using parallel query (2 columns, 1 filters, 0 exprs; 0 extra) |
+----+...+-------------------------------------------------------------------------+

mysql> explain select sql_no_cache p_mfgr from part where p_retailprice > 1000 group by
 p_mfgr having count(*) > 100;
+----+...
+-----------------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                  |
+----+...
+-----------------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (3
 columns, 0 filters, 1 exprs; 0 extra) |
+----+...
+-----------------------------------------------------------------------------------------------------------
+
```

Panggilan fungsi dalam klausa WHERE

Aurora dapat menerapkan pengoptimalan kueri paralel pada panggilan ke sebagian besar fungsi bawaan dalam klausa `WHERE`. Paralelisasi panggilan fungsi ini melimpahkan beberapa pekerjaan CPU dari simpul kepala. Mengevaluasi fungsi predikat secara paralel selama tahap kueri paling awal akan membantu Aurora meminimalkan jumlah data yang dikirim dan diproses selama tahap berikutnya.

Saat ini, paralelisasi tidak berlaku untuk panggilan fungsi dalam daftar pilihan. Fungsi-fungsi tersebut dievaluasi oleh simpul kepala, bahkan jika panggilan fungsi yang identik muncul dalam klausa `WHERE`. Nilai asli dari kolom yang relevan disertakan dalam urutan yang dikirim dari simpul penyimpanan kembali ke simpul kepala. Simpul kepala melakukan transformasi apa pun seperti `UPPER`, `CONCATENATE`, dan seterusnya untuk menghasilkan nilai akhir untuk set hasil.

Dalam contoh berikut, kueri paralel memaralelkan panggilan dengan `LOWER` karena muncul di klausa `WHERE`. Kueri paralel tidak memengaruhi panggilan ke `SUBSTR` dan `UPPER` karena muncul dalam daftar pilihan.

```
mysql> explain select sql_no_cache distinct substr(upper(p_name),1,5) from part
    -> where lower(p_name) like '%cornflower%' or lower(p_name) like '%goldenrod%';
+----+...
+-------------------------------------------------------------------------------------------
+
| id |...| Extra
                |
+----+...
+-------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using parallel query (2 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------
+
```

Pertimbangan yang sama berlaku untuk ekspresi lain, seperti ekspresi `CASE` atau operator `LIKE`. Misalnya, contoh berikut menunjukkan bahwa kueri paralel mengevaluasi ekspresi `CASE` dan operator `LIKE` di klausa `WHERE`.

```
mysql> explain select p_mfgr, p_retailprice from part
    -> where p_retailprice > case p_mfgr
```

```
    ->   when 'Manufacturer#1' then 1000
    ->   when 'Manufacturer#2' then 1200
    ->   else 950
    -> end
    -> and p_name like '%vanilla%'
    -> group by p_retailprice;
+----+...
+-------------------------------------------------------------------------------------------------
+
| id |...| Extra
                                |
+----+...
+-------------------------------------------------------------------------------------------------
+
|  1 |...| Using where; Using temporary; Using filesort; Using parallel query (4
 columns, 0 filters, 2 exprs; 0 extra) |
+----+...
+-------------------------------------------------------------------------------------------------
+
```

Klausa LIMIT

Saat ini, kueri paralel tidak digunakan untuk blok kueri yang mencakup klausa `LIMIT`. Kueri paralel masih dapat digunakan untuk fase kueri sebelumnya dengan `GROUP by`, `ORDER BY`, atau gabungan.

Operator perbandingan

Pengoptimal memperkirakan jumlah baris yang harus dipindai untuk mengevaluasi operator perbandingan, dan menentukan apakah akan menggunakan kueri paralel berdasarkan estimasi tersebut.

Contoh pertama berikut ini menunjukkan bahwa perbandingan setara terhadap kolom kunci primer dapat dilakukan secara efisien tanpa kueri paralel. Contoh kedua berikut menunjukkan bahwa perbandingan yang serupa terhadap kolom yang tidak diindeks memerlukan pemindaian jutaan baris, sehingga dapat diuntungkan dari kueri paralel.

```
mysql> explain select * from part where p_partkey = 10;
+----+...+------+-------+
| id |...| rows | Extra |
+----+...+------+-------+
|  1 |...|    1 | NULL  |
+----+...+------+-------+
```

```
mysql> explain select * from part where p_type = 'LARGE BRUSHED BRASS';
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

Pertimbangan yang sama berlaku untuk pengujian tidak sama dan untuk perbandingan rentang seperti kurang dari, lebih besar dari, atau sama dengan, atau `BETWEEN`. Pengoptimal memperkirakan jumlah baris yang akan dipindai, dan menentukan apakah kueri paralel bermanfaat berdasarkan volume keseluruhan I/O.

Gabungan

Kueri gabungan dengan tabel besar biasanya mencakup operasi sarat data yang diuntungkan dari pengoptimalan kueri paralel. Perbandingan nilai kolom di antara beberapa tabel (yaitu, predikat gabungan itu sendiri) saat ini tidak diparalelkan. Namun, kueri paralel dapat menurunkan beberapa pemrosesan internal untuk fase penggabungan lainnya, seperti membangun filter Bloom selama hash join. Kueri paralel dapat diterapkan pada kueri gabungan tanpa klausa `WHERE`. Oleh karena itu, kueri gabungan adalah pengecualian untuk aturan bahwa klausa `WHERE` diperlukan untuk menggunakan kueri paralel.

Setiap fase pemrosesan gabungan dievaluasi untuk memeriksa apakah fase tersebut memenuhi syarat untuk kueri paralel. Jika lebih dari satu fase dapat menggunakan kueri paralel, fase-fase ini dilakukan secara berurutan. Dengan demikian, setiap kueri gabungan dihitung sebagai satu sesi kueri paralel dalam hal batas keserentakan.

Contohnya, saat kueri gabungan mencakup predikat `WHERE` untuk memfilter baris dari salah satu tabel yang digabungkan, opsi pemfilteran tersebut dapat menggunakan kueri paralel. Sebagai contoh lain, anggaplah kueri gabungan menggunakan mekanisme hash join, contohnya untuk menggabungkan tabel besar dengan tabel kecil. Dalam hal ini, pemindaian tabel untuk menghasilkan struktur data filter Bloom mungkin dapat menggunakan kueri paralel.

> **Note**
>
> Kueri paralel biasanya digunakan untuk jenis kueri sarat sumber daya yang diuntungkan dari pengoptimalan hash join. Metode untuk mengaktifkan optimasi hash join ditentukan oleh versi Aurora MySQL. Untuk detail untuk setiap versi, lihat Mengaktifkan hash join untuk klaster kueri paralel. Untuk informasi tentang cara menggunakan hash join secara efektif, lihat Mengoptimalkan kueri join MySQL Aurora besar dengan hash join.

```
mysql> explain select count(*) from orders join customer where o_custkey = c_custkey;
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------
+
| id |...| table    | type  | possible_keys | key         |...| rows      | Extra

              |
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------
+
|  1 |...| customer | index | PRIMARY       | c_nationkey |...|  15051972 | Using index

              |
|  1 |...| orders   | ALL   | o_custkey     | NULL        |...| 154545408 | Using join
 buffer (Hash Join Outer table orders); Using parallel query (1 columns, 0 filters, 1
 exprs; 0 extra) |
+----+...+----------+-------+--------------+-------------+...+-----------
+-------------------------------------------------------------------------------------------
+
```

Untuk kueri gabungan yang menggunakan mekanisme nested loop, blok nested loop terluar mungkin menggunakan kueri paralel. Penggunaan kueri paralel bergantung pada faktor yang sama seperti biasanya, seperti adanya ketentuan filter tambahan dalam klausa `WHERE`.

```
mysql> -- Nested loop join with extra filter conditions can use parallel query.
mysql> explain select count(*) from part, partsupp where p_partkey != ps_partkey and
 p_name is not null and ps_availqty > 0;
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------+
| id | select_type | table    |...| rows     | Extra
                                  |
```

```
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
|  1 | SIMPLE      | part     |...| 20427936 | Using where; Using parallel query (2
 columns, 1 filters, 0 exprs; 0 extra) |
|  1 | SIMPLE      | partsupp |...| 78164450 | Using where; Using join buffer (Block
 Nested Loop)                          |
+----+-------------+----------+...+----------
+-------------------------------------------------------------------------------+
```

Subkueri

Blok kueri luar dan blok subkueri dalam masing-masing dapat menggunakan kueri paralel, atau tidak. Untuk setiap blok, menggunakan kueri paralel atau tidak, didasarkan pada karakteristik umum tabel, klausa `WHERE`, dan sebagainya. Contohnya, kueri berikut menggunakan kueri paralel untuk blok subkueri tetapi tidak untuk blok luar.

```
mysql> explain select count(*) from part where
   --> p_partkey < (select max(p_partkey) from part where p_name like '%vanilla%');
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
| id | select_type |...| rows     | Extra
                       |
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
|  1 | PRIMARY     |...|     NULL | Impossible WHERE noticed after reading const tables
                      |
|  2 | SUBQUERY    |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
+----+-------------+...+----------
+-------------------------------------------------------------------------------+
```

Saat ini, subkueri yang berkorelasi tidak dapat menggunakan pengoptimalan kueri paralel.

UNION

Setiap blok kueri dalam kueri `UNION` dapat menggunakan kueri paralel atau tidak, berdasarkan karakteristik umum tabel, klausa `WHERE`, dan sebagainya, untuk setiap bagian `UNION`.

```
mysql> explain select p_partkey from part where p_name like '%choco_ate%'
    -> union select p_partkey from part where p_name like '%vanil_a%';
+----+---------------+...+----------
+-------------------------------------------------------------------------------+
```

```
| id | select_type    |...| rows     | Extra
                          |
+----+----------------+...+----------
+-------------------------------------------------------------------------------+
|  1 | PRIMARY        |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
|  2 | UNION          |...| 20427936 | Using where; Using parallel query (2 columns, 0
 filters, 1 exprs; 0 extra) |
| NULL | UNION RESULT | <union1,2> |...|     NULL | Using temporary
                       |
+----+--------------+...+----------
+-------------------------------------------------------------------------------+
```

> **Note**
>
> Setiap klausa `UNION` dalam kueri dijalankan secara berurutan. Bahkan jika kueri mencakup beberapa tahap yang semuanya menggunakan kueri paralel, kueri tersebut hanya menjalankan satu kueri paralel dalam satu waktu. Oleh karena itu, bahkan kueri multitahap yang kompleks hanya dihitung sebagai 1 terhadap batas kueri paralel yang serentak.

Tampilan

Pengoptimal menulis ulang kueri apa pun menggunakan tampilan sebagai kueri yang lebih panjang menggunakan tabel yang mendasarinya. Dengan demikian, kueri paralel bekerja dengan cara yang sama baik referensi tabelnya berupa tampilan atau tabel nyata. Semua pertimbangan yang sama tentang apakah akan menggunakan kueri paralel untuk suatu kueri, dan bagian mana yang diturunkan, berlaku untuk kueri terakhir yang ditulis ulang.

Contohnya, rencana kueri berikut menunjukkan definisi tampilan yang biasanya tidak menggunakan kueri paralel. Saat tampilan dikueri dengan klausa `WHERE` tambahan, Aurora MySQL menggunakan kueri paralel.

```
mysql> create view part_view as select * from part;
mysql> explain select count(*) from part_view where p_partkey is not null;
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
```

```
|  1 |...| 20427936 | Using where; Using parallel query (1 columns, 0 filters, 0 exprs;
 1 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

Pernyataan bahasa manipulasi data (DML)

Pernyataan `INSERT` dapat menggunakan kueri paralel untuk fase pemrosesan `SELECT`, jika bagian `SELECT` sesuai dengan ketentuan lainnya untuk kueri paralel.

```
mysql> create table part_subset like part;
mysql> explain insert into part_subset select * from part where p_mfgr =
 'Manufacturer#1';
+----+...+----------
+-------------------------------------------------------------------------------+
| id |...| rows     | Extra
          |
+----+...+----------
+-------------------------------------------------------------------------------+
|  1 |...| 20427936 | Using where; Using parallel query (9 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+----------
+-------------------------------------------------------------------------------+
```

> **Note**
>
> Biasanya, setelah pernyataan `INSERT`, data untuk baris yang baru dimasukkan ada di dalam buffer pool. Oleh karena itu, suatu tabel mungkin tidak memenuhi syarat untuk kueri paralel segera setelah memasukkan baris dalam jumlah besar. Lalu, setelah data dikosongkan dari buffer pool selama operasi normal, kueri terhadap tabel tersebut dapat mulai menggunakan kueri paralel.

Pernyataan `CREATE TABLE AS SELECT` tidak menggunakan kueri paralel, meskipun bagian `SELECT` dari pernyataan tersebut akan memenuhi syarat untuk kueri paralel. Aspek DDL dari pernyataan ini menjadikannya tidak kompatibel dengan pemrosesan kueri paralel. Sebaliknya, dalam pernyataan `INSERT ... SELECT`, bagian `SELECT` dapat menggunakan kueri paralel.

Kueri paralel tidak pernah digunakan untuk pernyataan `DELETE` atau `UPDATE`, terlepas dari ukuran tabel dan predikat dalam klausa `WHERE`.

```
mysql> explain delete from part where p_name is not null;
+----+-------------+...+----------+-------------+
| id | select_type |...| rows     | Extra       |
+----+-------------+...+----------+-------------+
|  1 | SIMPLE      |...| 20427936 | Using where |
+----+-------------+...+----------+-------------+
```

Transaksi dan penguncian

Anda dapat menggunakan semua tingkat isolasi pada instans utama Aurora.

Pada instans pembaca Aurora DB, kueri paralel berlaku untuk pernyataan yang dilakukan dalam tingkat isolasi `REPEATABLE READ`. Aurora MySQL versi 2.09 atau lebih tinggi juga dapat menggunakan tingkat isolasi `READ COMMITTED` pada instans DB pembaca. `REPEATABLE READ` adalah tingkat isolasi default untuk instans DB pembaca Aurora. Untuk menggunakan tingkat isolasi `READ COMMITTED` pada instans reader DB membutuhkan pengaturan opsi konfigurasi `aurora_read_replica_read_committed` pada tingkat sesi. Tingkat isolasi `READ COMMITTED` untuk instans pembaca sesuai dengan perilaku standar SQL. Namun, isolasi ini lebih longgar pada instans pembaca dibandingkan ketika kueri menggunakan tingkat isolasi `READ COMMITTED` pada instans penulis.

Untuk informasi lebih lanjut tentang tingkat isolasi Aurora, terutama perbedaan dalam hal `READ COMMITTED` antara instans penulis dan pembaca, lihat Tingkat isolasi Aurora MySQL.

Setelah transaksi besar selesai, statistik tabel mungkin menjadi usang. Statistik yang usang tersebut mungkin memerlukan pernyataan `ANALYZE TABLE` sebelum Aurora dapat secara akurat memperkirakan jumlah baris. Pernyataan skala besar DML juga dapat memasukkan sebagian besar data tabel ke buffer pool. Memiliki data ini di dalam buffer pool dapat menyebabkan kueri paralel menjadi lebih jarang dipilih untuk tabel tersebut sampai data dikosongkan dari pool.

Jika sesi Anda berada dalam transaksi yang berlangsung lama (secara default, 10 menit), kueri lebih lanjut di dalam sesi tersebut tidak menggunakan kueri paralel. Waktu habis juga dapat terjadi dalam satu kueri yang berlangsung lama. Jenis waktu habis ini dapat terjadi jika kueri berjalan lebih lama dari interval maksimum (saat ini 10 menit) sebelum pemrosesan kueri paralel dimulai.

Anda dapat mengurangi kemungkinan memulai transaksi yang berlangsung lama secara tidak sengaja dengan mengatur `autocommit=1` dalam sesi `mysql` di mana Anda melakukan kueri ad hoc (satu waktu). Bahkan pernyataan `SELECT` terhadap suatu tabel akan memulai transaksi dengan membuat tampilan baca. Tampilan baca merupakan rangkaian data yang konsisten untuk kueri berikutnya yang tetap ada hingga transaksi dilakukan. Waspadai pembatasan ini juga saat

menggunakan aplikasi JDBC atau ODBC dengan Aurora, karena aplikasi tersebut mungkin berjalan dengan pengaturan `autocommit` yang tidak aktif.

Contoh berikut ini menunjukkan bagaimana, dengan pengaturan `autocommit` yang dinonaktifkan, menjalankan suatu kueri terhadap tabel akan menciptakan tampilan baca yang secara implisit memulai transaksi. Kueri yang dijalankan segera sesudahnya masih dapat menggunakan kueri paralel. Namun, setelah jeda beberapa menit, kueri tidak lagi memenuhi syarat untuk kueri paralel. Mengakhiri transaksi dengan `COMMIT` atau `ROLLBACK` akan memulihkan eligilibilitas kueri paralel.

```
mysql> set autocommit=0;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+-------------------------------------------------------------------------------+
| id |...| rows    | Extra
         |
+----+...+---------
+-------------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+-------------------------------------------------------------------------------+

mysql> select sleep(720); explain select sql_no_cache count(*) from part where
 p_retailprice > 10.0;
+------------+
| sleep(720) |
+------------+
|          0 |
+------------+
1 row in set (12 min 0.00 sec)

+----+...+---------+-------------+
| id |...| rows    | Extra       |
+----+...+---------+-------------+
|  1 |...| 2976129 | Using where |
+----+...+---------+-------------+

mysql> commit;

mysql> explain select sql_no_cache count(*) from part where p_retailprice > 10.0;
+----+...+---------
+-------------------------------------------------------------------------------+
```

```
| id |...| rows    | Extra
         |
+----+...+---------
+-----------------------------------------------------------------------------+
|  1 |...| 2976129 | Using where; Using parallel query (1 columns, 1 filters, 0 exprs;
 0 extra) |
+----+...+---------
+-----------------------------------------------------------------------------+
```

Untuk melihat berapa kali kueri tidak memenuhi syarat untuk kueri paralel karena kueri tersebut berada dalam transaksi yang berlangsung lama, periksa variabel status `Aurora_pq_request_not_chosen_long_trx`.

```
mysql> show global status like '%pq%trx%';
+--------------------------------------+-------+
| Variable_name                        | Value |
+--------------------------------------+-------+
| Aurora_pq_request_not_chosen_long_trx | 4     |
+--------------------------------+-------+
```

Setiap pernyataan `SELECT` yang memperoleh kunci, seperti sintaks `SELECT FOR UPDATE` atau `SELECT LOCK IN SHARE MODE`, tidak dapat menggunakan kueri paralel.

Kueri paralel dapat berfungsi untuk tabel yang dikunci oleh pernyataan `LOCK TABLES`.

```
mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055';
+----+...+-----------
+-----------------------------------------------------------------------------+
| id |...| rows      | Extra
          |
+----+...+-----------
+-----------------------------------------------------------------------------+
|  1 |...| 154545408 | Using where; Using parallel query (3 columns, 1 filters, 0
 exprs; 0 extra) |
+----+...+-----------
+-----------------------------------------------------------------------------+

mysql> explain select o_orderpriority, o_shippriority from orders where o_clerk =
 'Clerk#000095055' for update;
+----+...+-----------+-------------+
| id |...| rows      | Extra       |
```

```
+----+...+-----------+-------------+
|  1 |...| 154545408 | Using where |
+----+...+-----------+-------------+
```

Indeks pohon B

Statistik yang dikumpulkan oleh pernyataan `ANALYZE TABLE` membantu pengoptimal untuk memutuskan waktu untuk menggunakan kueri paralel atau pencarian indeks, berdasarkan karakteristik data untuk setiap kolom. Tetap perbarui statistik dengan menjalankan `ANALYZE TABLE` setelah operasi DML yang menerapkan perubahan substansial pada data dalam tabel.

Jika pencarian indeks dapat melakukan kueri secara efisien tanpa pemindaian sarat data, Aurora mungkin menggunakan pencarian indeks. Melakukan hal ini akan menghindari pengeluaran tambahan dari pemrosesan kueri paralel. Terdapat juga batas keserentakan pada jumlah kueri paralel yang dapat berjalan secara bersamaan di klaster Aurora DB mana pun. Pastikan Anda menggunakan praktik terbaik untuk mengindeks tabel, sehingga kueri yang paling sering dan paling sering muncul bersamaan menggunakan pencarian indeks.

Indeks pencarian teks penuh (FTS)

Saat ini, kueri paralel tidak digunakan untuk tabel yang berisi indeks pencarian teks penuh, terlepas dari apakah kueri tersebut mengacu pada kolom yang diindeks tersebut atau menggunakan operator `MATCH`.

Kolom virtual

Saat ini, kueri paralel tidak digunakan untuk tabel yang berisi kolom virtual, terlepas dari apakah kueri tersebut merujuk pada kolom virtual mana pun.

Mekanisme caching bawaan

Aurora mencakup mekanisme caching bawaan, yaitu buffer pool dan cache kueri. Pengoptimal Aurora memilih antara mekanisme caching ini dan kueri paralel bergantung pada mana yang paling efektif untuk kueri tertentu.

Saat kueri paralel memfilter baris dan mengubah serta mengekstraksi nilai kolom, data dikirim kembali ke simpul kepala sebagai urutan dan bukan sebagai halaman data. Oleh karena itu, menjalankan kueri paralel tidak akan menambahkan halaman apa pun ke dalam buffer pool, atau mengosongkan halaman yang sudah ada di dalam buffer pool.

Aurora memeriksa jumlah halaman data tabel yang ada dalam buffer pool, dan bagian mana dari data tabel tersebut yang direpresentasikan oleh angka tersebut. Aurora menggunakan informasi tersebut untuk menentukan apakah lebih efisien untuk menggunakan kueri paralel (dan mem-bypass data dalam buffer pool). Sebagai alternatif, Aurora mungkin menggunakan jalur pemrosesan kueri nonparalel, yang menggunakan cache data dalam buffer pool. Halaman mana yang dibuat cache dan bagaimana kueri sarat data dapat memengaruhi caching dan pengosongan bergantung pada pengaturan konfigurasi yang terkait dengan buffer pool. Oleh karena itu, akan sulit untuk memprediksi apakah kueri tertentu menggunakan kueri paralel, karena pilihannya bergantung pada data yang selalu berubah dalam buffer pool.

Selain itu, Aurora menerapkan batas keserentakan pada kueri paralel. Karena tidak setiap kueri menggunakan kueri paralel, tabel yang diakses oleh beberapa kueri secara serentak biasanya memiliki bagian yang substansial dari data mereka dalam buffer pool. Oleh karena itu, Aurora sering kali tidak memilih tabel ini untuk kueri paralel.

Ketika Anda menjalankan urutan kueri nonparalel pada tabel yang sama, kueri pertama mungkin lambat karena data tidak ada dalam buffer pool. Lalu kueri kedua dan selanjutnya jauh lebih cepat karena buffer pool sekarang sudah "dipanaskan". Kueri paralel biasanya menunjukkan kinerja yang konsisten dari kueri pertama terhadap tabel. Saat melakukan uji kinerja, buat tolok ukur kueri nonparalel dengan buffer pool yang dingin dan hangat. Dalam beberapa kasus, hasil dari buffer pool hangat dapat menghasilkan perbandingan yang baik dengan waktu kueri paralel. Dalam kasus ini, pertimbangkan faktor seperti frekuensi kueri terhadap tabel tersebut. Pertimbangkan juga apakah bermanfaat untuk menyimpan data untuk tabel tersebut di dalam buffer pool.

Cache kueri menghindari menjalankan ulang kueri saat kueri yang identik dikirimkan dan data tabel yang mendasarinya tidak berubah. Kueri yang dioptimalkan dengan fitur kueri paralel dapat masuk ke cache kueri, yang secara efektif menjadikannya instan saat dijalankan lagi.

> **Note**
>
> Saat melakukan perbandingan kinerja, cache kueri dapat menghasilkan jumlah pengaturan waktu yang rendah secara buatan. Oleh karena itu, dalam situasi serupa tolok ukur, Anda dapat menggunakan petunjuk `sql_no_cache`. Petunjuk ini mencegah hasil tersebut tersaji dari cache kueri, meskipun kueri yang sama telah dijalankan sebelumnya. Petunjuk segera muncul setelah pernyataan `SELECT` dalam kueri. Banyak contoh kueri paralel dalam topik ini mencakup petunjuk ini, untuk menjadikan waktu kueri sebanding antara versi kueri dengan kueri paralel yang diaktifkan dan dinonaktifkan.

Pastikan Anda menghapus petunjuk ini dari sumber Anda saat beralih ke penggunaan kueri paralel untuk produksi.

Petunjuk pengoptimal

Cara lain untuk mengontrol pengoptimal adalah dengan menggunakan petunjuk pengoptimal, yang dapat ditentukan dalam pernyataan individual. Misalnya, Anda dapat mengaktifkan pengoptimalan untuk satu tabel dalam sebuah pernyataan, kemudian mematikan pengoptimalan untuk tabel yang berbeda. Untuk informasi selengkapnya tentang petunjuk ini, lihat Petunjuk Pengoptimal di Panduan Referensi MySQL.

Anda dapat menggunakan petunjuk SQL dengan kueri Aurora MySQL untuk kinerja yang sempurna. Anda juga dapat menggunakan petunjuk agar rencana eksekusi untuk kueri penting tidak berubah karena kondisi yang tidak dapat diprediksi.

Kami telah memperluas fitur petunjuk SQL untuk membantu Anda mengontrol pilihan pengoptimal untuk rencana kueri Anda. Petunjuk ini berlaku untuk kueri yang menggunakan pengoptimalan kueri paralel. Untuk informasi selengkapnya, lihat Petunjuk Aurora MySQL.

Tabel sementara MyISAM

Pengoptimalan kueri paralel hanya berlaku untuk tabel InnoDB. Karena Aurora MySQL menggunakan MyISAM di balik layar untuk tabel sementara, fase kueri internal yang mencakup tabel sementara tidak pernah menggunakan kueri paralel. Fase kueri ini ditunjukkan oleh `Using temporary` dalam output `EXPLAIN`.

Menggunakan Audit Lanjutan dengan klaster DB Amazon Aurora MySQL

Anda dapat menggunakan fitur Audit Lanjutan beperforma tinggi di Amazon Aurora MySQL untuk mengaudit aktivitas basis data. Untuk melakukannya, Anda mengaktifkan pengumpulan log audit dengan mengatur beberapa parameter klaster DB. Ketika Audit Lanjutan diaktifkan, Anda dapat menggunakannya untuk mencatat log kombinasi peristiwa yang didukung.

Anda dapat melihat atau mengunduh log audit untuk meninjau informasi audit untuk instans DB satu per satu. Untuk melakukannya, Anda dapat menggunakan prosedur dalam Memantau file log Amazon Aurora.

> **Tip**
>
> Untuk klaster DB Aurora yang berisi beberapa instans DB, Anda mungkin merasa lebih praktis untuk memeriksa log audit untuk semua instans di klaster. Untuk melakukannya, Anda dapat menggunakan Log CloudWatch. Anda dapat mengaktifkan pengaturan di tingkat klaster untuk menerbitkan data log audit Aurora MySQL ke grup log di CloudWatch. Kemudian, Anda dapat melihat, memfilter, dan mencari log audit melalui antarmuka CloudWatch. Untuk informasi selengkapnya, lihat Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch .

Mengaktifkan Audit Lanjutan

Gunakan parameter yang dijelaskan di bagian ini untuk mengaktifkan dan mengonfigurasi Audit Lanjutan untuk klaster DB Anda.

Gunakan parameter `server_audit_logging` untuk mengaktifkan atau menonaktifkan Audit Lanjutan.

Gunakan parameter `server_audit_events` untuk menentukan peristiwa apa yang akan dicatat lognya.

Gunakan `server_audit_incl_users` dan `server_audit_excl_users` untuk menentukan pengguna yang akan diaudit. Secara default, semua pengguna diaudit. Untuk detail tentang cara kerja parameter ini ketika salah satu atau keduanya dibiarkan kosong, atau nama pengguna yang sama ditentukan di keduanya, lihat server_audit_incl_users dan server_audit_excl_users.

Konfigurasikan Audit Lanjutan dengan mengatur parameter ini di grup parameter yang digunakan oleh klaster DB Anda. Anda dapat menggunakan prosedur yang ditunjukkan dalam Memodifikasi parameter dalam grup parameter DB untuk mengubah parameter klaster DB menggunakan AWS Management Console. Anda dapat menggunakan perintah modify-db-cluster-parameter-group AWS CLI atau operasi API Amazon RDS ModifyDBClusterParameterGroup untuk mengubah parameter klaster DB secara programatis.

Memodifikasi parameter ini tidak memerlukan pengaktifan ulang klaster DB saat grup parameter sudah dikaitkan dengan klaster Anda. Saat Anda mengaitkan grup parameter dengan klaster untuk pertama kalinya, pengaktifan ulang klaster diperlukan.

Topik

- server_audit_logging
- server_audit_events
- server_audit_incl_users
- server_audit_excl_users

server_audit_logging

Mengaktifkan atau menonaktifkan Audit Lanjutan. Parameter diatur secara default ini ke NONAKTIF; ubah ke AKTIF untuk mengaktifkan Audit Lanjutan.

Tidak ada data audit yang muncul di log kecuali jika Anda juga menentukan satu atau beberapa jenis peristiwa yang akan diaudit menggunakan parameter `server_audit_events`.

Untuk mengonfirmasi bahwa data audit dicatat lognya untuk instans DB, periksa apakah beberapa file log untuk instans tersebut memiliki nama form `audit/audit.log.`*`other_identifying_information`*. Untuk melihat nama file log, ikuti prosedur dalam Melihat dan mencantumkan file log basis data.

server_audit_events

Berisi daftar peristiwa yang dipisahkan koma yang akan dicatat lognya. Peristiwa harus ditentukan dalam huruf besar semua, dan tidak boleh ada spasi antara elemen daftar, misalnya: `CONNECT,QUERY_DDL`. Parameter ini diatur secara default ke string kosong.

Anda dapat mencatat log untuk kombinasi peristiwa berikut:

- CONNECT – Mencatat log koneksi yang berhasil maupun gagal dan juga pemutusan koneksi. Peristiwa ini mencakup informasi pengguna.
- QUERY – Mencatat log semua kueri dalam teks biasa, termasuk kueri yang gagal karena kesalahan sintaksis atau izin.

> **Tip**
>
> Dengan jenis peristiwa ini diaktifkan, data audit akan mencakup informasi tentang informasi pemantauan berkelanjutan dan pemeriksaan kondisi yang dilakukan Aurora secara otomatis. Jika Anda hanya tertarik pada jenis operasi tertentu, Anda dapat menggunakan jenis peristiwa yang lebih spesifik. Anda juga dapat menggunakan antarmuka CloudWatch untuk mencari dalam log untuk menemukan peristiwa yang terkait dengan basis data, tabel, atau pengguna tertentu.

- QUERY_DCL – Serupa dengan peristiwa QUERY, tetapi hanya menghasilkan kueri bahasa kontrol data (DCL) (GRANT, REVOKE, dan seterusnya).
- QUERY_DDL – Serupa dengan peristiwa QUERY, tetapi hanya menghasilkan kueri bahasa definisi data (DDL) (CREATE, ALTER, dan seterusnya).
- QUERY_DML – Serupa dengan peristiwa QUERY, tetapi hanya menghasilkan kueri bahasa manipulasi data (DML) (INSERT, UPDATE, dan seterusnya, serta SELECT).
- TABLE – Mencatat log tabel yang dipengaruhi oleh eksekusi kueri.

server_audit_incl_users

Berisi daftar nama pengguna yang dipisahkan koma untuk pengguna yang aktivitasnya dicatat dalam log. Tidak boleh ada spasi antara elemen daftar, misalnya: `user_3,user_4`. Parameter ini diatur secara default ke string kosong. Panjang maksimumnya adalah 1024 karakter. Nama pengguna yang ditentukan harus cocok dengan nilai yang sesuai di kolom `User` dalam tabel `mysql.user`. Untuk informasi selengkapnya tentang nama pengguna, lihat Account User Names and Passwords dalam dokumentasi MySQL.

Jika `server_audit_incl_users` dan `server_audit_excl_users` keduanya kosong (default), semua pengguna diaudit.

Jika Anda menambahkan pengguna ke `server_audit_incl_users` dan mengosongkan `server_audit_excl_users`, hanya pengguna tersebut yang diaudit.

Jika Anda menambahkan pengguna ke `server_audit_excl_users` dan mengosongkan `server_audit_incl_users`, semua pengguna diaudit, kecuali yang tercantum dalam `server_audit_excl_users`.

Jika Anda menambahkan pengguna yang sama ke `server_audit_excl_users` dan `server_audit_incl_users`, pengguna tersebut diaudit. Ketika pengguna yang sama tercantum di kedua pengaturan tersebut, `server_audit_incl_users` akan diberi prioritas yang lebih tinggi.

Peristiwa koneksi dan pemutusan koneksi tidak terpengaruh oleh variabel ini; peristiwa ini akan selalu dicatat lognya jika ditentukan. Pengguna dicatat lognya meskipun pengguna tersebut juga ditentukan dalam parameter `server_audit_excl_users` karena `server_audit_incl_users` memiliki prioritas yang lebih tinggi.

server_audit_excl_users

Berisi daftar nama pengguna yang dipisahkan koma untuk pengguna yang aktivitasnya tidak dicatat dalam log. Tidak boleh ada spasi antara elemen daftar, misalnya: `rdsadmin,user_1,user_2`. Parameter ini diatur secara default ke string kosong. Panjang maksimumnya adalah 1024 karakter. Nama pengguna yang ditentukan harus cocok dengan nilai yang sesuai di kolom `User` dalam tabel `mysql.user`. Untuk informasi selengkapnya tentang nama pengguna, lihat Account User Names and Passwords dalam dokumentasi MySQL.

Jika `server_audit_incl_users` dan `server_audit_excl_users` keduanya kosong (default), semua pengguna diaudit.

Jika Anda menambahkan pengguna ke `server_audit_excl_users` dan mengosongkan `server_audit_incl_users`, maka hanya pengguna yang Anda cantumkan di `server_audit_excl_users` yang tidak diaudit, dan semua pengguna lainnya diaudit.

Jika Anda menambahkan pengguna yang sama ke `server_audit_excl_users` dan `server_audit_incl_users`, pengguna tersebut diaudit. Ketika pengguna yang sama tercantum di kedua pengaturan tersebut, `server_audit_incl_users` akan diberi prioritas yang lebih tinggi.

Peristiwa koneksi dan pemutusan koneksi tidak terpengaruh oleh variabel ini; peristiwa ini akan selalu dicatat lognya jika ditentukan. Pengguna dicatat lognya jika pengguna tersebut juga ditentukan dalam parameter `server_audit_incl_users` karena pengaturan tersebut memiliki prioritas yang lebih tinggi daripada `server_audit_excl_users`.

Melihat log audit

Anda dapat melihat dan mengunduh log audit menggunakan konsol. Di halaman Basis data, pilih instans DB untuk menampilkan detailnya, kemudian gulir ke bagian Log. Log audit yang dihasilkan oleh fitur Audit Lanjutan memiliki nama form `audit/audit.log.`*`other_identifying_information`*.

Untuk mengunduh file log, pilih file tersebut di bagian Log lalu pilih Unduh.

Anda juga bisa mendapatkan daftar file log dengan menggunakan perintah describe-db-log-files AWS CLI. Anda dapat mengunduh konten file log dengan menggunakan perintah download-db-log-file-portion AWS CLI. Untuk informasi selengkapnya, lihat Melihat dan mencantumkan file log basis data dan Mengunduh file log basis data.

Detail log audit

File log direpresentasikan sebagai file variabel yang dipisahkan koma (CSV) dalam format UTF-8. Kueri juga di-wrapping dengan tanda kutip tunggal (').

Log audit disimpan secara terpisah di penyimpanan lokal (efemeral) dari setiap instans. Setiap instans Aurora mendistribusikan penulisan ke empat file log sekaligus. Ukuran maksimum log adalah 100 MB secara agregat. Ketika batas yang tidak dapat dikonfigurasi ini tercapai, Aurora akan merotasi file log dan menghasilkan empat file baru.

> **Tip**
>
> Entri file log tidak berurutan. Untuk mengurutkan entri, gunakan nilai stempel waktu. Untuk melihat peristiwa terbaru, Anda mungkin harus meninjau semua file log. Untuk fleksibilitas lebih dalam mengurutkan dan mencari data log, aktifkan pengaturan untuk mengunggah log audit ke CloudWatch dan melihatnya menggunakan antarmuka CloudWatch.
> Untuk melihat data audit dengan lebih banyak jenis bidang dan dengan output dalam format JSON, Anda juga dapat menggunakan fitur Database Activity Streams. Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.

File log audit berisi informasi yang dipisahkan koma berikut dalam baris, sesuai dengan urutan yang ditentukan:

Bidang	Deskripsi
timestamp	Stempel waktu Unix untuk peristiwa yang dicatat lognya dengan tingkat presisi mikrodetik.
serverhost	Nama instans yang dicatat lognya untuk peristiwa tersebut.
username	Nama pengguna yang terhubung milik pengguna.
host	Host yang digunakan pengguna untuk terhubung.
connectionid	Nomor ID koneksi untuk operasi yang dicatat lognya.
queryid	Nomor ID kueri, yang dapat digunakan untuk menemukan peristiwa tabel relasional dan kueri terkait. Untuk peristiwa `TABLE`, beberapa baris ditambahkan.
operation	Jenis tindakan yang tercatat. Kemungkinan nilainya adalah: `CONNECT`, `QUERY`, `READ`, `WRITE`, `CREATE`, `ALTER`, `RENAME`, dan `DROP`.
database	Basis data aktif, yang diatur oleh perintah `USE`.
object	Untuk peristiwa `QUERY`, nilai ini menunjukkan kueri yang dilakukan basis data. Untuk peristiwa `TABLE`, nilai ini menunjukkan nama tabel.
retcode	Kode yang dihasilkan dari operasi yang dicatat lognya.

Replikasi dengan Amazon Aurora MySQL

Fitur-fitur replikasi Aurora MySQL merupakan kunci untuk ketersediaan dan performa yang tinggi dari klaster Anda. Aurora mempermudah pembuatan atau perubahan ukuran klaster dengan maksimal 15 Replika Aurora.

Semua replika beroperasi dari data acuan yang sama. Jika beberapa instans basis data menjadi offline, yang lain akan tetap tersedia untuk melanjutkan pemrosesan kueri atau mengambil alih sebagai penulis jika diperlukan. Aurora akan secara otomatis menyebarkan koneksi hanya-baca Anda ke beberapa instans basis data, sehingga membantu klaster Aurora mendukung beban kerja sarat kueri.

Dalam topik berikut, Anda dapat menemukan informasi tentang cara kerja replikasi Aurora MySQL dan cara menyesuaikan pengaturan replikasi untuk ketersediaan dan performa terbaik.

Topik

- Menggunakan Replika Aurora
- Opsi replikasi untuk Amazon Aurora MySQL
- Pertimbangan performa untuk replikasi Amazon Aurora MySQL
- Zero-downtime restart (ZDR) untuk Amazon Aurora MySQL
- Mengonfigurasi filter replikasi dengan Aurora MySQL
- Memantau replikasi Amazon Aurora MySQL
- Menggunakan penerusan tulis lokal di klaster DB Amazon Aurora MySQL
- Mereplikasi klaster DB Amazon Aurora MySQL di seluruh Wilayah AWS
- Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner)
- Menggunakan replikasi berbasis GTID untuk Amazon Aurora MySQL

Menggunakan Replika Aurora

Replika Aurora adalah titik akhir independen dalam klaster basis data Aurora, yang paling berguna untuk menskalakan operasi baca dan meningkatkan ketersediaan. Hingga 15 Replika Aurora dapat didistribusikan ke seluruh Zona Ketersediaan yang dicakup oleh sebuah klaster DB di satu Wilayah AWS. Meskipun volume klaster DB terdiri dari beberapa salinan data untuk klaster DB, data dalam volume klaster direpresentasikan sebagai volume logis tunggal untuk instans primer dan Replika

Aurora dalam klaster DB tersebut. Untuk informasi selengkapnya tentang Replika Aurora, lihat Replika Aurora.

Replika Aurora berfungsi dengan baik untuk penskalaan baca karena ditujukan sepenuhnya untuk operasi baca pada volume klaster Anda. Operasi tulis dikelola oleh instans primer. Karena volume klaster dibagikan di antara semua instans dalam klaster DB Aurora MySQL Anda, tidak diperlukan pekerjaan tambahan untuk mereplikasi salinan data untuk setiap Replika Aurora. Sebaliknya, replika baca MySQL harus mengulang, pada thread tunggal, semua operasi tulis dari instans DB sumber daya ke penyimpanan data lokalnya. Batasan ini dapat memengaruhi kemampuan replika baca MySQL untuk mendukung volume lalu lintas baca yang besar.

Dengan Aurora MySQL, saat Replika Aurora dihapus, titik akhir instansnya akan segera dihapus, dan Replika Aurora akan dihapus dari titik akhir pembaca. Jika ada pernyataan yang berjalan di Replika Aurora yang dihapus, ada masa tenggang selama tiga menit. Pernyataan yang ada dapat diselesaikan dengan baik selama masa tenggang. Setelah masa tenggang berakhir, Replika Aurora akan dinonaktifkan dan dihapus.

Important

Replika Aurora untuk Aurora MySQL selalu menggunakan tingkat isolasi transaksi default `REPEATABLE READ` untuk operasi pada tabel InnoDB. Anda dapat menggunakan perintah `SET TRANSACTION ISOLATION LEVEL` untuk mengubah tingkat transaksi hanya untuk instans primer klaster DB Aurora MySQL. Pembatasan ini bertujuan untuk menghindari kunci tingkat pengguna pada Replika Aurora, dan memungkinkan penskalaan Replika Aurora untuk mendukung ribuan koneksi pengguna aktif sambil tetap menjaga lag replika seminimal mungkin.

Note

Pernyataan DDL yang berjalan pada instans primer dapat menginterupsi koneksi basis data pada Replika Aurora terkait. Jika koneksi Replika Aurora secara aktif menggunakan objek basis data, seperti tabel, dan objek tersebut dimodifikasi pada instans primer menggunakan pernyataan DDL, koneksi Replika Aurora dapat terinterupsi.

> **Note**
>
> Wilayah Tiongkok (Ningxia) tidak mendukung replika baca lintas Wilayah.

Opsi replikasi untuk Amazon Aurora MySQL

Anda dapat mengatur replikasi dengan opsi-opsi berikut:

- Dua klaster DB Aurora MySQL di Wilayah AWS yang berbeda, dengan membuat replika baca lintas Wilayah dari klaster DB Aurora MySQL.

  Untuk informasi selengkapnya, lihat Mereplikasi klaster DB Amazon Aurora MySQL di seluruh Wilayah AWS.
- Dua klaster DB Aurora MySQL di Wilayah AWS yang sama, dengan menggunakan replikasi log biner (binlog) MySQL.

  Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner).
- Instans DB RDS for MySQL sebagai sumber dan klaster DB Aurora MySQL, dengan membuat replika baca Aurora dari instans DB RDS for MySQL.

  Anda dapat menggunakan pendekatan ini untuk memindahkan perubahan data yang ada dan yang sedang berlangsung ke Aurora MySQL selama migrasi ke Aurora. Untuk informasi selengkapnya, lihat Memigrasikan data dari instans DB RDS for MySQL ke klaster DB Amazon Aurora MySQL menggunakan replika baca Aurora.

  Anda juga dapat menggunakan pendekatan ini untuk meningkatkan skalabilitas kueri baca untuk data Anda. Caranya adalah dengan mengkueri data menggunakan satu atau beberapa instans DB dalam klaster Aurora MySQL hanya baca. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora untuk menskalakan baca untuk basis data MySQL Anda.
- Sebuah klaster DB Aurora MySQL di satu Wilayah AWS dan maksimal lima klaster DB Aurora MySQL hanya baca di Wilayah yang berbeda-beda, dengan membuat basis data global Aurora.

  Anda dapat menggunakan basis data global Aurora untuk mendukung aplikasi dengan jejak global. Klaster DB Aurora MySQL primer memiliki instans Penulis dan maksimal 15 Replika Aurora. Klaster DB Aurora MySQL sekunder hanya baca masing-masing dapat terdiri dari maksimal 16

Replika Aurora. Untuk informasi selengkapnya, lihat Menggunakan basis data global Amazon Aurora.

> **Note**
>
> Me-boot ulang instans primer klaster DB Amazon Aurora juga akan secara otomatis mem-boot ulang Replika Aurora untuk klaster DB tersebut. Hal ini bertujuan untuk menetapkan kembali titik masuk yang menjamin konsistensi baca/tulis di seluruh klaster DB.

Pertimbangan performa untuk replikasi Amazon Aurora MySQL

Fitur berikut dapat membantu Anda menyesuaikan performa replikasi Aurora MySQL.

Fitur kompresi log replika secara otomatis mengurangi bandwidth jaringan untuk pesan replikasi. Karena setiap pesan ditransmisikan ke semua Replika Aurora, manfaat yang diberikan akan lebih besar untuk klaster yang lebih besar. Fitur ini memerlukan beberapa overhead CPU pada simpul penulis untuk melakukan kompresi. Fitur ini selalu diaktifkan di Aurora MySQL versi 2 dan versi 3.

Fitur pemfilteran binlog secara otomatis mengurangi bandwidth jaringan untuk pesan replikasi. Karena Replika Aurora tidak menggunakan informasi binlog yang disertakan dalam pesan replikasi, data tersebut dihilangkan dari pesan yang dikirim ke simpul-simpul tersebut.

Di Aurora MySQL versi 2, Anda dapat mengontrol fitur ini dengan mengubah parameter `aurora_enable_repl_bin_log_filtering`. Parameter ini aktif secara default. Karena optimisasi ini dimaksudkan agar bersifat transparan, Anda dapat menonaktifkan pengaturan ini hanya selama diagnosis atau pemecahan masalah yang terkait dengan replikasi. Misalnya, Anda dapat melakukannya untuk mencocokkan dengan perilaku klaster Aurora MySQL lama yang tidak menyediakan fitur ini.

Pemfilteran Binlog selalu diaktifkan di Aurora MySQL versi 3.

Zero-downtime restart (ZDR) untuk Amazon Aurora MySQL

Fitur zero-downtime restart (ZDR) dapat mempertahankan beberapa atau semua koneksi aktif ke instans DB selama pengaktifan ulang dari jenis tertentu. ZDR berlaku pada pengaktifan ulang yang dilakukan Aurora secara otomatis untuk mengatasi kondisi kesalahan, misalnya saat replika mulai tertinggal terlalu jauh di belakang sumbernya.

> **Important**
>
> Mekanisme ZDR beroperasi dengan basis upaya terbaik. Versi Aurora MySQL, kelas instans, kondisi kesalahan, operasi SQL yang kompatibel, dan faktor lain yang menentukan penerapan ZDR dapat berubah kapan saja.

ZDR untuk Aurora MySQL 2.x membutuhkan versi 2.10 dan lebih tinggi. ZDR tersedia di semua versi minor Aurora MySQL 3.x. Di Aurora MySQL versi 2 dan 3, mekanisme ZDR diaktifkan secara default dan Aurora tidak menggunakan parameter `aurora_enable_zdr`.

Di halaman Peristiwa, Aurora melaporkan aktivitas yang terkait dengan zero-downtime restart. Aurora mencatat peristiwa ketika mencoba pengaktifan ulang menggunakan mekanisme ZDR. Peristiwa ini menyatakan alasan Aurora melakukan pengaktifan ulang. Kemudian, Aurora mencatat peristiwa lain ketika pengaktifan ulang selesai. Peristiwa akhir ini melaporkan berapa lama prosesnya, dan berapa banyak koneksi yang dipertahankan atau diputus selama pengaktifan ulang. Anda dapat melihat log kesalahan basis data untuk melihat detail selengkapnya tentang apa yang terjadi selama pengaktifan ulang.

Meskipun koneksi tetap utuh setelah operasi ZDR yang sukses, beberapa variabel dan fitur diinisialisasi ulang. Jenis informasi berikut ini tidak dipertahankan selama pengaktifan ulang yang disebabkan oleh zero-downtime restart:

- Variabel global. Aurora memulihkan variabel sesi, tetapi tidak memulihkan variabel global setelah pengaktifan ulang.
- Variabel status. Secara khusus, nilai uptime yang dilaporkan oleh status mesin akan direset.
- `LAST_INSERT_ID`.
- Status `auto_increment` dalam memori untuk tabel. Status inkremen otomatis dalam memori diinisialisasi ulang. Untuk informasi selengkapnya tentang nilai inkremen otomatis, lihat Panduan Referensi MySQL.
- Informasi diagnostik dari tabel `INFORMATION_SCHEMA` dan `PERFORMANCE_SCHEMA`. Informasi diagnostik ini juga muncul dalam output perintah seperti `SHOW PROFILE` dan `SHOW PROFILES`.

Tabel berikut menunjukkan versi, peran instance, dan keadaan lain yang menentukan apakah Aurora dapat menggunakan mekanisme ZDR saat memulai ulang instans DB di cluster Anda.

Versi Aurora MySQL	ZDR berlaku untuk penulis?	ZDR berlaku untuk pembaca?	ZDR selalu diaktifkan?	Catatan
2.x, lebih rendah dari 2.10.0	Tidak	Tidak	N/A	ZDR tidak tersedia untuk versi ini.
2.10.0—2.11.0	Ya	Ya	Ya	Aurora mengembalikan semua transaksi yang sedang berlangsung pada koneksi aktif. Aplikasi Anda harus mencoba lagi transaksi. Aurora membatalkan koneksi apa pun yang menggunakan TLS/SSL, tabel sementara, kunci tabel, atau kunci pengguna.
2.11.1 dan lebih tinggi	Ya	Ya	Ya	Aurora mengembalikan semua transaksi yang sedang berlangsung pada koneksi aktif. Aplikasi Anda harus mencoba lagi transaksi. Aurora membatalkan koneksi apa pun yang menggunakan tabel sementara, kunci meja, atau kunci pengguna.
3.01—3.03	Ya	Ya	Ya	Aurora mengembalikan semua transaksi yang sedang berlangsung pada koneksi aktif. Aplikasi Anda harus mencoba lagi transaksi. Aurora membatalkan koneksi apa pun yang menggunakan TLS/SSL, tabel sementara, kunci tabel, atau kunci pengguna.
3.04 dan lebih tinggi	Ya	Ya	Ya	Aurora mengembalikan semua transaksi yang sedang berlangsung pada koneksi aktif. Aplikasi Anda harus mencoba lagi transaksi.

Versi Aurora MySQL	ZDR berlaku untuk penulis?	ZDR berlaku untuk pembaca?	ZDR selalu diaktifkan?	Catatan
				Aurora membatalkan koneksi apa pun yang menggunakan tabel sementara, kunci meja, atau kunci pengguna.

Mengonfigurasi filter replikasi dengan Aurora MySQL

Anda dapat menggunakan filter replikasi untuk mengetahui basis data dan tabel mana yang direplikasi dengan replika baca. Filter replikasi dapat menyertakan basis data dan tabel ke dalam replikasi atau mengecualikan mereka dari replikasi.

Berikut ini adalah beberapa kasus penggunaan untuk replikasi filter:

- Untuk mengurangi ukuran replika baca. Dengan filter replikasi, Anda dapat mengecualikan basis data dan tabel yang tidak diperlukan pada replika baca.
- Untuk mengecualikan basis data dan tabel dari replika baca untuk alasan keamanan.
- Untuk mereplikasi basis data yang berbeda dan tabel untuk kasus penggunaan tertentu di replika baca yang berbeda. Misalnya, Anda mungkin menggunakan replika baca khusus untuk analitik atau sharding.
- Untuk klaster DB yang memiliki replika baca di Wilayah AWS yang berbeda, untuk mereplikasi basis data atau tabel yang berbeda di Wilayah AWS yang berbeda.
- Untuk menentukan basis data dan tabel mana yang direplikasi dengan klaster DB Aurora MySQL yang dikonfigurasi sebagai replika dalam topologi replikasi masuk. Untuk informasi selengkapnya tentang konfigurasi ini, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner).

Topik

- Mengatur parameter pemfilteran replikasi untuk Aurora MySQL
- Batasan pemfilteran replikasi untuk Aurora MySQL
- Contoh pemfilteran replikasi untuk Aurora MySQL
- Melihat filter replikasi untuk replika baca

Mengatur parameter pemfilteran replikasi untuk Aurora MySQL

Untuk mengonfigurasi filter replikasi, atur parameter berikut:

- `binlog-do-db` – Mereplikasi perubahan ke log biner yang ditentukan. Ketika Anda mengatur parameter ini untuk klaster sumber binlog, hanya log biner yang ditentukan dalam parameter yang direplikasi.
- `binlog-ignore-db` – Jangan mereplikasi ke log biner yang ditentukan. Ketika parameter `binlog-do-db` diatur untuk klaster sumber binlog, parameter ini tidak dievaluasi.
- `replicate-do-db` – Mereplikasi perubahan ke basis data yang ditentukan. Ketika Anda mengatur parameter ini untuk klaster replika binlog, hanya basis data yang ditentukan dalam parameter yang direplikasi.
- `replicate-ignore-db` – Jangan mereplikasi perubahan ke basis data yang ditentukan. Ketika parameter `replicate-do-db` diatur untuk klaster replika binlog, parameter ini tidak dievaluasi.
- `replicate-do-table` – Mereplikasi perubahan ke tabel yang ditentukan. Ketika Anda menetapkan parameter ini untuk replika baca, hanya tabel yang ditentukan dalam parameter yang direplikasi. Selain itu, ketika parameter `replicate-ignore-db` atau `replicate-do-db` diatur, pastikan untuk menyertakan basis data yang mencakup tabel tertentu dalam replikasi dengan klaster replika binlog.
- `replicate-ignore-table` – Jangan mereplikasi perubahan ke tabel yang ditentukan. Ketika parameter `replicate-do-table` diatur untuk klaster replika binlog, parameter ini tidak dievaluasi.
- `replicate-wild-do-table` – Mereplikasi tabel berdasarkan basis data dan pola nama tabel yang ditentukan. Karakter wildcard `%` dan `_` didukung. Ketika parameter `replicate-ignore-db` atau `replicate-do-db` diatur, pastikan untuk menyertakan basis data yang mencakup tabel tertentu dalam replikasi dengan klaster replika binlog.
- `replicate-wild-ignore-table` – Jangan mereplikasi tabel berdasarkan basis data dan pola nama tabel yang ditentukan. Karakter wildcard `%` dan `_` didukung. Ketika parameter `replicate-wild-do-table` atau `replicate-do-table` diatur untuk klaster replika binlog, parameter ini tidak dievaluasi.

Parameter dievaluasi sesuai dengan urutannya dalam daftar. Untuk informasi selengkapnya tentang bagaimana parameter ini bekerja, lihat dokumentasi MySQL:

- Untuk informasi umum, lihat Opsi dan Variabel Server Replika.

- Untuk informasi tentang bagaimana parameter filter replikasi basis data dievaluasi, lihat Evaluasi Opsi Replikasi Tingkat Basis Data dan Logging Biner.
- Untuk informasi tentang bagaimana parameter filter replikasi tabel dievaluasi, lihat Evaluasi Opsi Replikasi Tingkat Tabel.

Secara default, masing-masing parameter ini memiliki nilai kosong. Pada setiap klaster binlog, Anda dapat menggunakan parameter ini untuk mengatur, mengubah, dan menghapus filter replikasi. Ketika Anda menetapkan salah satu parameter ini, pisahkan masing-masing filter dari yang lain dengan koma.

Anda dapat menggunakan karakter wildcard `%` dan `_` dalam parameter `replicate-wild-do-table` dan `replicate-wild-ignore-table`. Parameter wildcard `%` mencocokkan dengan sejumlah karakter, dan wildcard `_` hanya mencocokkan satu karakter.

Format logging biner dari instans DB sumber penting untuk replikasi karena menentukan catatan perubahan data. Pengaturan parameter `binlog_format` menentukan apakah replikasi berbasis baris atau berbasis pernyataan. Untuk informasi selengkapnya, lihat Mengkonfigurasi pengelogan biner Aurora MySQL.

> **Note**
>
> Semua pernyataan bahasa definisi data (DDL) direplikasi sebagai pernyataan, terlepas dari pengaturan `binlog_format` pada instans DB sumber.

Batasan pemfilteran replikasi untuk Aurora MySQL

Batasan berikut ini berlaku untuk pemfilteran replikasi untuk Aurora MySQL:

- Filter replikasi hanya didukung untuk Aurora MySQL versi 3.
- Setiap parameter filter replikasi memiliki batas 2.000 karakter.
- Koma tidak didukung dalam filter replikasi.
- Filter replikasi tidak mendukung transaksi XA.

  Untuk informasi selengkapnya, lihat Restrictions on XA Transactions dalam dokumentasi MySQL.

Contoh pemfilteran replikasi untuk Aurora MySQL

Untuk mengonfigurasi filter replikasi untuk replika baca, modifikasi parameter filter replikasi dalam grup parameter klaster DB yang terkait dengan replika baca.

> **Note**
>
> Anda tidak dapat memodifikasi grup parameter klaster DB default. Jika replika baca menggunakan grup parameter default, buat grup parameter baru dan kaitkan dengan replika baca tersebut. Untuk informasi selengkapnya tentang grup parameter klaster DB, lihat Bekerja dengan grup parameter.

Anda dapat mengatur parameter dalam grup parameter klaster DB menggunakan AWS Management Console, AWS CLI, atau API RDS. Untuk informasi tentang mengatur parameter, lihat Memodifikasi parameter dalam grup parameter DB. Ketika Anda mengatur parameter dalam grup parameter klaster DB, semua klaster DB yang terkait dengan grup parameter tersebut akan menggunakan pengaturan parameter. Jika Anda mengatur parameter filter replikasi dalam grup parameter klaster DB, pastikan bahwa grup parameter dikaitkan hanya dengan klaster replika baca. Biarkan parameter filter replikasi kosong untuk instans DB sumber.

Contoh berikut mengatur parameter menggunakan AWS CLI. Contoh ini menetapkan `ApplyMethod` ke `immediate` sehingga perubahan parameter terjadi segera setelah perintah CLI selesai. Jika Anda ingin menerapkan perubahan tertunda setelah replika baca di-reboot, atur `ApplyMethod` ke `pending-reboot`.

Contoh berikut mengatur filter replikasi:

- Including databases in replication
- Including tables in replication
- Including tables in replication with wildcard characters
- Excluding databases from replication
- Excluding tables from replication
- Excluding tables from replication using wildcard characters

Example Termasuk basis data dalam replikasi

Contoh berikut menyertakan basis data `mydb1` dan `mydb2` dalam replikasi.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
db,ParameterValue='mydb1,mydb2',ApplyMethod=immediate"
```

Example Termasuk tabel dalam replikasi

Contoh berikut menyertakan tabel `table1` dan `table2` dalam basis data `mydb1` dalam replikasi.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-do-
table,ParameterValue='mydb1.table1,mydb1.table2',ApplyMethod=immediate"
```

Example Menyertakan tabel dalam replikasi menggunakan karakter wildcard

Contoh berikut menyertakan tabel dengan nama berawalan `order` dan `return` dalam basis data `mydb` dalam replikasi.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-do-table,ParameterValue='mydb.order
%,mydb.return%',ApplyMethod=immediate"
```

Example Mengecualikan basis data dari replikasi

Contoh berikut mengecualikan basis data `mydb5` dan `mydb6` dari replikasi.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6',ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
db,ParameterValue='mydb5,mydb6,ApplyMethod=immediate"
```

Example Mengecualikan tabel dari replikasi

Contoh berikut tidak termasuk tabel `table1` dalam database `mydb5` dan `table2` database `mydb6` dari replikasi.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
```

```
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-ignore-
table,ParameterValue='mydb5.table1,mydb6.table2',ApplyMethod=immediate"
```

Example Mengecualikan tabel dari replikasi menggunakan karakter wildcard

Contoh berikut mengecualikan tabel dengan nama berawalan `order` dan `return` dalam basis data `mydb7` dari replikasi.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
  --db-cluster-parameter-group-name myparametergroup \
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
  --db-cluster-parameter-group-name myparametergroup ^
  --parameters "ParameterName=replicate-wild-ignore-table,ParameterValue='mydb7.order
%,mydb7.return%',ApplyMethod=immediate"
```

Melihat filter replikasi untuk replika baca

Anda dapat melihat filter replikasi untuk replika baca dengan cara berikut:

- Memeriksa pengaturan parameter filter replikasi dalam grup parameter yang terkait dengan replika baca.

  Untuk petunjuknya, lihat Melihat nilai parameter untuk grup parameter DB.
- Dalam klien MySQL, hubungkan ke replika baca dan jalankan pernyataan `SHOW REPLICA STATUS`.

Dalam output, bidang berikut menunjukkan filter replikasi untuk replika baca:

- `Binlog_Do_DB`
- `Binlog_Ignore_DB`
- `Replicate_Do_DB`
- `Replicate_Ignore_DB`
- `Replicate_Do_Table`
- `Replicate_Ignore_Table`
- `Replicate_Wild_Do_Table`
- `Replicate_Wild_Ignore_Table`

Untuk informasi selengkapnya tentang bidang ini, lihat Checking Replication Status dalam dokumentasi MySQL.

Memantau replikasi Amazon Aurora MySQL

Penskalaan baca dan ketersediaan yang tinggi tergantung dari waktu lag minimal. Anda dapat memantau seberapa jauh Aurora Replica tertinggal dari instance utama cluster DB MySQL Aurora Anda dengan memantau metrik Amazon. CloudWatch `AuroraReplicaLag` Metrik `AuroraReplicaLag` dicatat dalam setiap Replika Aurora.

Instans primer DB juga merekam metrik `AuroraReplicaLagMaximum` dan `AuroraReplicaLagMinimum` Amazon CloudWatch. Metrik `AuroraReplicaLagMaximum` akan mencatat jumlah maksimum lag antara instans DB primer dan setiap Replika Aurora dalam klaster DB. Metrik `AuroraReplicaLagMinimum` akan mencatat jumlah minimum lag antara instans DB primer dan setiap Replika Aurora dalam klaster DB.

Jika Anda membutuhkan nilai terbaru untuk lag Aurora Replica, Anda dapat memeriksa metrik `AuroraReplicaLag` di Amazon. CloudWatch Lag Replika Aurora juga dicatat pada setiap Replika Aurora dari klaster DB Aurora MySQL Anda dalam tabel `information_schema.replica_host_status`. Untuk informasi selengkapnya tentang tabel ini, lihat information_schema.replica_host_status.

Untuk informasi selengkapnya tentang pemantauan instans dan CloudWatch metrik RDS, lihat. Memantau metrik di klaster Amazon Aurora

Menggunakan penerusan tulis lokal di klaster DB Amazon Aurora MySQL

Penerusan tulis lokal (dalam klaster) memungkinkan aplikasi Anda mengeluarkan transaksi baca/tulis langsung pada Replika Aurora. Transaksi ini kemudian diteruskan ke instans DB penulis untuk di-commit. Anda dapat menggunakan penerusan tulis lokal ketika aplikasi Anda memerlukan konsistensi baca-setelah-tulis, yang merupakan kemampuan untuk membaca penulisan terbaru dalam suatu transaksi.

Replika baca menerima pembaruan secara asinkron dari penulis. Tanpa penerusan tulis, Anda harus bertransaksi pembacaan apa pun yang memerlukan konsistensi baca-setelah-tulis pada instans DB penulis. Atau Anda harus mengembangkan logika aplikasi khusus yang kompleks untuk memanfaatkan beberapa replika baca untuk skalabilitas. Aplikasi Anda harus sepenuhnya membagi semua lalu lintas baca dan tulis, sehingga mempertahankan dua set koneksi basis data untuk mengirim lalu lintas ke titik akhir yang benar. Overhead pengembangan ini memperumit desain aplikasi ketika kueri merupakan bagian dari satu sesi logis, atau transaksi, di dalam aplikasi. Selain itu, karena lag replikasi dapat berbeda di antara replika baca, sulit untuk mencapai konsistensi baca global di semua instans di dalam basis data.

Penerusan tulis menghindari kebutuhan untuk membagi transaksi tersebut atau mengirimkannya secara eksklusif ke penulis, yang menyederhanakan pengembangan aplikasi. Kemampuan baru ini memudahkan pencapaian skala baca untuk beban kerja yang perlu membaca penulisan terbaru dalam suatu transaksi dan tidak sensitif terhadap latensi penulisan.

Penerusan tulis lokal berbeda dari penerusan tulis global, yang meneruskan penulisan dari klaster DB sekunder ke klaster DB primer dalam basis data global Aurora. Anda dapat menggunakan penerusan tulis lokal dalam klaster DB yang merupakan bagian dari basis data global Aurora. Untuk informasi selengkapnya, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.

Penerusan tulis lokal membutuhkan Aurora MySQL versi 3.04 atau versi yang lebih tinggi.

Topik

- Mengaktifkan penerusan tulis lokal
- Memeriksa apakah klaster DB memiliki penerusan tulis yang diaktifkan
- Kompatibilitas aplikasi dan SQL dengan penerusan tulis
- Tingkat isolasi untuk penerusan tulis
- Konsistensi baca untuk penerusan tulis
- Menjalankan pernyataan multibagian dengan penerusan tulis

- Transaksi dengan penerusan tulis
- Parameter konfigurasi untuk penerusan tulis
- Metrik Amazon CloudWatch dan variabel status Aurora MySQL untuk penerusan tulis
- Mengidentifikasi transaksi dan kueri yang diteruskan

Mengaktifkan penerusan tulis lokal

Secara default, penerusan tulis lokal tidak diaktifkan untuk klaster DB MySQL Aurora. Anda mengaktifkan penerusan tulis lokal pada tingkat klaster, bukan pada tingkat instans.

> **Important**
>
> Anda juga dapat mengaktifkan penerusan tulis lokal untuk replika baca lintas wilayah yang menggunakan pencatatan log biner, tetapi operasi tulis tidak diteruskan ke Wilayah AWS sumber. Operasi tulis diteruskan ke instans DB penulis dari klaster replika baca binlog. Gunakan metode ini hanya jika Anda memiliki kasus penggunaan penulisan ke replika baca binlog di Wilayah AWS sekunder. Jika tidak, Anda mungkin berakhir dengan skenario “otak terbelah” di mana set data yang direplikasi tidak konsisten satu sama lain.
> Kami menyarankan Anda menggunakan penerusan tulis global dengan basis data global, daripada penerusan tulis lokal pada replika baca lintas wilayah, kecuali benar-benar diperlukan. Untuk informasi selengkapnya, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.

Konsol

Menggunakan AWS Management Console, pilih kotak centang Aktifkan penerusan tulis lokal di bawah Penerusan tulis replika baca saat Anda membuat atau memodifikasi klaster DB.

AWS CLI

Untuk mengaktifkan penerusan tulis dengan AWS CLI, gunakan opsi `--enable-local-write-forwarding`. Opsi ini berfungsi saat Anda membuat klaster DB baru menggunakan perintah `create-db-cluster`. Opsi ini juga berfungsi saat Anda memodifikasi klaster DB yang ada dengan menggunakan perintah `modify-db-cluster`. Anda dapat menonaktifkan penerusan tulis dengan menggunakan opsi `--no-enable-local-write-forwarding` dengan perintah CLI yang sama ini.

Contoh berikut ini membuat sebuah klaster DB Aurora MySQL dengan penerusan tulis yang diaktifkan.

```
aws rds create-db-cluster \
  --db-cluster-identifier write-forwarding-test-cluster \
  --enable-local-write-forwarding \
  --engine aurora-mysql \
  --engine-version 8.0.mysql_aurora.3.04.0 \
  --master-username myuser \
  --master-user-password mypassword \
  --backup-retention 1
```

Anda kemudian membuat instans DB penulis dan pembaca sehingga Anda dapat menggunakan penerusan tulis. Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.

API RDS

Untuk mengaktifkan penerusan tulis menggunakan API Amazon RDS, atur parameter `EnableLocalWriteForwarding` ke `true`. Parameter ini berfungsi saat Anda membuat klaster DB baru menggunakan operasi `CreateDBCluster`. Opsi ini juga berfungsi saat Anda memodifikasi klaster DB yang ada dengan menggunakan operasi `ModifyDBCluster`. Anda dapat menonaktifkan penerusan tulis dengan mengatur parameter `EnableLocalWriteForwarding` ke `false`.

Mengaktifkan penerusan tulis untuk sesi basis data

Parameter `aurora_replica_read_consistency` adalah parameter DB dan parameter klaster DB yang memungkinkan penerusan tulis. Anda dapat menentukan `EVENTUAL`, `SESSION`, atau `GLOBAL` untuk tingkat konsistensi baca. Untuk mempelajari selengkapnya tentang tingkat konsistensi, lihat Konsistensi baca untuk penerusan tulis.

Aturan berikut berlaku untuk parameter ini:

- Nilai default-nya adalah " (nol).
- Penerusan tulis hanya tersedia jika Anda menyetel `aurora_replica_read_consistency` ke `EVENTUAL`, `SESSION`, atau `GLOBAL`. Parameter ini hanya relevan dalam instans pembaca klaster DB yang memiliki penerusan tulis diaktifkan.
- Anda tidak dapat mengatur parameter ini (saat kosong) atau membatalkan pengaturan (saat sudah diatur) di dalam transaksi multipernyataan. Anda dapat mengubahnya dari satu nilai yang valid ke nilai valid lainnya selama transaksi tersebut, tetapi kami tidak merekomendasikan tindakan ini.

Memeriksa apakah klaster DB memiliki penerusan tulis yang diaktifkan

Untuk menentukan bahwa Anda dapat menggunakan penerusan tulis di klaster DB, konfirmasikan bahwa klaster memiliki atribut `LocalWriteForwardingStatus` disetel ke `enabled`.

Di AWS Management Console, pada tab Konfigurasi halaman detail untuk klaster, Anda melihat status Diaktifkan untuk Penerusan tulis replika baca lokal.

Untuk melihat status pengaturan penerusan tulis untuk semua klaster Anda, jalankan perintah AWS CLI berikut.

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,LocalWriteForwardingStatus:LocalWriteForwardingStatus}

[
    {
        "LocalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-1"
    },
    {
        "LocalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "write-forwarding-test-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "requested",
        "DBClusterIdentifier": "test-global-cluster-2"
    },
    {
        "LocalWriteForwardingStatus": "null",
        "DBClusterIdentifier": "aurora-mysql-v2-cluster"
    }
]
```

Sebuah klaster DB dapat memiliki nilai berikut untuk `LocalWriteForwardingStatus`:

- `disabled` – Penerusan tulis dinonaktifkan.
- `disabling` – Penerusan tulis dalam proses penonaktifan.
- `enabled` – Penerusan tulis diaktifkan.
- `enabling` – Penerusan tulis dalam proses pengaktifan.

- `null` – Penerusan tulis tidak tersedia untuk klaster DB ini.
- `requested` – Penerusan tulis telah diminta, tetapi belum aktif.

Kompatibilitas aplikasi dan SQL dengan penerusan tulis

Anda dapat menggunakan jenis pernyataan SQL berikut dengan penerusan tulis:

- Pernyataan bahasa manipulasi data (DML), seperti `INSERT`, `DELETE`, dan `UPDATE`. Ada beberapa pembatasan pada properti pernyataan ini yang dapat Anda gunakan dengan penerusan tulis, seperti yang dijelaskan di bawah ini.
- Pernyataan `SELECT ... LOCK IN SHARE MODE` dan `SELECT FOR UPDATE`.
- Pernyataan `PREPARE` dan `EXECUTE`.

Pernyataan tertentu tidak diizinkan atau dapat menghasilkan hasil usang saat Anda menggunakannya dalam klaster DB dengan penerusan tulis. Oleh karena itu, pengaturan `EnableLocalWriteForwarding` dinonaktifkan secara default untuk klaster DB. Sebelum diaktifkan, periksa untuk memastikan bahwa kode aplikasi Anda tidak terpengaruh oleh pembatasan ini.

Batasan berikut berlaku untuk pernyataan SQL yang Anda gunakan untuk penerusan tulis. Dalam beberapa kasus, Anda dapat menggunakan pernyataan pada klaster DB dengan penerusan tulis yang diaktifkan. Pendekatan ini berfungsi jika penerusan tulis tidak diaktifkan dalam sesi oleh parameter konfigurasi `aurora_replica_read_consistency`. Jika Anda mencoba menggunakan suatu pernyataan yang tidak diizinkan karena adanya penerusan tulis, Anda akan melihat pesan kesalahan seperti berikut ini:

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

Bahasa definisi data (DDL)

Hubungkan ke instans DB tulis untuk menjalankan pernyataan DDL. Anda tidak dapat menjalankannya dari instans DB pembaca.

Memperbarui tabel permanen menggunakan data dari tabel sementara

Anda dapat menggunakan tabel sementara pada klaster DB dengan penerusan tulis yang diaktifkan. Namun, Anda tidak dapat menggunakan pernyataan DML untuk mengubah tabel

permanen jika pernyataan tersebut mengacu pada tabel sementara. Misalnya, Anda tidak dapat menggunakan pernyataan `INSERT ... SELECT` yang mengambil data dari tabel sementara.

Transaksi XA

Anda tidak dapat menggunakan pernyataan berikut pada klaster DB saat penerusan tulis diaktifkan dalam sesi. Anda dapat menggunakan pernyataan ini pada klaster DB yang tidak memiliki penerusan tulis yang diaktifkan, atau dalam sesi dengan pengaturan `aurora_replica_read_consistency` yang kosong. Sebelum mengaktifkan penerusan tulis dalam sebuah sesi, periksa apakah kode Anda menggunakan pernyataan ini.

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

Pernyataan LOAD untuk tabel permanen

Anda tidak dapat menggunakan pernyataan berikut pada klaster DB dengan penerusan tulis yang diaktifkan.

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

Pernyataan plugin

Anda tidak dapat menggunakan pernyataan berikut pada klaster DB dengan penerusan tulis yang diaktifkan.

```
INSTALL PLUGIN example SONAME 'ha_example.so';
UNINSTALL PLUGIN example;
```

Pernyataan SAVEPOINT

Anda tidak dapat menggunakan pernyataan berikut pada klaster DB saat penerusan tulis diaktifkan dalam sesi. Anda dapat menggunakan pernyataan ini pada klaster DB yang tidak memiliki penerusan tulis yang diaktifkan, atau dalam sesi dengan pengaturan `aurora_replica_read_consistency` yang kosong. Periksa apakah kode Anda menggunakan pernyataan ini sebelum mengaktifkan penerusan tulis dalam sebuah sesi.

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

Tingkat isolasi untuk penerusan tulis

Dalam sesi yang menggunakan penerusan tulis, Anda hanya dapat menggunakan tingkat isolasi `REPEATABLE READ`. Meskipun Anda juga dapat menggunakan tingkat isolasi `READ COMMITTED` dengan Replika Aurora, tingkat isolasi tersebut tidak berfungsi dengan penerusan tulis. Untuk informasi tentang tingkat isolasi `REPEATABLE READ` dan `READ COMMITTED`, lihat Tingkat isolasi Aurora MySQL.

Konsistensi baca untuk penerusan tulis

Anda dapat mengontrol tingkat konsistensi baca di klaster DB. Tingkat konsistensi baca menentukan berapa lama klaster DB menunggu sebelum operasi baca untuk memastikan beberapa atau semua perubahan direplikasi dari penulis. Anda dapat menyesuaikan tingkat konsistensi baca untuk memastikan bahwa semua operasi tulis yang diteruskan dari sesi Anda terlihat dalam klaster DB sebelum kueri berikutnya. Anda juga dapat menggunakan pengaturan ini untuk memastikan kueri pada klaster DB selalu melihat pembaruan terkini dari penulis. Pengaturan ini juga berlaku untuk kueri yang dikirimkan oleh sesi lain atau klaster lainnya. Untuk menentukan jenis perilaku ini untuk aplikasi Anda, pilih nilai untuk parameter DB `aurora_replica_read_consistency` atau parameter klaster DB.

> **Important**
>
> Selalu atur parameter DB `aurora_replica_read_consistency` atau parameter klaster DB saat Anda ingin meneruskan penulisan. Jika Anda tidak mengaturnya, Aurora tidak akan meneruskan penulisan. Parameter ini mempunyai nilai kosong secara default, jadi pilih nilai tertentu apabila Anda menggunakan parameter ini. Parameter `aurora_replica_read_consistency` hanya memengaruhi klaster atau instans DB yang mengaktifkan penerusan tulis.

Saat Anda meningkatkan tingkat konsistensi, aplikasi Anda menghabiskan lebih banyak waktu untuk menunggu perubahan disebarkan di antara instans-instans DB. Anda dapat memilih keseimbangan

antara waktu respons yang cepat dan memastikan bahwa perubahan yang dilakukan dalam instans DB lainnya sepenuhnya tersedia sebelum kueri Anda berjalan.

Anda dapat menentukan nilai berikut untuk parameter `aurora_replica_read_consistency`:

- `EVENTUAL` – Hasil operasi tulis dalam sesi yang sama tidak terlihat sampai operasi tulis dilakukan pada instans DB penulis. Kueri tidak menunggu hasil yang diperbarui untuk tersedia. Dengan demikian, kueri dapat mengambil data yang lebih lama atau data yang diperbarui, bergantung pada waktu pernyataan dan jumlah lag replikasi. Ini adalah konsistensi yang sama seperti untuk klaster DB Aurora MySQL yang tidak menggunakan penerusan tulis.
- `SESSION` – Semua kueri yang menggunakan penerusan tulis melihat hasil semua perubahan yang dilakukan dalam sesi tersebut. Perubahan dapat dilihat terlepas dari apakah transaksi dilakukan. Jika perlu, kueri menunggu hasil operasi tulis yang diteruskan untuk direplikasi.
- `GLOBAL` – Sebuah sesi melihat semua perubahan yang dilakukan di semua sesi dan instans di dalam klaster DB. Setiap kueri mungkin akan menunggu selama periode yang berbeda-beda tergantung jumlah lag sesi. Kueri berlanjut saat klaster DB dimutakhirkan dengan semua data yang di-commit dari penulis, pada saat kueri dimulai.

Untuk informasi tentang parameter konfigurasi yang terlibat dalam penerusan tulis, lihat Parameter konfigurasi untuk penerusan tulis.

> **Note**
>
> Anda juga dapat menggunakan `aurora_replica_read_consistency` sebagai variabel sesi, misalnya:
>
> ```
> mysql> set aurora_replica_read_consistency = 'session';
> ```

Contoh menggunakan penerusan tulis

Contoh berikut ini menunjukkan efek parameter `aurora_replica_read_consistency` pada pernyataan `INSERT` yang berjalan diikuti oleh pernyataan `SELECT`. Hasilnya dapat berbeda, tergantung nilai `aurora_replica_read_consistency` dan waktu pernyataan.

Untuk mencapai konsistensi yang lebih tinggi, Anda mungkin menunggu sebentar sebelum mengeluarkan pernyataan `SELECT`. Atau Aurora dapat secara otomatis menunggu sampai hasil selesai mereplikasi sebelum melanjutkan dengan `SELECT`.

Untuk informasi tentang cara menetapkan parameter DB, lihat Bekerja dengan grup parameter.

Example dengan **aurora_replica_read_consistency** diatur ke **EVENTUAL**

Menjalankan pernyataan INSERT, yang langsung diikuti dengan pernyataan SELECT, memunculkan nilai untuk COUNT(*) dengan jumlah baris sebelum baris baru disisipkan. Dengan menjalankan SELECT lagi beberapa saat kemudian, muncul hitungan baris yang diperbarui. Pernyataan SELECT tidak menunggu.

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

Example dengan **aurora_replica_read_consistency** diatur ke **SESSION**

Pernyataan SELECT langsung setelah INSERT akan menunggu sampai perubahan dari pernyataan INSERT terlihat. Pernyataan SELECT selanjutnya tidak menunggu.

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
```

```
+----------+
1 row in set (0.01 sec)

mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.37 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

Dengan pengaturan konsistensi baca tetap disetel ke `SESSION`, memperkenalkan waktu tunggu singkat setelah melaksanakan pernyataan `INSERT` membuat hitungan baris yang diperbarui tersedia pada saat pernyataan `SELECT` berikutnya berjalan.

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
1 row in set (2.01 sec)
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

Example dengan **`aurora_replica_read_consistency`** diatur ke **`GLOBAL`**

Setiap pernyataan `SELECT` menunggu sampai semua perubahan data, sejak waktu mulai pernyataan, terlihat sebelum melakukan kueri. Waktu tunggu untuk setiap pernyataan `SELECT` bervariasi, tergantung jumlah lag replikasi.

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.37 sec)

mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.66 sec)
```

Menjalankan pernyataan multibagian dengan penerusan tulis

Pernyataan DML mungkin terdiri atas beberapa bagian, seperti pernyataan `INSERT ... SELECT` atau pernyataan `DELETE ... WHERE`. Dalam kasus ini, seluruh pernyataan diteruskan ke instans DB tulis dan berjalan di sana.

Transaksi dengan penerusan tulis

Jika mode akses transaksi disetel ke hanya-baca, penerusan tulis tidak digunakan. Anda dapat menentukan mode akses untuk transaksi dengan menggunakan pernyataan `SET TRANSACTION` atau pernyataan `START TRANSACTION`. Anda juga dapat menentukan mode akses transaksi dengan mengubah nilai variabel sesi transaction_read_only. Anda dapat mengubah nilai sesi ini hanya saat Anda terhubung dengan klaster DB yang memiliki penerusan tulis diaktifkan.

Jika transaksi yang berlangsung lama tidak mengeluarkan pernyataan apa pun dalam waktu yang lama, transaksi tersebut kemungkinan melebihi periode batas waktu idle. Periode ini memiliki nilai default satu menit. Anda dapat mengatur parameter `aurora_fwd_writer_idle_timeout` untuk

meningkatkannya hingga satu hari. Transaksi yang melebihi batas waktu idle dibatalkan oleh instans tulis. Pernyataan berikutnya yang Anda kirimkan akan menerima kesalahan waktu habis. Kemudian Aurora akan mengembalikan transaksi.

Jenis kesalahan ini dapat terjadi dalam kasus lain ketika penerusan tulis menjadi tidak tersedia. Misalnya, Aurora membatalkan transaksi apa pun yang menggunakan penerusan tulis jika Anda memulai ulang klaster DB atau jika Anda menonaktifkan penerusan tulis.

Ketika instans penulis dalam sebuah klaster yang menggunakan penerusan tulis lokal dimulai ulang, transaksi dan kueri yang aktif dan diteruskan pada instans pembaca yang menggunakan penerusan tulis lokal akan ditutup secara otomatis. Setelah instans penulis tersedia lagi, Anda dapat mencoba kembali transaksi ini.

Parameter konfigurasi untuk penerusan tulis

Grup parameter DB Aurora mencakup pengaturan untuk fitur penerusan tulis. Detail tentang parameter ini dirangkum dalam tabel berikut, dengan catatan penggunaan setelah tabel.

Parameter	Cakupan	Tipe	Nilai default	Nilai valid
`aurora_fwd_writer_idle_timeout`	Klaster	Integer tanpa tanda	60	1-86.400
`aurora_fwd_writer_max_connections_pct`	Klaster	Integer panjang tanpa tanda	10	0–90
`aurora_replica_read_consistency`	Klaster atau instans	Enum	" (null)	`EVENTUAL`, `SESSION`, `GLOBAL`

Untuk mengontrol permintaan tulis yang masuk, gunakan pengaturan ini:

- `aurora_fwd_writer_idle_timeout` – Jumlah detik instans yang diperlukan oleh instans DB tulis untuk menunggu aktivitas pada koneksi yang diteruskan dari instans pembaca sebelum menutupnya. Jika sesi tetap idle setelah periode ini, Aurora akan membatalkan sesi.

- `aurora_fwd_writer_max_connections_pct` – Batas atas pada koneksi basis data yang dapat digunakan pada instans DB penulis untuk menangani kueri yang diteruskan dari instans pembaca. Ini dinyatakan sebagai persentase dari pengaturan `max_connections` untuk penulis. Misalnya, jika `max_connections` adalah 800 dan `aurora_fwd_master_max_connections_pct` atau `aurora_fwd_writer_max_connections_pct` adalah 10, maka penulis mengizinkan maksimal 80 sesi terusan serentak. Koneksi ini berasal dari pool koneksi yang sama yang dikelola oleh pengaturan `max_connections`.

  Pengaturan ini hanya berlaku pada penulis saat penerusan tulis diaktifkan. Jika Anda menurunkan nilai, koneksi yang ada tidak akan terpengaruh. Aurora mempertimbangkan nilai baru pengaturan saat mencoba membuat koneksi baru dari klaster DB. Nilai defaultnya adalah 10, mewakili 10% dari nilai `max_connections`.

> **Note**
>
> Karena `aurora_fwd_writer_idle_timeout` dan `aurora_fwd_writer_max_connections_pct` adalah parameter klaster, semua instans DB di setiap klaster memiliki nilai yang sama untuk parameter ini.

Untuk informasi selengkapnya tentang `aurora_replica_read_consistency`, lihat Konsistensi baca untuk penerusan tulis.

Untuk informasi selengkapnya tentang grup parameter DB, lihat Bekerja dengan grup parameter.

Metrik Amazon CloudWatch dan variabel status Aurora MySQL untuk penerusan tulis

Metrik Amazon CloudWatch dan variabel status Aurora MySQL berikut ini berlaku saat Anda menggunakan penerusan tulis pada satu atau beberapa klaster DB. Semua metrik dan variabel status ini diukur pada instans DB penulis.

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
`ForwardingWriterDMLLatency`	–	Milidetik	Rata-rata waktu untuk memproses setiap pernyataan DML

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
			yang diteruskan pada instans DB penulis. Ini tidak termasuk waktu yang diperlukan klaster DB untuk meneruskan permintaan tulis, atau waktu untuk mereplikasi perubahan kembali ke penulis.
`ForwardingWriterDMLThroughput`	–	Hitungan per detik	Jumlah pernyataan DML yang diteruskan yang diproses setiap detik oleh instans DB penulis ini.
`ForwardingWriterOpenSessions`	`Aurora_fwd_writer_open_sessions`	Hitungan	Jumlah sesi yang diteruskan pada instans DB tulis.
–	`Aurora_fwd_writer_dml_stmt_count`	Hitungan	Total jumlah pernyataan DML yang diteruskan ke instans DB penulis ini.
–	`Aurora_fwd_writer_dml_stmt_duration`	Mikrodetik	Total durasi pernyataan DML yang diteruskan ke instans DB penulis ini.

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
–	`Aurora_fwd_writer_select_stmt_count`	Hitungan	Total jumlah pernyataan `SELECT` yang diteruskan ke instans DB penulis ini.
–	`Aurora_fwd_writer_select_stmt_duration`	Mikrodetik	Total durasi pernyataan `SELECT` yang diteruskan ke instans DB penulis ini.

Metrik CloudWatch dan variabel status Aurora MySQL berikut ini diukur pada setiap instans DB pembaca dalam klaster DB dengan penerusan tulis yang diaktifkan.

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
`ForwardingReplicaDMLLatency`	–	Milidetik	Rata-rata waktu respons DML yang diteruskan pada replika.
`ForwardingReplicaDMLThroughput`	–	Hitungan per detik	Jumlah pernyataan DML yang diteruskan yang diproses setiap detiknya.
`ForwardingReplicaOpenSessions`	`Aurora_fwd_replica_open_sessions`	Hitungan	Jumlah sesi yang menggunakan penerusan tulis pada instans DB pembaca.
`ForwardingReplicaReadWaitLatency`	–	Milidetik	Rata-rata waktu tunggu yang diperlukan sebuah pernyataa

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
			n SELECT di instans DB pembaca untuk menunggu agar dapat mengimbangi penulis. Batas tunggu instans DB pembaca sebelum memproses kueri ditentukan oleh pengatura n aurora_re plica_rea d_consistency .
ForwardingReplicaReadWaitThroughput	–	Hitungan per detik	Jumlah total pernyataan SELECT yang diproses setiap detik di semua sesi yang meneruskan tulis.
ForwardingReplicaSelectLatency	–	Milidetik	Latensi SELECT yang diteruskan, dirata-ratakan atas semua pernyataa n SELECT yang diteruskan dalam periode pemantauan.
ForwardingReplicaSelectThroughput	–	Hitungan per detik	Rata-rata throughpu t SELECT per detik yang diteruska n dalam periode pemantauan.

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
–	`Aurora_fwd_replica_dml_stmt_count`	Hitungan	Total jumlah pernyataan DML yang diteruskan ke instans DB pembaca ini.
–	`Aurora_fwd_replica_dml_stmt_duration`	Mikrodetik	Total durasi semua pernyataan DML yang diteruskan ke instans DB pembaca ini.
–	`Aurora_fwd_replica_errors_session_limit`	Hitungan	Jumlah sesi yang ditolak oleh klaster primer karena salah satu kondisi kesalahan berikut: • penulis penuh • Terlalu banyak pernyataan yang diteruskan dalam proses.
–	`Aurora_fwd_replica_read_wait_count`	Hitungan	Total jumlah tunggu baca-sesudah-tulis pada instans DB pembaca ini.
–	`Aurora_fwd_replica_read_wait_duration`	Mikrodetik	Total durasi tunggu karena pengaturan konsistensi baca pada instans DB pembaca ini.

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
–	`Aurora_fwd_replica_select_stmt_count`	Hitungan	Total jumlah pernyataan `SELECT` yang diteruskan ke instans DB pembaca ini.
–	`Aurora_fwd_replica_select_stmt_duration`	Mikrodetik	Total durasi pernyataan `SELECT` yang diteruskan ke instans DB pembaca ini.

Mengidentifikasi transaksi dan kueri yang diteruskan

Anda dapat menggunakan tabel `information_schema.aurora_forwarding_processlist` untuk mengidentifikasi transaksi dan kueri yang diteruskan. Untuk informasi selengkapnya tentang tabel ini, lihat information_schema.aurora_forwarding_processlist.

Contoh berikut ini menunjukkan semua koneksi yang diteruskan pada instans DB tulis.

```
mysql> select * from information_schema.AURORA_FORWARDING_PROCESSLIST where
 IS_FORWARDED=1 order by REPLICA_SESSION_ID;

+-----+----------+-------------------+----------+---------+------+--------------
+-------------------------------------------+--------------+-------------------
+--------------------------------+----------------------+----------------+
| ID  | USER     | HOST              | DB       | COMMAND | TIME | STATE        |
 INFO                                     | IS_FORWARDED | REPLICA_SESSION_ID |
 REPLICA_INSTANCE_IDENTIFIER     | REPLICA_CLUSTER_NAME | REPLICA_REGION |
+-----+----------+-------------------+----------+---------+------+--------------
+-------------------------------------------+--------------+-------------------
+--------------------------------+----------------------------------------+
| 648 | myuser   | IP_address:port1   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest58 SET k=k+1 WHERE id=4802579 |            1 |                637 | my-
db-cluster-instance-2        | my-db-cluster        | us-west-2      |
| 650 | myuser   | IP_address:port2   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest54 SET k=k+1 WHERE id=2503953 |            1 |                639 | my-
db-cluster-instance-2        | my-db-cluster        | us-west-2      |
```

```
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+--------------+-------------------
+---------------------------------+---------------------+----------------+
```

Pada instans DB pembaca penerusan, Anda dapat melihat thread yang terkait dengan koneksi DB penulis ini dengan menjalankan `SHOW PROCESSLIST`. Nilai `REPLICA_SESSION_ID` pada penulis, 637 dan 639, sama dengan nilai `Id` pada pembaca.

```
mysql> select @@aurora_server_id;

+---------------------------------+
| @@aurora_server_id              |
+---------------------------------+
| my-db-cluster-instance-2        |
+---------------------------------+
1 row in set (0.00 sec)

mysql> show processlist;

+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+
| Id  | User     | Host               | db       | Command | Time | State        | Info
                                       |
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+
| 637 | myuser   | IP_address:port1   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest12 SET k=k+1 WHERE id=4802579  |
| 639 | myuser   | IP_address:port2   | sysbench | Query   |    0 | async commit |
 UPDATE sbtest61 SET k=k+1 WHERE id=2503953  |
+-----+----------+--------------------+----------+---------+------+--------------
+-------------------------------------------+
12 rows in set (0.00 sec)
```

Mereplikasi klaster DB Amazon Aurora MySQL di seluruh Wilayah AWS

Anda dapat membuat klaster DB Amazon Aurora MySQL sebagai replika baca di Wilayah AWS yang berbeda dari klaster DB sumber. Pendekatan ini dapat meningkatkan kemampuan pemulihan bencana Anda, memungkinkan Anda menskalakan operasi baca ke Wilayah AWS yang lebih dekat dengan pengguna Anda, dan mempermudah migrasi antar-Wilayah AWS.

Anda dapat membuat replika baca klaster DB terenkripsi atau tidak terenkripsi. Replika baca harus dienkripsi jika klaster DB sumber dienkripsi.

Untuk setiap klaster DB sumber, Anda dapat memiliki hingga lima klaster DB lintas Wilayah yang berupa replika baca.

> **Note**
>
> Sebagai alternatif untuk replika baca lintas Wilayah, Anda dapat menskalakan operasi baca dengan waktu lag minimal dengan menggunakan basis data global Aurora. Basis data global Aurora memiliki klaster DB Aurora primer dalam satu Wilayah AWS dan hingga lima klaster DB hanya-baca sekunder di Wilayah yang berbeda-beda. Setiap klaster DB sekunder dapat menyertakan hingga 16 (bukan 15) Replika Aurora. Replikasi dari klaster DB primer ke semua sekunder ditangani oleh lapisan penyimpanan Aurora, bukan oleh mesin basis data, sehingga waktu lag untuk mereplikasi perubahan menjadi minimal—biasanya, kurang dari 1 detik. Mengecualikan mesin basis data dari proses replikasi berarti bahwa mesin basis data dikhususkan untuk memproses beban kerja. Ini juga berarti bahwa Anda tidak perlu mengonfigurasi atau mengelola replikasi binlog (binlog biner) Aurora MySQL. Untuk mempelajari selengkapnya, lihat Menggunakan basis data global Amazon Aurora.

Saat Anda membuat replika baca klaster DB Aurora MySQL di Wilayah AWS lain, Anda harus mengetahui hal berikut:

- Klaster DB sumber Anda dan klaster DB replika baca lintas Wilayah Anda dapat memiliki hingga 15 Replika Aurora, di samping instans primer untuk klaster DB tersebut. Dengan menggunakan fungsi ini, Anda dapat menskalakan operasi baca untuk Wilayah AWS sumber dan Wilayah AWS target replikasi Anda.
- Dalam sebuah skenario lintas Wilayah, terdapat lebih banyak waktu lag antara klaster DB sumber dan replika baca karena saluran jaringan yang lebih panjang antar-Wilayah AWS.

- Data yang ditransfer untuk replikasi lintas Wilayah akan menimbulkan biaya transfer data Amazon RDS. Tindakan replikasi lintas Wilayah berikut menghasilkan biaya untuk data yang ditransfer dari Wilayah AWS sumber:
  - Saat Anda membuat replika baca, Amazon RDS mengambil snapshot dari klaster sumber dan mentransfer snapshot tersebut ke Wilayah AWS yang menyimpan replika baca.
  - Untuk setiap modifikasi data yang dilakukan dalam basis data sumber, Amazon RDS mentransfer data dari wilayah sumber ke Wilayah AWS yang menyimpan replika baca.

  Untuk informasi selengkapnya tentang harga transfer data Amazon RDS, lihat Harga Amazon Aurora.
- Anda dapat menjalankan beberapa tindakan pembuatan atau penghapusan konkuren untuk replika baca yang mengacu pada klaster DB sumber yang sama. Namun, Anda harus tetap berada dalam batasan lima replika baca untuk setiap klaster DB sumber.
- Agar replikasi dapat beroperasi secara efektif, setiap replika baca harus memiliki jumlah sumber daya komputasi dan penyimpanan yang sama dengan klaster DB sumber. Jika Anda menskalakan klaster DB sumber, Anda juga harus menskalakan replika baca.

Topik

- Sebelum Anda memulai
- Membuat sebuah klaster DB Amazon Aurora MySQL yang berupa replika baca lintas Wilayah
- Melihat replika lintas Wilayah Amazon Aurora MySQL
- Mempromosikan replika baca menjadi klaster DB
- Pemecahan masalah replika lintas Wilayah Amazon Aurora MySQL

Sebelum Anda memulai

Sebelum Anda dapat membuat klaster DB Aurora MySQL yang merupakan replika baca lintas Wilayah, Anda harus mengaktifkan logging biner pada klaster DB Aurora MySQL sumber Anda. Replikasi lintas wilayah untuk Aurora MySQL menggunakan replikasi biner MySQL untuk mengulang perubahan pada klaster DB replika baca lintas Wilayah.

Untuk mengaktifkan logging biner pada sebuah klaster DB Aurora MySQL, perbarui parameter `binlog_format` untuk klaster DB sumber Anda. Parameter `binlog_format` adalah parameter tingkat klaster yang berada dalam grup parameter klaster default. Jika klaster DB Anda menggunakan grup parameter klaster DB default, buat sebuah grup parameter klaster DB baru untuk

memodifikasi pengaturan `binlog_format`. Kami sarankan Anda mengatur `binlog_format` ke `MIXED`. Namun, Anda juga dapat mengatur `binlog_format` ke `ROW` atau `STATEMENT` jika Anda memerlukan format binlog tertentu. Boot ulang klaster DB Aurora Anda agar perubahan dapat diterapkan.

Untuk informasi selengkapnya tentang menggunakan logging biner dengan Aurora MySQL, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner). Untuk informasi selengkapnya tentang memodifikasi parameter konfigurasi Aurora MySQL, lihat Parameter instans DB dan klaster DB Amazon Aurora dan Bekerja dengan grup parameter.

Membuat sebuah klaster DB Amazon Aurora MySQL yang berupa replika baca lintas Wilayah

Anda dapat membuat sebuah klaster DB Aurora yang berupa replika baca lintas Wilayah dengan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau API Amazon RDS. Anda dapat membuat replika baca klaster lintas Wilayah dari klaster DB yang terenkripsi atau tidak terenkripsi.

Saat Anda membuat replika baca lintas Wilayah untuk Aurora MySQL dengan menggunakan AWS Management Console, Amazon RDS akan membuat sebuah klaster DB dalam Wilayah AWS target, lalu akan secara otomatis membuat sebuah instans DB yang merupakan instans primer untuk klaster DB tersebut.

Saat Anda membuat replika baca lintas Wilayah menggunakan AWS CLI atau API RDS, pertama-tama Anda harus membuat klaster DB di Wilayah AWS target dan menunggu hingga wilayah ini menjadi aktif. Setelah aktif, Anda kemudian dapat membuat sebuah instans DB yang merupakan instans primer untuk klaster DB tersebut.

Replikasi dimulai saat instans primer dari klaster DB replika baca menjadi tersedia.

Gunakan prosedur berikut untuk membuat sebuah replika baca lintas Wilayah dari sebuah klaster DB Aurora MySQL. Prosedur ini berfungsi untuk pembuatan replika baca dari klaster DB terenkripsi atau tidak terenkripsi.

Konsol

Untuk membuat klaster DB Aurora MySQL yang merupakan replika baca lintas Wilayah dengan AWS Management Console

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

2. Di sudut kanan atas AWS Management Console, pilih Wilayah AWS yang meng-host klaster DB sumber Anda.
3. Di panel navigasi, pilih Basis data.
4. Pilih klaster DB yang ingin Anda buat replika baca lintas Wilayahnya.
5. Untuk Tindakan, pilih Buat replika baca lintas Wilayah.
6. Pada halaman Buat replika baca lintas wilayah, pilih pengaturan opsi untuk klaster DB replika baca lintas Wilayah Anda, seperti yang dijelaskan dalam tabel berikut.

Opsi	Deskripsi
Wilayah tujuan	Pilih Wilayah AWS untuk meng-host klaster DB replika baca lintas Wilayah yang baru.
Grup subnet DB tujuan	Pilih grup subnet DB yang akan digunakan untuk klaster DB replika baca lintas Wilayah.
Dapat diakses publik	Pilih Ya untuk memberikan alamat IP publik ke klaster DB replika baca lintas Wilayah; jika tidak, pilih Tidak.
Enkripsi	Pilih Aktifkan Enkripsi untuk mengaktifkan enkripsi saat diam untuk klaster DB ini. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.
AWS KMS key	Hanya tersedia jika Enkripsi diatur ke Aktifkan Enkripsi. Pilih AWS KMS key yang akan digunakan untuk mengenkripsi klaster DB ini. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.
Kelas instans DB	Pilih kelas instans DB yang menentukan persyaratan pemrosesan dan memori untuk instans primer dalam DB klaster. Untuk informasi selengkapnya tentang opsi kelas instans DB, lihat Kelas instans DB Aurora.
Deployment Multi-AZ	Pilih Ya untuk membuat sebuah replika baca dari klaster DB baru di Zona Ketersediaan lainnya di

Opsi	Deskripsi
	Wilayah AWS target untuk dukungan failover. Untuk informasi selengkapnya tentang beberapa Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan.
Sumber replika baca	Pilih klaster DB sumber yang akan dibuatkan replika baca lintas Wilayahnya.
Pengidentifikasi instans DB	Ketikkan sebuah nama untuk instans primer dalam klaster DB replika baca lintas Wilayah Anda. Pengidentifikasi ini digunakan dalam alamat titik akhir untuk instans primer klaster DB baru. Pengidentifikasi instans DB memiliki batasan berikut: • Pengidentifikasi ini harus berisi 1 hingga 63 karakter alfanumerik atau tanda hubung. • Karakter pertamanya harus berupa huruf. • Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut. • Pengidentifikasi ini harus unik untuk semua instans DB untuk setiap Akun AWS dan setiap Wilayah AWS. Karena klaster DB replika baca lintas Wilayah dibuat dari sebuah snapshot klaster DB sumber, nama pengguna master dan kata sandi master untuk replika baca harus sama dengan nama pengguna master dan kata sandi master untuk klaster DB sumber.

Opsi	Deskripsi
Pengidentifikasi klaster DB	Ketikkan sebuah nama untuk klaster DB replika baca lintas Wilayah yang unik untuk akun Anda dalam Wilayah AWS target untuk replika Anda. Pengident ifikasi ini digunakan dalam alamat titik akhir klaster untuk klaster DB Anda. Untuk informasi tentang titik akhir klaster, lihat Manajemen koneksi Amazon Aurora. Pengidentifikasi klaster DB memiliki batasan berikut: • Pengidentifikasi ini harus berisi 1 hingga 63 karakter alfanumerik atau tanda hubung. • Karakter pertamanya harus berupa huruf. • Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut. • Pengidentifikasi ini harus unik untuk semua klaster DB untuk setiap Akun AWS dan setiap Wilayah AWS.
Prioritas	Pilih sebuah prioritas failover untuk instans primer dari klaster DB baru. Prioritas ini akan menentukan urutan promosi Aurora Replika saat melakukan pemulihan dari kegagalan instans primer. Jika Anda tidak memilih nilai, nilai default-nya adalah tier-1. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora.
Port basis data	Tentukan port untuk aplikasi dan utilitas yang akan digunakan untuk mengakses basis data. Klaster DB Aurora ditetapkan secara default ke port MySQL default, 3306. Firewall di beberapa perusahaan memblokir koneksi ke port ini. Jika firewall perusahaa n Anda memblokir port default ini, pilih port lain untuk klaster DB baru.

Opsi	Deskripsi
Pemantauan yang ditingkatkan	Pilih Aktifkan pemantauan yang ditingkatkan untuk mengaktifkan pengumpulan metrik secara waktu nyata untuk sistem operasi tempat klaster DB Anda berjalan. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan.
Peran Pemantauan	Hanya tersedia jika Pemantauan yang Ditingkatkan diatur ke Aktifkan pemantauan yang ditingkatkan. Pilih peran IAM yang Anda buat untuk mengizinkan Amazon RDS berkomunikasi dengan CloudWatch Log Amazon untuk Anda, atau pilih Default agar RDS membuat peran untuk nama Anda. `rds-monitoring-role` Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan.
Granularitas	Hanya tersedia jika Pemantauan yang Ditingkatkan diatur ke Aktifkan pemantauan yang ditingkatkan. Atur interval, dalam detik, di antara waktu pengumpulan metrik untuk klaster DB Anda.
Peningkatan versi minor otomatis	Pengaturan ini tidak berlaku untuk klaster DB Aurora MySQL. Untuk informasi selengkapnya tentang pembaruan mesin untuk Aurora MySQL, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL.

7. Pilih Buat untuk membuat replika baca lintas Wilayah Anda untuk Aurora.

AWS CLI

Untuk membuat sebuah klaster DB Aurora MySQL yang berupa replika baca lintas Wilayah dengan CLI

1. Panggil AWS CLI create-db-clusterperintah di Wilayah AWS tempat Anda ingin membuat cluster DB replika baca. Sertakan opsi `--replication-source-identifier` dan tentukan Amazon Resource Name (ARN) milik klaster DB sumber untuk membuat sebuah replika baca.

   Untuk replikasi lintas Wilayah yang mengenkripsi klaster DB yang diidentifikasi berdasarkan `--replication-source-identifier`, tentukan opsi `--kms-key-id` dan opsi `--storage-encrypted`.

   > **Note**
   >
   > Anda dapat mengatur replikasi lintas Wilayah dari sebuah klaster DB yang tidak terenkripsi ke replika baca terenkripsi dengan menentukan `--storage-encrypted` dan menyediakan sebuah nilai untuk `--kms-key-id`.

   Anda tidak dapat menentukan parameter `--master-username` dan `--master-user-password`. Nilai tersebut diambil dari klaster DB sumber.

   Contoh kode berikut ini membuat sebuah replika baca di Wilayah us-east-1 dari sebuah snapshot klaster DB tidak terenkripsi di Wilayah us-west-2. Perintah ini dipanggil di Wilayah us-east-1. Contoh ini menentukan opsi `--manage-master-user-password` untuk menghasilkan kata sandi pengguna master dan mengelolanya di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. Alternatifnya, Anda dapat menggunakan opsi `--master-password` untuk menentukan dan mengelola kata sandi sendiri.

   Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
  --db-cluster-identifier sample-replica-cluster \
  --engine aurora \
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster
```

Untuk Windows:

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster
```

Contoh kode berikut ini membuat sebuah replika baca di Wilayah us-east-1 dari sebuah snapshot klaster DB terenkripsi di Wilayah us-west-2. Perintah ini dipanggil di Wilayah us-east-1.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
  --db-cluster-identifier sample-replica-cluster \
  --engine aurora \
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster \
  --kms-key-id my-us-east-1-key \
  --storage-encrypted
```

Untuk Windows:

```
aws rds create-db-cluster ^
  --db-cluster-identifier sample-replica-cluster ^
  --engine aurora ^
  --replication-source-identifier arn:aws:rds:us-
west-2:123456789012:cluster:sample-master-cluster ^
  --kms-key-id my-us-east-1-key ^
  --storage-encrypted
```

`--source-region`Opsi ini diperlukan untuk replikasi lintas wilayah antara Wilayah AWS GovCloud (AS-Timur) dan AWS GovCloud (AS-Barat), di mana cluster DB yang diidentifikasi oleh dienkripsi. `--replication-source-identifier` Untuk `--source-region`, tentukan Wilayah AWS klaster DB sumber.

Jika `--source-region` tidak ditentukan, tentukan nilai `--pre-signed-url`. URL yang telah ditandatangani adalah URL yang berisi permintaan bertanda tangan Signature Versi 4 untuk perintah `create-db-cluster` yang dipanggil di Wilayah AWS sumber. Untuk mempelajari

lebih lanjut tentang `pre-signed-url` opsi, lihat create-db-clusterdi Referensi AWS CLI Perintah.

2. Periksa apakah klaster DB sudah tersedia untuk digunakan dengan menggunakan perintah AWS CLI describe-db-clusters, seperti yang ditunjukkan dalam contoh berikut.

```
aws rds describe-db-clusters --db-cluster-identifier sample-replica-cluster
```

Saat hasil **`describe-db-clusters`** menunjukkan sebuah status dari `available`, buat instans primer untuk klaster DB sehingga replikasi dapat dimulai. Untuk melakukannya, gunakan perintah AWS CLI create-db-instance seperti pada contoh berikut.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier sample-replica-cluster \
  --db-instance-class db.r3.large \
  --db-instance-identifier sample-replica-instance \
  --engine aurora
```

Untuk Windows:

```
aws rds create-db-instance ^
  --db-cluster-identifier sample-replica-cluster ^
  --db-instance-class db.r3.large ^
  --db-instance-identifier sample-replica-instance ^
  --engine aurora
```

Ketika instans DB dibuat dan tersedia, replikasi akan dimulai. Anda dapat menentukan apakah instans DB tersedia dengan menghubungi perintah AWS CLI describe-db-instances.

API RDS

Untuk membuat sebuah klaster DB Aurora MySQL yang berupa replika baca lintas Wilayah dengan API

1. Panggil operasi API RDS CreateDBCluster di Wilayah AWS tempat Anda ingin membuat klaster DB replika baca. Sertakan parameter `ReplicationSourceIdentifier` dan tentukan Amazon Resource Name (ARN) milik klaster DB sumber untuk membuat sebuah replika baca.

Untuk replikasi lintas Wilayah yang mengenkripsi klaster DB yang diidentifikasi berdasarkan `ReplicationSourceIdentifier`, tentukan parameter `KmsKeyId` dan atur parameter `StorageEncrypted` ke `true`.

> **Note**
>
> Anda dapat mengatur replikasi lintas Wilayah dari sebuah klaster DB yang tidak terenkripsi ke replika baca terenkripsi dengan menentukan `StorageEncrypted` sebagai **`true`** dan menyediakan sebuah nilai untuk `KmsKeyId`. Dalam hal ini, Anda tidak perlu menentukan `PreSignedUrl`.

Anda tidak perlu menyertakan parameter `MasterUsername` dan `MasterUserPassword` karena nilai-nilai tersebut diambil dari klaster DB sumber.

Contoh kode berikut ini membuat sebuah replika baca di Wilayah us-east-1 dari sebuah snapshot klaster DB tidak terenkripsi di Wilayah us-west-2. Tindakan ini dipanggil di Wilayah us-east-1.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBCluster
  &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
  &DBClusterIdentifier=sample-replica-cluster
  &Engine=aurora
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T001547Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

Contoh kode berikut ini membuat sebuah replika baca di Wilayah us-east-1 dari sebuah snapshot klaster DB terenkripsi di Wilayah us-west-2. Tindakan ini dipanggil di Wilayah us-east-1.

```
https://rds.us-east-1.amazonaws.com/
```

```
  ?Action=CreateDBCluster
  &KmsKeyId=my-us-east-1-key
  &StorageEncrypted=true
  &PreSignedUrl=https%253A%252F%252Frds.us-west-2.amazonaws.com%252F
         %253FAction%253DCreateDBCluster
         %2526DestinationRegion%253Dus-east-1
         %2526KmsKeyId%253Dmy-us-east-1-key
         %2526ReplicationSourceIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-
west-2%25253A123456789012%25253Acluster%25253Asample-master-cluster
         %2526SignatureMethod%253DHmacSHA256
         %2526SignatureVersion%253D4
         %2526Version%253D2014-10-31
         %2526X-Amz-Algorithm%253DAWS4-HMAC-SHA256
         %2526X-Amz-Credential%253DAKIADQKE4SARGYLE%252F20161117%252Fus-
west-2%252Frds%252Faws4_request
         %2526X-Amz-Date%253D20161117T215409Z
         %2526X-Amz-Expires%253D3600
         %2526X-Amz-SignedHeaders%253Dcontent-type%253Bhost%253Buser-agent%253Bx-
amz-content-sha256%253Bx-amz-date
         %2526X-Amz-Signature
%253D255a0f17b4e717d3b67fad163c3ec26573b882c03a65523522cf890a67fca613
  &ReplicationSourceIdentifier=arn:aws:rds:us-west-2:123456789012:cluster:sample-
master-cluster
  &DBClusterIdentifier=sample-replica-cluster
  &Engine=aurora
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T001547Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=a04c831a0b54b5e4cd236a90dcb9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

Untuk replikasi lintas wilayah antara Wilayah AWS GovCloud (AS-Timur) dan AWS GovCloud (AS-Barat), di mana cluster DB yang diidentifikasi oleh `ReplicationSourceIdentifier` dienkripsi, juga tentukan parameternya. `PreSignedUrl` URL yang telah ditandatangani harus berupa sebuah permintaan yang valid untuk operasi API `CreateDBCluster` yang dapat dilakukan di Wilayah AWS sumber yang berisi klaster DB terenkripsi yang akan direplikasi. Pengidentifikasi kunci KMS digunakan untuk mengenkripsi replika baca, dan harus berupa kunci KMS yang valid untuk Wilayah AWS tujuan. Untuk membuat URL yang sudah ditandatangani

secara otomatis ketimbang secara manual, gunakan perintah AWS CLI create-db-cluster dengan opsi `--source-region`.

2. Periksa apakah klaster DB sudah tersedia untuk digunakan dengan menggunakan operasi API RDS DescribeDBClusters, seperti yang ditunjukkan pada contoh berikut.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=DescribeDBClusters
  &DBClusterIdentifier=sample-replica-cluster
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T002223Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=84c2e4f8fba7c577ac5d820711e34c6e45ffcd35be8a6b7c50f329a74f35f426
```

Saat hasil `DescribeDBClusters` menunjukkan status dari `available`, buat instans primer untuk klaster DB sehingga replikasi dapat dimulai. Untuk melakukannya, gunakan tindakan API RDS CreateDBInstance seperti pada contoh berikut.

```
https://rds.us-east-1.amazonaws.com/
  ?Action=CreateDBInstance
  &DBClusterIdentifier=sample-replica-cluster
  &DBInstanceClass=db.r3.large
  &DBInstanceIdentifier=sample-replica-instance
  &Engine=aurora
  &SignatureMethod=HmacSHA256
  &SignatureVersion=4
  &Version=2014-10-31
  &X-Amz-Algorithm=AWS4-HMAC-SHA256
  &X-Amz-Credential=AKIADQKE4SARGYLE/20161117/us-east-1/rds/aws4_request
  &X-Amz-Date=20160201T003808Z
  &X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
  &X-Amz-Signature=125fe575959f5bbcebd53f2365f907179757a08b5d7a16a378dfa59387f58cdb
```

Ketika instans DB dibuat dan tersedia, replikasi akan dimulai. Anda dapat menentukan apakah instans DB tersedia dengan memanggil perintah AWS CLI DescribeDBInstances.

Melihat replika lintas Wilayah Amazon Aurora MySQL

Anda dapat melihat hubungan replikasi lintas wilayah untuk cluster DB Amazon Aurora MySQL Anda dengan memanggil describe-db-clustersAWS CLIperintah atau operasi DescribedBClusters RDS API. Dalam respons, lihat bidang `ReadReplicaIdentifiers` untuk menemukan pengidentifikasi klaster DB milik setiap klaster DB replika baca lintas Wilayah. Lihat elemen `ReplicationSourceIdentifier` untuk menemukan ARN milik klaster DB sumber yang merupakan sumber replikasi.

Mempromosikan replika baca menjadi klaster DB

Anda dapat mempromosikan replika baca Aurora MySQL menjadi klaster DB mandiri. Saat Anda mempromosikan replika baca Aurora MySQL, instans DB akan di-boot ulang sebelum instans tersebut tersedia.

Biasanya, Anda mempromosikan replika baca Aurora MySQL menjadi klaster DB mandiri sebagai skema pemulihan data jika klaster DB sumber mengalami kegagalan.

Untuk melakukannya, pertama-tama buat replika baca lalu pantau klaster DB sumber untuk mengetahui kegagalan. Jika terjadi kegagalan, lakukan hal berikut:

1. Promosikan replika baca.
2. Arahkan lalu lintas basis data ke klaster DB yang dipromosikan.
3. Buat replika baca pengganti dengan klaster DB yang dipromosikan sebagai sumbernya.

Saat Anda mempromosikan replika baca, replika baca tersebut menjadi klaster DB Aurora mandiri. Proses promosi dapat memakan waktu beberapa menit atau lebih lama, tergantung dari ukuran replika baca. Setelah Anda mempromosikan replika baca menjadi klaster DB baru, replika tersebut akan sama seperti klaster DB lainnya. Misalnya, Anda dapat membuat replika baca darinya dan melakukan operasi point-in-time pemulihan. Anda juga dapat membuat Replika Aurora untuk klaster DB.

Karena klaster DB yang dipromosikan bukan lagi merupakan replika baca, Anda tidak dapat menggunakannya sebagai target replikasi.

Langkah-langkah berikut menunjukkan proses umum untuk mempromosikan replika baca menjadi klaster DB:

1. Hentikan penulisan transaksi apa pun ke klaster DB sumber replika baca, lalu tunggu semua pembaruan yang akan dilakukan ke replika baca. Pembaruan basis data terjadi pada replika baca setelah pembaruan terjadi pada klaster DB sumber, dan lag replikasi ini dapat bervariasi secara signifikan. Gunakan metrik `ReplicaLag` untuk menentukan saat semua pembaruan sudah dilakukan pada replika baca. Metrik `ReplicaLag` mencatat jumlah waktu lag instans DB replika di belakang instans DB sumber. Saat metrik `ReplicaLag` mencapai `0`, replika baca telah menyamai instans DB sumber.
2. Promosikan replika baca dengan menggunakan opsi Promosikan di konsol Amazon RDS, AWS CLI perintah promote-read-replica-db-cluster, atau operasi DBCluster PromoteReadReplicaAmazon RDS API.

   Anda dapat memilih instans DB Aurora MySQL untuk mempromosikan replika baca. Setelah replika baca dipromosikan, klaster DB Aurora MySQL akan dipromosikan menjadi klaster DB mandiri. Instans DB dengan prioritas failover tertinggi akan dipromosikan menjadi instans DB primer untuk klaster DB. Instans DB lainnya menjadi Replika Aurora.

   > **Note**
   >
   > Proses promosi memakan waktu beberapa menit. Saat Anda mempromosikan replika baca, replikasi dihentikan dan instans DB di-boot ulang. Saat boot ulang selesai, replika baca tersedia sebagai klaster DB baru.

Konsol

Untuk mempromosikan replika baca Aurora MySQL menjadi klaster DB

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pada konsol, pilih Instans.

   Panel Instans akan muncul.
3. Dalam panel Instans, pilih replika baca yang ingin Anda promosikan.

   Replika baca akan muncul sebagai instans DB Aurora MySQL.
4. Untuk Tindakan, pilih Promosikan replika baca.
5. Pada halaman konfirmasi, pilih Promosikan replika baca.

AWS CLI

Untuk mempromosikan replika baca ke cluster DB, gunakan perintah AWS CLI promote-read-replica-db-cluster.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier mydbcluster
```

Untuk Windows:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier mydbcluster
```

API RDS

Untuk mempromosikan replika baca ke cluster DB, panggil PromoteReadReplicaDBCluster.

Pemecahan masalah replika lintas Wilayah Amazon Aurora MySQL

Di bagian berikut ini, Anda dapat menemukan daftar pesan kesalahan umum yang mungkin Anda temukan saat membuat replika baca lintas Wilayah Amazon Aurora, dan cara menyelesaikan kesalahan yang ditentukan.

Klaster sumber [ARN klaster DB] tidak memiliki binlog aktif

Untuk mengatasi masalah ini, aktifkan logging biner pada klaster DB sumber. Untuk informasi selengkapnya, lihat Sebelum Anda memulai.

Klaster sumber [ARN klaster DB] tidak memiliki grup parameter klaster yang sinkron pada penulis

Anda akan menerima kesalahan ini jika Anda telah memperbarui parameter klaster DB `binlog_format`, tetapi belum mem-boot ulang instans primer untuk klaster DB. Boot ulang instans primer (yaitu penulis) untuk klaster DB lalu coba lagi.

Klaster sumber [ARN klaster DB] sudah memiliki replika baca di wilayah ini

Anda dapat memiliki hingga lima klaster DB lintas Wilayah yang berupa replika baca untuk setiap klaster DB sumber di Wilayah AWS mana pun. Jika Anda sudah memiliki jumlah maksimum replika

baca pada klaster DB di Wilayah AWS tertentu, Anda harus menghapus yang sudah ada sebelum Anda dapat membuat klaster DB lintas Wilayah baru di Wilayah tersebut.

Klaster DB [ARN klaster DB] memerlukan upgrade mesin basis data untuk dukungan replikasi lintas Wilayah

Untuk mengatasi masalah ini, upgrade versi mesin basis data pada semua instans dalam klaster DB sumber ke versi mesin basis data terbaru, lalu coba buat lagi DB replika baca lintas Wilayah.

Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner)

Karena Amazon Aurora MySQL kompatibel dengan MySQL, Anda dapat mengatur replikasi antara basis data MySQL dan klaster DB Amazon Aurora MySQL. Jenis replikasi ini menggunakan replikasi log biner MySQL, dan secara umum disebut sebagai replikasi binlog. Jika Anda menggunakan replikasi log biner dengan Aurora, sebaiknya basis data MySQL Anda menjalankan MySQL versi 5.5 atau yang lebih baru. Anda dapat mengatur replikasi jika klaster DB Aurora MySQL Anda merupakan sumber replikasi atau replika. Anda dapat mereplikasi dengan instans DB Amazon RDS MySQL, basis data MySQL eksternal di luar Amazon RDS, atau klaster DB Aurora MySQL lainnya.

> **Note**
>
> Anda tidak dapat menggunakan replikasi binlog ke atau dari jenis klaster Aurora DB tertentu. Secara khusus, replikasi binlog tidak tersedia untuk klaster Aurora Serverless v1. Jika pernyataan `SHOW MASTER STATUS` dan `SHOW SLAVE STATUS` (Aurora MySQL versi 2) atau (`SHOW REPLICA STATUS`Aurora MySQL versi 3) tidak menghasilkan output, periksa apakah klaster yang Anda gunakan mendukung replikasi binlog.
> Di Aurora MySQL versi 3, replikasi log biner tidak mereplikasi ke basis data sistem `mysql`. Kata sandi dan akun tidak direplikasi oleh replikasi binlog di Aurora MySQL versi 3. Oleh karena itu, pernyataan Bahasa Kontrol Data (DCL) seperti `CREATE USER`, `GRANT`, dan `REVOKE` tidak direplikasi.

Anda juga dapat mereplikasi dengan instans DB RDS for MySQL atau klaster DB Aurora MySQL di Wilayah AWS lainnya. Saat Anda melakukan replikasi Wilayah AWS, pastikan cluster DB dan instans DB Anda dapat diakses publik. Jika klaster DB MySQL Aurora berada di subnet privat di VPC Anda, gunakan peering VPC antar- Wilayah AWS. Untuk informasi selengkapnya, lihat klaster DB di VPC yang diakses oleh instans EC2 di VPC yang berbeda.

Jika Anda ingin mengonfigurasi replikasi antara cluster DB MySQL Aurora dan cluster DB MySQL Aurora Wilayah AWS di cluster lain, Anda dapat membuat cluster DB MySQL Aurora sebagai replika baca yang berbeda dari cluster DB sumber. Wilayah AWS Untuk informasi selengkapnya, lihat Mereplikasi klaster DB Amazon Aurora MySQL di seluruh Wilayah AWS.

Dengan Aurora MySQL versi 2 dan 3, Anda dapat mereplikasi antara Aurora MySQL dan sumber atau target eksternal yang menggunakan pengidentifikasi transaksi global (GTID) untuk replikasi. Pastikan parameter terkait GTID dalam klaster DB Aurora MySQL tersebut memiliki pengaturan yang kompatibel dengan status GTID basis data eksternal. Untuk mempelajari cara melakukannya, lihat Menggunakan replikasi berbasis GTID untuk Amazon Aurora MySQL. Di Aurora MySQL versi 3.01 dan yang lebih tinggi, Anda dapat memilih cara menetapkan GTID ke transaksi yang direplikasi dari sumber yang tidak menggunakan GTID. Untuk informasi tentang prosedur tersimpan yang mengontrol pengaturan tersebut, lihat mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL versi 3).

> **Warning**
>
> Saat Anda mereplikasi antara Aurora MySQL dan MySQL, pastikan Anda hanya menggunakan tabel InnoDB. Jika Anda memiliki tabel MyISAM yang ingin Anda replikasi, Anda dapat mengonversinya menjadi InnoDB sebelum mengatur replikasi dengan perintah berikut.
>
> ```
> alter table <schema>.<table_name> engine=innodb, algorithm=copy;
> ```

Pengaturan replikasi dengan MySQL atau klaster DB Aurora lainnya

Pengaturan replikasi MySQL dengan Aurora MySQL memerlukan langkah-langkah berikut, yang dibahas secara terperinci:

1. Aktifkan pencatatan log biner pada sumber replikasi
2. Pertahankan log biner pada sumber replikasi hingga tidak diperlukan lagi
3. Buat snapshot atau dump sumber replikasi Anda
4. Muat snapshot atau dump ke target replika Anda
5. Buat pengguna replikasi pada sumber replikasi Anda

6. Aktifkan replikasi pada target replika Anda

7. Pantau replika Anda

1. Aktifkan pencatatan log biner pada sumber replikasi

Temukan petunjuk tentang cara mengaktifkan pencatatan log biner pada sumber replikasi untuk mesin basis data Anda.

Mesin basis data	Petunjuk
Aurora MySQL	Untuk mengaktifkan pencatatan log biner pada klaster DB Aurora MySQL Atur parameter klaster DB `binlog_format` ke `ROW`, `STATEMENT`, atau `MIXED`. `MIXED` direkomendasikan kecuali jika Anda membutuhkan format binlog tertentu. (Nilai default-nya adalah `OFF`.) Untuk mengubah parameter `binlog_format`, buat grup parameter klaster DB kustom dan kaitkan grup parameter kustom tersebut dengan klaster DB Anda. Anda tidak dapat mengubah parameter dalam grup parameter klaster DB default. Jika Anda mengubah parameter `binlog_format` dari `OFF` ke nilai lainnya, boot ulang klaster DB Aurora Anda agar perubahan dapat diterapkan. Untuk informasi lebih lanjut, lihat Parameter instans DB dan klaster DB Amazon Aurora dan Bekerja dengan grup parameter.
RDS for MySQL	Untuk mengaktifkan pencatatan log biner pada instans DB Amazon RDS Anda tidak dapat mengaktifkan pencatatan log biner secara langsung pada instans DB Amazon RDS, tetapi Anda dapat mengaktifkannya dengan melakukan salah satu hal berikut ini: • Aktifkan cadangan otomatis untuk instans DB. Anda dapat mengaktifkan cadangan otomatis saat Anda membuat instans DB, atau Anda dapat mengaktifkan cadangan dengan memodifikasi instans DB yang sudah ada. Untuk informasi selengkapnya, lihat Membuat instans DB dalam Panduan Pengguna Amazon RDS.

Mesin basis data	Petunjuk
	• Buat replika baca untuk instans DB. Untuk informasi selengkapnya, lihat Menggunakan replika baca dalam Panduan Pengguna Amazon RDS.

Mesin basis data	Petunjuk
MySQL (eksternal)	Untuk mengatur replikasi terenkripsi Untuk mereplikasi data secara aman dengan Aurora MySQL versi 2, Anda dapat menggunakan replikasi terenkripsi. **Note** Jika Anda tidak perlu menggunakan replikasi terenkripsi, Anda dapat melewati langkah-langkah ini. Berikut ini adalah prasyarat untuk menggunakan replikasi terenkripsi: • Lapisan Soket Aman (SSL) harus diaktifkan pada basis data sumber MySQL eksternal. • Kunci klien dan sertifikat klien harus disiapkan untuk klaster DB Aurora MySQL. Selama replikasi terenkripsi, klaster DB Aurora MySQL berperan sebagai klien untuk server basis data MySQL. Sertifikat dan kunci untuk klien Aurora MySQL ada di dalam file dengan format .pem. 1. Pastikan bahwa Anda siap untuk replikasi terenkripsi: • Jika Anda tidak memiliki SSL yang diaktifkan pada basis data sumber MySQL eksternal dan tidak menyiapkan kunci klien dan sertifikat klien, aktifkan SSL pada server basis data MySQL serta buat kunci klien dan sertifikat klien yang diperlukan. • Jika SSL diaktifkan pada sumber eksternal, sediakan kunci dan sertifikat klien untuk klaster DB Aurora MySQL. Jika Anda tidak memilikinya, buat kunci dan sertifikat baru untuk klaster DB Aurora MySQL. Untuk menandatangani sertifikat klien, Anda harus memiliki kunci otoritas sertifikat yang Anda gunakan untuk mengonfigurasi SSL pada basis data sumber MySQL eksternal.

Mesin basis data	Petunjuk

Untuk informasi selengkapnya, lihat Creating SSL certificates and keys using openssl dalam dokumentasi MySQL.

Anda memerlukan sertifikat otoritas sertifikat, kunci klien, dan sertifikat klien.

2. Hubungkan ke klaster DB Aurora MySQL sebagai pengguna master menggunak an SSL.

   Untuk informasi tentang menghubungkan ke klaster DB Aurora MySQL dengan SSL, lihat Menggunakan TLS dengan klaster DB Aurora MySQL.

3. Jalankan prosedur tersimpan `mysql.rds_import_binlog_ssl _material` untuk mengimpor informasi SSL ke dalam klaster DB Aurora MySQL.

   Untuk parameter `ssl_material_value` , masukkan informasi dari file format .pem untuk klaster DB Aurora MySQL di payload JSON yang benar.

   Contoh berikut mengimpor informasi SSL ke dalam klaster DB Aurora MySQL. Dalam file berformat .pem, kode konten biasanya lebih panjang dari kode konten yang ditunjukkan dalam contoh.

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj
qP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96
xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/
i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22
1CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICA
TE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcj
qP3maAhDFcvBS7O6V
```

Mesin basis data	Petunjuk

```
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96
xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/
i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz22
1CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE
 KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pc
jqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSF
JuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJ
tjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMu
cxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

Lihat informasi yang lebih lengkap di mysql.rds_import_binlog_ssl_material dan Menggunakan TLS dengan klaster DB Aurora MySQL.

> **Note**
>
> Setelah menjalankan prosedur ini, rahasia disimpan dalam file. Untuk menghapus file nanti, Anda dapat menjalankan prosedur tersimpan mysql.rds_remove_binlog_ssl_material.

Untuk mengaktifkan pencatatan log biner pada basis data MySQL eksternal

1. Dari shell perintah, hentikan layanan mysql.

```
sudo service mysqld stop
```

2. Edit file `my.cnf` (file ini biasanya ada di bawah `/etc`).

Mesin basis data	Petunjuk

```
sudo vi /etc/my.cnf
```

Tambahkan opsi `log_bin` dan `server_id` ke bagian `[mysqld]`. Opsi `log_bin` menyediakan sebuah pengidentifikasi nama file untuk file log biner. Opsi `server_id` menyediakan pengidentifikasi unik untuk server dalam hubungan sumber-replika.

Jika replikasi terenkripsi tidak diperlukan, pastikan basis data MySQL eksternal dimulai dengan binlog diaktifkan dan SSL dinonaktifkan.

Berikut ini adalah entri yang relevan dalam file `/etc/my.cnf` untuk data yang tidak terenkripsi.

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1
```

Jika replikasi terenkripsi diperlukan, pastikan basis data MySQL eksternal dimulai dengan SSL dan binlog diaktifkan.

Entri dalam file `/etc/my.cnf` mencakup lokasi file .pem untuk server basis data MySQL.

```
log-bin=mysql-bin
server-id=2133421
innodb_flush_log_at_trx_commit=1
sync_binlog=1

# Setup SSL.
ssl-ca=/home/sslcerts/ca.pem
ssl-cert=/home/sslcerts/server-cert.pem
ssl-key=/home/sslcerts/server-key.pem
```

Mesin basis data	Petunjuk

Selain tersebut, opsi `sql_mode` untuk instans DB MySQL Anda harus diatur ke 0, atau tidak boleh disertakan dalam file my.cnf Anda.

Saat terhubung ke basis data MySQL eksternal, catat posisi log biner basis data MySQL eksternal.

```
mysql> SHOW MASTER STATUS;
```

Output Anda harus seperti yang berikut ini:

```
+------------------+----------+--------------+------------------
+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
 Executed_Gtid_Set |
+------------------+----------+--------------+------------------
+-------------------+
| mysql-bin.000031 |      107 |              |                  |
              |
+------------------+----------+--------------+------------------
+-------------------+
1 row in set (0.00 sec)
```

Untuk informasi selengkapnya, lihat Setting the replication source configuration dalam dokumentasi MySQL.

3. Mulai layanan mysql.

```
sudo service mysqld start
```

2. Pertahankan log biner pada sumber replikasi hingga tidak diperlukan lagi

Jika Anda menggunakan replikasi log biner MySQL, Amazon RDS tidak akan mengelola proses replikasi. Akibatnya, Anda harus memastikan bahwa file binlog pada sumber replikasi Anda

dipertahankan hingga perubahan sudah diterapkan ke replika. Dengan mempertahankannya, Anda akan dapat memulihkan basis data sumber Anda jika terjadi kegagalan.

Gunakan petunjuk berikut untuk mempertahankan log biner untuk mesin basis data Anda.

Mesin basis data	Petunjuk
Aurora MySQL	Untuk mempertahankan log biner pada klaster DB Aurora MySQL Anda tidak memiliki akses ke file binlog untuk klaster DB Aurora MySQL. Oleh karena itu, Anda harus memilih rentang waktu yang sesuai untuk mempertahankan file binlog pada sumber replikasi Anda untuk memastikan bahwa perubahan sudah diterapkan pada replika Anda sebelum file binlog dihapus oleh Amazon RDS. Anda dapat mempertahankan file binlog pada klaster DB Aurora MySQL hingga 90 hari. Jika Anda mengatur replikasi dengan basis data MySQL atau instans DB RDS for MySQL sebagai replika, dan basis data yang Anda buat sebagai replika berukuran sangat besar, pilih rentang waktu yang lama untuk mempertahankan file binlog hingga salinan awal basis data ke replika selesai dan lag replika telah mencapai 0. Untuk mengatur rentang waktu retensi log biner, gunakan prosedur mysql.rds_set_configuration dan tentukan parameter konfigurasi `'binlog retention hours'` bersama dengan jumlah jam untuk mempertahankan file binlog di klaster DB. Nilai maksimum untuk Aurora MySQL versi 2.11.0 dan lebih tinggi dan versi 3 adalah 2160 (90 hari). Contoh berikut menetapkan periode retensi untuk file binlog menjadi 6 hari: `CALL mysql.rds_set_configuration('binlog retention hours', 144);` Setelah replikasi dimulai, Anda dapat memverifikasi bahwa perubahan telah diterapkan ke replika Anda dengan menjalankan perintah `SHOW SLAVE STATUS` (Aurora MySQL versi 2) atau `SHOW REPLICA STATUS` (Aurora MySQL versi 3) pada replika Anda dan memeriksa bidang `Seconds behind master`. Jika bidang `Seconds behind master` adalah 0, maka tidak ada lag replika. Jika tidak ada lag replika,

Mesin basis data	Petunjuk
	kurangi durasi waktu retensi file binlog dengan mengatur parameter konfigurasi `binlog retention hours` ke rentang waktu yang lebih kecil. Jika pengaturan ini tidak ditentukan, nilai default untuk Aurora MySQL adalah 24 (1 hari). Jika Anda menentukan nilai untuk `'binlog retention hours'` yang lebih tinggi dari nilai maksimum, maka Aurora MySQL menggunakan nilai maksimum.
RDS for MySQL	Untuk mempertahankan log biner pada instans DB Amazon RDS Anda dapat mempertahankan file log biner pada instans DB Amazon RDS dengan mengatur jam retensi binlog sama seperti klaster DB Aurora MySQL, yang sudah dijelaskan dalam baris sebelumnya. Anda juga dapat mempertahankan file binlog pada instans DB Amazon RDS dengan membuat replika baca untuk instans DB. Replika baca ini bersifat sementara dan hanya untuk mempertahankan file binlog. Setelah replika baca dibuat, panggil prosedur mysql.rds_stop_replication pada replika baca. Saat replikasi dihentikan, Amazon RDS tidak akan menghapus file binlog mana pun pada sumber replikasi. Setelah Anda mengatur replikasi dengan replika permanen, Anda dapat menghapus replika baca saat lag replika (bidang `Seconds behind master`) antara sumber replikasi dan replika permanen Anda mencapai 0.
MySQL (eksternal)	Untuk mempertahankan log biner pada basis data MySQL eksternal Karena file binlog pada basis data MySQL eksternal tidak dikelola oleh Amazon RDS, file ini akan dipertahankan hingga Anda menghapusnya. Setelah replikasi dimulai, Anda dapat memverifikasi bahwa perubahan telah diterapkan ke replika Anda dengan menjalankan perintah `SHOW SLAVE STATUS` (Aurora MySQL versi 2) atau `SHOW REPLICA STATUS` (Aurora MySQL versi 3) pada replika Anda dan memeriksa bidang `Seconds behind master`. Jika bidang `Seconds behind master` adalah 0, maka tidak ada lag replika. Saat tidak ada lag replika, Anda dapat menghapus file binlog lama.

3. Buat snapshot atau dump sumber replikasi Anda

Anda dapat menggunakan snapshot atau dump sumber replikasi untuk memuat salinan acuan dasar data Anda ke replika Anda, lalu mulai mereplikasi dari titik tersebut.

Gunakan petunjuk berikut untuk membuat snapshot atau dump sumber replikasi untuk mesin basis data Anda.

Mesin basis data	Petunjuk

Aurora MySQL	Untuk membuat snapshot klaster DB Aurora MySQL
	1. Buat snapshot klaster DB untuk klaster DB Amazon Aurora Anda. Untuk informasi selengkapnya, lihat Membuat snapshot klaster DB.
	2. Buat klaster DB Aurora baru dengan memulihkan dari snapshot klaster DB yang baru saja Anda buat. Pastikan untuk mempertahankan grup parameter DB yang sama untuk klaster DB yang Anda pulihkan sebagai klaster DB asli Anda. Tindakan ini akan memastikan bahwa salinan klaster DB Anda memiliki pencatatan log biner aktif. Untuk informasi selengkapnya, lihat Memulihkan dari snapshot klaster DB.
	3. Dalam konsol, pilih Basis data lalu pilih instans primer (penulis) pada klaster DB Aurora Anda yang dipulihkan untuk menampilkan detailnya. Gulir ke Peristiwa Terbaru. Akan muncul pesan peristiwa yang menyertakan nama dan posisi file binlog. Pesan peristiwa memiliki format seperti berikut ini.
	`Binlog position from crash recovery is binlog-file-name binlog-position`
	Simpan nilai nama dan posisi file binlog saat Anda memulai replikasi.
	Anda juga dapat memperoleh nama dan posisi file binlog dengan memanggil perintah describe-events dari AWS CLI. Berikut ini adalah contoh perintah `describe-events` disertai dengan contoh output.
	`PROMPT> aws rds describe-events`
	`{`

Mesin basis data	Petunjuk

```
    "Events": [
        {
            "EventCategories": [],
            "SourceType": "db-instance",
            "SourceArn": "arn:aws:rds:us-west-2:123456789012:
db:sample-restored-instance",
            "Date": "2016-10-28T19:43:46.862Z",
            "Message": "Binlog position from crash recovery is mysql-
bin-changelog.000003 4278",
            "SourceIdentifier": "sample-restored-instance"
        }
    ]
}
```

Anda juga bisa mendapatkan nama dan posisi file binlog dengan memeriksa log kesalahan MySQL untuk posisi file binlog MySQL terakhir.

4. Jika target replika Anda adalah cluster Aurora DB yang dimiliki oleh yang Akun AWS lain, database MySQL eksternal, atau RDS untuk instans DB MySQL, maka Anda tidak dapat memuat data dari snapshot cluster Amazon Aurora DB. Sebagai gantinya, buat dump klaster DB Aurora Anda dengan menghubungkan ke klaster DB Anda menggunakan klien MySQL dan menerbitkan perintah `mysqldump` . Pastikan untuk menjalankan perintah `mysqldump` terhadap salinan klaster DB Aurora yang Anda buat. Berikut adalah contohnya.

```
PROMPT> mysqldump --databases <database_name> --single-transaction
--order-by-primary -r backup.sql -u <local_user> -p
```

5. Setelah Anda selesai membuat dump data Anda dari klaster DB Aurora yang baru dibuat, hapus klaster DB tersebut karena sudah tidak lagi diperlukan.

Mesin basis data	Petunjuk
RDS for MySQL	Untuk membuat snapshot instans DB Amazon RDS Buat replika baca instans DB Amazon RDS Anda. Untuk informasi selengkapnya, lihat Membuat replika baca dalam Panduan Pengguna Amazon Relational Database Service. 1. Hubungkan ke replika baca Anda dan hentikan replikasi dengan menjalankan prosedur mysql.rds_stop_replication. 2. Saat replika baca Dihentikan, hubungkan ke replika baca dan jalankan perintah `SHOW SLAVE STATUS` (Aurora MySQL versi 2) atau `SHOW REPLICA STATUS` (Aurora MySQL versi 3). Ambil nama file log biner saat ini dari bidang `Relay_Master_Log_File` dan posisi file log dari bidang `Exec_Master_Log_Pos` . Simpan nilai-nilai ini saat Anda memulai replikasi. 3. Jika replika baca tetap Dihentikan, buat snapshot DB replika baca. Untuk informasi selengkapnya, lihat Membuat snapshot DB dalam Panduan Pengguna Amazon Relational Database Service. 4. Hapus replika baca.

Mesin basis data	Petunjuk

MySQL (eksternal)	Untuk membuat dump basis data MySQL eksternal
	1. Sebelum Anda membuat dump, Anda harus memastikan bahwa lokasi binlog untuk dump sama barunya dengan data dalam instans sumber Anda. Untuk melakukan hal ini, Anda harus terlebih dahulu menghentikan operasi tulis apa pun ke instans dengan perintah berikut:
	`mysql> FLUSH TABLES WITH READ LOCK;`
	2. Buat dump basis data MySQL menggunakan perintah `mysqldump` seperti yang ditunjukkan berikut ini:
	`PROMPT> sudo mysqldump --databases <database_name> --master-data=2 --single-transaction \ --order-by-primary -r backup.sql -u <local_user> -p`
	3. Setelah Anda membuat dump, buka tabel dalam basis data MySQL Anda dengan perintah berikut:
	`mysql> UNLOCK TABLES;`

4. Muat snapshot atau dump ke target replika Anda

Jika Anda berencana memuat data dari dump basis data MySQL yang berada di luar Amazon RDS, Anda sebaiknya membuat instans EC2 dan menyalin file dump ini ke dalamnya, lalu memuat data ke klaster DB atau instans DB Anda dari instans EC2 tersebut. Dengan menggunakan pendekatan ini, Anda dapat mengompresi file dump sebelum menyalinnya ke instans EC2 untuk mengurangi biaya jaringan yang terkait dengan penyalinan data ke Amazon RDS. Anda juga dapat mengenkripsi file dump untuk mengamankan data saat data tersebut ditransfer melewati jaringan.

Gunakan petunjuk berikut untuk memuat snapshot atau dump sumber replikasi ke dalam target replika untuk mesin basis data Anda.

Mesin basis data	Petunjuk

Aurora MySQL	Untuk memuat snapshot atau dump ke klaster DB Aurora MySQL
	• Jika snapshot sumber replikasi Anda adalah snapshot klaster DB, Anda dapat memulihkan dari snapshot klaster DB untuk membuat klaster DB Aurora MySQL baru sebagai target replika Anda. Untuk informasi selengkapnya, lihat Memulihkan dari snapshot klaster DB.
	• Jika snapshot sumber replikasi Anda adalah snapshot DB, maka Anda dapat memigrasikan data dari snapshot DB Anda ke dalam klaster DB Aurora MySQL baru. Untuk informasi selengkapnya, lihat Memigrasikan data ke klaster DB Amazon Aurora MySQL.
	• Jika data dari sumber replikasi Anda merupakan output dari perintah `mysqldump` , maka ikuti langkah-langkah ini:
	1. Salin output perintah `mysqldump` dari sumber replikasi Anda ke lokasi yang juga dapat terhubung ke klaster DB Aurora MySQL Anda.
	2. Hubungkan ke klaster DB Aurora MySQL Anda dengan menggunakan perintah `mysql`. Berikut adalah contohnya.
	`PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p`
	3. Pada permintaan `mysql`, jalankan perintah `source` dan berikan nama file dump basis data Anda untuk memuat data ke dalam klaster DB Aurora MySQL, misalnya:
	`mysql> source backup.sql;`

RDS for MySQL	Untuk memuat dump ke instans DB Amazon RDS 1. Salin output perintah `mysqldump` dari sumber replikasi Anda ke lokasi yang juga dapat terhubung ke instans DB MySQL Anda. 2. Hubungkan ke instans DB MySQL Anda dengan menggunakan perintah `mysql`. Berikut adalah contohnya.

Mesin basis data	Petunjuk

	`PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p`
	3. Pada permintaan `mysql`, jalankan perintah `source` dan berikan nama file dump basis data Anda untuk memuat data ke dalam instans DB MySQL, misalnya:
	`mysql> source backup.sql;`

MySQL (eksternal)	Untuk memuat dump ke basis data MySQL eksternal
	Anda tidak dapat memuat snapshot DB atau snapshot klaster DB ke dalam basis data MySQL eksternal. Sebagai gantinya, Anda harus menggunakan output dari perintah `mysqldump` .
	1. Salin output perintah `mysqldump` dari sumber replikasi Anda ke lokasi yang juga dapat terhubung ke basis data MySQL Anda.
	2. Hubungkan ke basis data MySQL Anda dengan menggunakan perintah `mysql`. Berikut adalah contohnya.
	`PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p`
	3. Pada permintaan `mysql`, jalankan perintah `source` dan berikan nama file dump basis data Anda untuk memuat data ke basis data MySQL Anda. Berikut adalah contohnya.
	`mysql> source backup.sql;`

5. Buat pengguna replikasi pada sumber replikasi Anda

Buat ID pengguna pada sumber yang digunakan hanya untuk replikasi. Contoh berikut adalah untuk RDS untuk MySQL atau database sumber MySQL eksternal.

```
mysql> CREATE USER 'repl_user'@'domain_name' IDENTIFIED BY 'password';
```

Untuk database sumber MySQL Aurora, parameter cluster `skip_name_resolve` DB diatur ke 1 (0N) dan tidak dapat dimodifikasi, jadi Anda harus menggunakan alamat IP untuk host alih-alih nama domain. Untuk informasi selengkapnya, lihat skip_name_resolve di dokumentasi MySQL.

```
mysql> CREATE USER 'repl_user'@'IP_address' IDENTIFIED BY 'password';
```

Pengguna memerlukan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE`. Berikan hak akses ini kepada pengguna.

Jika Anda perlu menggunakan replikasi terenkripsi, wajibkan koneksi SSL untuk pengguna replikasi. Misalnya, Anda dapat menggunakan salah satu pernyataan berikut untuk meminta koneksi SSL di akun `repl_user` pengguna.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'IP_address';
```

```
GRANT USAGE ON *.* TO 'repl_user'@'IP_address' REQUIRE SSL;
```

> **Note**
>
> Jika `REQUIRE SSL` tidak disertakan, koneksi replikasi dapat kembali ke koneksi yang tidak terenkripsi tanpa peringatan.

6. Aktifkan replikasi pada target replika Anda

Sebelum Anda mengaktifkan replikasi, kami menyarankan agar Anda mengambil snapshot manual klaster DB Aurora MySQL atau target replika instans DB RDS for MySQL. Jika masalah muncul dan Anda harus membuat ulang replikasi dengan klaster DB atau target replika instans DB, Anda dapat memulihkan klaster DB atau instans DB dari snapshot ini daripada harus mengimpor data ke dalam target replika Anda lagi.

Gunakan petunjuk berikut untuk mengaktifkan replikasi untuk mesin basis data Anda.

Mesin basis data	Petunjuk
Aurora MySQL	Untuk mengaktifkan replikasi dari klaster DB Aurora MySQL 1. Temukan tempat awal untuk replikasi. Anda memerlukan nama file binlog dan posisi binlog. Jika target replika klaster DB Anda dibuat dari berikut ini: • Snapshot klaster DB - Ambil nama dan posisi file binlog dari peristiwa terbaru untuk klaster DB Anda yang dipulihkan, seperti yang ditunjukkan pada 3. Buat snapshot atau dump sumber replikasi Anda. • Snapshot DB - Anda mengambil nama dan posisi file binlog dari perintah `SHOW SLAVE STATUS` (Aurora MySQL versi 2) atau `SHOW REPLICA STATUS` (Aurora MySQL versi 3) saat Anda membuat snapshot sumber replikasi Anda. 2. Hubungkan ke klaster DB dan panggil prosedur berikut untuk memulai replikasi dengan sumber replikasi Anda menggunakan nama dan lokasi file log biner dari langkah sebelumnya: • mysql.rds_set_external_source (Aurora MySQL versi 3) • mysql.rds_set_external_master (Aurora MySQL versi 2) • mysql.rds_start_replication (semua versi) Contoh berikut ditujukan untuk Aurora MySQL versi 3. `CALL mysql.rds_set_external_source ('mydbinstance.123456789012 .us-east-1.rds.amazonaws.com', 3306, 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0); CALL mysql.rds_start_replication;` Untuk menggunakan enkripsi SSL, tetapkan nilai akhir menjadi `1` bukan `0`.
RDS for MySQL	Untuk mengaktifkan replikasi dari instans DB Amazon RDS 1. Jika target replika instans DB Anda dibuat dari snapshot DB, maka Anda memerlukan file binlog dan posisi binlog yang merupakan tempat awal untuk

Mesin basis data	Petunjuk
	replikasi. Anda mengambil nilai-nilai ini dari perintah `SHOW SLAVE STATUS` (Aurora MySQL versi 2) atau `SHOW REPLICA STATUS` (Aurora MySQL versi 3) ketika Anda membuat snapshot sumber replikasi Anda. 2. Hubungkan ke instans DB dan panggil prosedur mysql.rds_set_external_master (Aurora MySQL versi 2) atau mysql.rds_set_external_source (Aurora MySQL versi 3) dan mysql.rds_start_replication untuk memulai replikasi dengan sumber replikasi Anda. Gunakan nama dan lokasi file log biner dari langkah sebelumnya. Berikut adalah contohnya. `CALL mysql.rds_set_external_master ('mydbcluster.cluster-12345 6789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0); CALL mysql.rds_start_replication;` Untuk menggunakan enkripsi SSL, tetapkan nilai akhir menjadi `1` bukan `0`.

Mesin basis data	Petunjuk

MySQL (eksternal)	Untuk mengaktifkan replikasi dari basis data MySQL eksternal
	
	1. Ambil file binlog dan posisi binlog yang merupakan tempat awal untuk replikasi. Anda mengambil nilai-nilai ini dari perintah `SHOW SLAVE STATUS` (Aurora MySQL versi 2) atau `SHOW REPLICA STATUS` (Aurora MySQL versi 3) ketika Anda membuat snapshot sumber replikasi Anda. Jika target replika MySQL eksternal Anda diisi dari output perintah `mysqldump` dengan opsi `--master-data=2`, file binlog dan posisi binlog akan disertakan dalam output tersebut. Berikut adalah contohnya.
	
	`-- -- Position to start replication or point-in-time recovery from -- -- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031', MASTER_LOG_POS=107;`
	
	2. Hubungkan ke target replika MySQL eksternal, lalu terbitkan `CHANGE MASTER TO` dan `START SLAVE` (Aurora MySQL versi 2) atau `START REPLICA` (Aurora MySQL versi 3) untuk memulai replikasi dengan sumber replikasi Anda menggunakan nama dan lokasi file log biner dari langkah sebelumnya, misalnya:
	
	`CHANGE MASTER TO MASTER_HOST = 'mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', MASTER_PORT = 3306, MASTER_USER = 'repl_user', MASTER_PASSWORD = 'password', MASTER_LOG_FILE = 'mysql-bin-changelog.000031', MASTER_LOG_POS = 107; -- And one of these statements depending on your engine version: START SLAVE; -- Aurora MySQL version 2 START REPLICA; -- Aurora MySQL version 3`

Jika replikasi gagal, hal tersebut dapat mengakibatkan peningkatan besar I/O yang tidak disengaja pada replika, yang dapat menurunkan performa. Jika replikasi gagal atau tidak lagi diperlukan, Anda dapat menjalankan prosedur tersimpan mysql.rds_reset_external_master (Aurora MySQL versi 2) atau mysql.rds_reset_external_source (Aurora MySQL versi 3) untuk menghapus konfigurasi replikasi.

Mengatur lokasi untuk menghentikan replikasi ke replika baca

Di Aurora MySQL versi 3.04 dan yang lebih tinggi, Anda dapat memulai replikasi lalu menghentikannya di lokasi file log biner yang ditentukan menggunakan prosedur tersimpan mysql.rds_start_replication_until (Aurora MySQL versi 3).

Untuk memulai replikasi ke replika baca dan menghentikan replikasi di lokasi tertentu

1. Menggunakan klien MySQL, sambungkan ke replika Aurora MySQL DB cluster sebagai pengguna utama.
2. Jalankan prosedur yang tersimpan di mysql.rds_start_replication_until (Aurora MySQL versi 3).

   Contoh berikut memulai replikasi dan mereplikasi perubahan hingga mencapai lokasi `120` di file biner `mysql-bin-changelog.000777`. Dalam skenario pemulihan bencana, asumsikan bahwa lokasi `120` tepat sebelum bencana.

   ```
   call mysql.rds_start_replication_until(
     'mysql-bin-changelog.000777',
     120);
   ```

Replikasi berhenti secara otomatis ketika stop point tercapai. Peristiwa RDS berikut dibuat: `Replication has been stopped since the replica reached the stop point specified by the rds_start_replication_until stored procedure`.

Jika Anda menggunakan replikasi berbasis GTID, gunakan prosedur tersimpan mysql.rds_start_replication_until_gtid (Aurora MySQL versi 3), bukan prosedur tersimpan mysql.rds_start_replication_until (Aurora MySQL versi 3). Untuk informasi selengkapnya tentang replikasi berbasis GTID, lihat Menggunakan replikasi berbasis GTID untuk Amazon Aurora MySQL.

7. Pantau replika Anda

Saat Anda mengatur replikasi MySQL dengan klaster DB Aurora MySQL, Anda harus memantau peristiwa failover untuk klaster DB Aurora MySQL jika klaster DB ini merupakan target replika. Jika

terjadi failover, maka klaster DB yang merupakan target replika Anda dapat dibuat ulang pada host baru dengan alamat jaringan yang berbeda. Untuk informasi tentang cara memantau peristiwa failover, lihat Bekerja dengan pemberitahuan peristiwa Amazon RDS.

Anda juga dapat memantau seberapa jauh target replika tertinggal di belakang sumber replikasi dengan terhubung ke target replika dan menjalankan perintah `SHOW SLAVE STATUS` (Aurora MySQL versi 2) atau `SHOW REPLICA STATUS` (Aurora MySQL versi 3). Dalam output perintah, bidang `Seconds Behind Master` akan memberi tahu Anda seberapa jauh target replika tertinggal di belakang sumber.

Menyinkronkan kata sandi antara sumber dan target replikasi

Saat Anda mengubah akun pengguna dan kata sandi pada sumber replikasi menggunakan pernyataan SQL, perubahan tersebut direplikasi ke target replikasi secara otomatis.

Jika Anda menggunakan AWS Management Console, the AWS CLI, atau RDS API untuk mengubah kata sandi utama pada sumber replikasi, perubahan tersebut tidak secara otomatis direplikasi ke target replikasi. Jika Anda ingin menyinkronkan pengguna master dan kata sandi master antara sistem sumber dan target, Anda harus membuat sendiri perubahan yang sama pada target replikasi.

Menghentikan replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya

Untuk menghentikan replikasi binlog dengan instans DB MySQL, basis data MySQL eksternal, atau klaster DB Aurora lainnya, ikuti langkah-langkah ini, yang dibahas secara terperinci dalam topik ini.

1. Hentikan replikasi log biner pada target replika

2. Nonaktifkan pencatatan log biner pada sumber replikasi

1. Hentikan replikasi log biner pada target replika

Gunakan petunjuk berikut untuk menghentikan replikasi log biner untuk mesin basis data Anda.

Mesin basis data	Petunjuk
Aurora MySQL	Untuk menghentikan replikasi log biner pada target replika klaster DB Aurora MySQL

Mesin basis data	Petunjuk
	Hubungkan ke klaster DB Aurora yang merupakan target replika, dan panggil prosedur mysql.rds_stop_replication.
RDS for MySQL	Untuk menghentikan replikasi log biner pada instans DB Amazon RDS Hubungkan ke instans DB RDS yang merupakan target replika dan panggil prosedur mysql.rds_stop_replication.
MySQL (eksternal)	Untuk menghentikan replikasi log biner pada basis data MySQL eksternal Hubungkan ke basis data MySQL dan jalankan perintah `STOP SLAVE` (versi 5.7) atau `STOP REPLICA` (versi 8.0).

2. Nonaktifkan pencatatan log biner pada sumber replikasi

Gunakan petunjuk dalam tabel berikut untuk menonaktifkan pencatatan log biner pada sumber replikasi untuk mesin basis data Anda.

Mesin basis data	Petunjuk
Aurora MySQL	Untuk menonaktifkan pencatatan log biner pada klaster DB Amazon Aurora 1. Hubungkan ke klaster DB Aurora yang merupakan sumber replikasi. 2. Gunakan prosedur mysql.rds_set_configuration dan tentukan parameter konfigurasi `binlog retention hours`, dengan nilai `NULL`, seperti yang ditunjukkan pada contoh berikut. `CALL mysql.rds_set_configuration('binlog retention hours', NULL);`

Mesin basis data	Petunjuk
	> **Note** Anda tidak dapat menggunakan nilai `0` untuk `binlog retention hours`. 3. Atur parameter `binlog_format` ke `OFF` pada sumber replikasi. Parameter `binlog_format` ada di grup parameter klaster DB kustom yang terkait dengan klaster DB Anda. Setelah Anda mengubah nilai parameter `binlog_format` , boot ulang klaster DB Anda agar perubahan dapat diterapkan. Untuk informasi lebih lanjut, lihat Parameter instans DB dan klaster DB Amazon Aurora dan Memodifikasi parameter dalam grup parameter DB.
RDS for MySQL	Untuk menonaktifkan pencatatan log biner pada instans DB Amazon RDS Anda tidak dapat menonaktifkan pencatatan log biner secara langsung pada instans DB Amazon RDS, tetapi Anda dapat menonaktifkannya dengan melakukan hal berikut ini: 1. Nonaktifkan cadangan otomatis untuk instans DB. Anda dapat menonaktifkan cadangan otomatis dengan memodifikasi instans DB yang sudah ada dan mengatur Periode Retensi Cadangan menjadi 0. Untuk informasi selengkapnya, lihat Memodifikasi instans DB Amazon RDS dan Menggunakan cadangan dalam Panduan Pengguna Amazon Relational Database Service. 2. Hapus semua replika baca untuk instans DB. Untuk informasi selengkapnya, lihat Menggunakan replika baca dari instans DB MariaDB, MySQL, dan PostgreSQL dalam Panduan Pengguna Amazon Relational Database Service.

Mesin basis data	Petunjuk
MySQL (eksternal)	Untuk menonaktifkan pencatatan log biner pada basis data MySQL eksternal Hubungkan ke basis data MySQL dan panggil perintah `STOP REPLICATION` . 1. Dari shell perintah, hentikan layanan `mysqld`. ``` sudo service mysqld stop ``` 2. Edit file `my.cnf` (file ini biasanya ada dalam `/etc`). ``` sudo vi /etc/my.cnf ``` Hapus opsi `log_bin` dan `server_id` dari bagian `[mysqld]`. Untuk informasi selengkapnya, lihat Setting the replication source configuration dalam dokumentasi MySQL. 3. Mulai layanan mysql. ``` sudo service mysqld start ```

Menggunakan Amazon Aurora untuk menskalakan baca untuk basis data MySQL Anda

Anda dapat menggunakan Amazon Aurora dengan instans DB MySQL Anda untuk memanfaatkan kemampuan penskalaan baca Amazon Aurora dan memperluas beban kerja baca untuk instans DB MySQL Anda. Untuk menggunakan Aurora untuk menskalakan baca untuk instans DB MySQL Anda, buat klaster DB Amazon Aurora MySQL dan jadikan klaster DB ini sebagai replika baca instans DB MySQL Anda. Hal ini berlaku untuk satu instans DB RDS for MySQL, atau basis data MySQL yang dijalankan secara eksternal di luar Amazon RDS.

Untuk informasi tentang cara membuat klaster DB Amazon Aurora, lihat Membuat klaster DB Amazon Aurora.

Saat Anda mengatur replikasi antara instans DB MySQL dan klaster DB Amazon Aurora Anda, pastikan untuk mengikuti panduan ini:

- Gunakan alamat titik akhir klaster DB Amazon Aurora saat Anda merujuk ke klaster DB Amazon Aurora MySQL Anda. Jika terjadi failover, maka Replika Aurora yang dipromosikan ke instans primer untuk klaster DB Aurora MySQL akan terus menggunakan alamat titik akhir klaster DB.
- Pertahankan binlog pada instans penulis Anda hingga Anda memverifikasi bahwa binlog tersebut telah diterapkan ke Replika Aurora. Dengan mempertahankannya, Anda akan dapat memulihkan instans penulis Anda jika terjadi kegagalan.

Important

Saat menggunakan replikasi yang dikelola sendiri, Anda bertanggung jawab untuk memantau dan menyelesaikan masalah replikasi yang mungkin terjadi. Untuk informasi selengkapnya, lihat Mendiagnosis dan mengatasi jeda di antara replika baca.

Note

Izin yang diperlukan untuk memulai replikasi pada klaster DB Aurora MySQL dibatasi dan tidak tersedia untuk pengguna master Amazon RDS Anda. Oleh karena itu, Anda harus menggunakan prosedur mysql.rds_set_external_master (Aurora MySQL versi 2) atau mysql.rds_set_external_source (Aurora MySQL versi 3) dan mysql.rds_start_replication untuk mengatur replikasi antara klaster DB Aurora MySQL dan instans DB MySQL Anda.

Mulai replikasi antara instans sumber eksternal dan klaster DB Aurora MySQL

1. Buat instans DB MySQL sumber menjadi hanya baca:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

2. Jalankan perintah `SHOW MASTER STATUS` pada instans DB MySQL sumber untuk menentukan lokasi binlog. Anda akan menerima output yang serupa dengan contoh berikut:

```
File                        Position
-----------------------------------
```

```
 mysql-bin-changelog.000031      107
-----------------------------------
```

3. Salin basis data dari instans DB MySQL eksternal ke klaster DB Amazon Aurora MySQL menggunakan `mysqldump`. Untuk basis data yang sangat besar, Anda sebaiknya menggunakan prosedur dalam Mengimpor data ke instans DB MySQL atau MariaDB dengan waktu henti yang lebih singkat dalam Panduan Pengguna Amazon Relational Database Service.

   Untuk Linux, macOS, atau Unix:

```
mysqldump \
    --databases <database_name> \
    --single-transaction \
    --compress \
    --order-by-primary \
    -u local_user \
    -p local_password | mysql \
        --host aurora_cluster_endpoint_address \
        --port 3306 \
        -u RDS_user_name \
        -p RDS_password
```

   Untuk Windows:

```
mysqldump ^
    --databases <database_name> ^
    --single-transaction ^
    --compress ^
    --order-by-primary ^
    -u local_user ^
    -p local_password | mysql ^
        --host aurora_cluster_endpoint_address ^
        --port 3306 ^
        -u RDS_user_name ^
        -p RDS_password
```

> **Note**
>
> Pastikan tidak ada spasi antara opsi `-p` dan kata sandi yang dimasukkan.

Gunakan opsi `--host`, `--user (-u)`, `--port`, dan `-p` dalam perintah `mysql` untuk menentukan nama host, nama pengguna, port, dan kata sandi untuk terhubung ke klaster DB Aurora Anda. Nama host adalah nama DNS dari titik akhir klaster DB Amazon Aurora, misalnya, `mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com`. Anda dapat menemukan nilai titik akhir dalam detail klaster di Konsol Manajemen Amazon RDS.

4. Buat instans DB MySQL sumber menjadi dapat ditulis lagi:

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

Untuk informasi lebih lanjut tentang cara membuat cadangan untuk digunakan dengan replikasi, lihat Backing up a source or replica by making it read only dalam dokumentasi MySQL.

5. Dalam Konsol Manajemen Amazon RDS, tambahkan alamat IP server yang meng-host basis data MySQL sumber ke grup keamanan VPC untuk klaster DB Amazon Aurora. Untuk informasi selengkapnya tentang memodifikasi grup keamanan VPC, lihat Grup keamanan untuk VPC Anda dalam Panduan Pengguna Amazon Virtual Private Cloud.

   Anda juga mungkin harus mengonfigurasi jaringan lokal Anda untuk mengizinkan koneksi dari alamat IP klaster DB Amazon Aurora Anda agar klaster DB ini dapat berkomunikasi dengan instans MySQL sumber Anda. Untuk menemukan alamat IP klaster DB Amazon Aurora, gunakan perintah `host`.

```
host aurora_endpoint_address
```

   Nama host adalah nama DNS dari titik akhir klaster DB Amazon Aurora.

6. Dengan menggunakan klien pilihan Anda, hubungkan ke instans MySQL eksternal dan buat akun pengguna MySQL yang akan digunakan untuk replikasi. Akun ini digunakan hanya untuk replikasi dan harus dibatasi pada domain Anda untuk meningkatkan keamanan. Berikut adalah contohnya.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
```

7. Untuk instans MySQL eksternal, berikan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE` kepada pengguna replikasi Anda. Misalnya, untuk memberikan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada semua basis data bagi pengguna '`repl_user`' untuk domain Anda, jalankan perintah berikut.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
    IDENTIFIED BY 'password';
```

8. Ambil snapshot manual klaster DB Aurora MySQL untuk dijadikan sebagai replika baca sebelum mengatur replikasi. Jika Anda harus membuat ulang replikasi dengan klaster DB sebagai replika baca, Anda dapat memulihkan klaster DB Aurora MySQL dari snapshot ini daripada harus mengimpor data dari instans DB MySQL ke dalam klaster DB Aurora MySQL baru.
9. Jadikan klaster DB Amazon Aurora sebagai replika. Hubungkan ke klaster DB Amazon Aurora sebagai pengguna master dan identifikasi basis data MySQL sumber sebagai master replikasi dengan menggunakan prosedur mysql.rds_set_external_master (Aurora MySQL versi 2) atau mysql.rds_set_external_source (Aurora MySQL versi 3) dan mysql.rds_start_replication.

   Gunakan nama file log master dan posisi log master yang Anda tentukan pada Langkah 2. Berikut adalah contohnya.

   ```
   For Aurora MySQL version 2:
   CALL mysql.rds_set_external_master ('mymasterserver.mydomain.com', 3306,
       'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);

   For Aurora MySQL version 3:
   CALL mysql.rds_set_external_source ('mymasterserver.mydomain.com', 3306,
       'repl_user', 'password', 'mysql-bin-changelog.000031', 107, 0);
   ```

10. Pada klaster DB Amazon Aurora, panggil prosedur mysql.rds_start_replication untuk memulai replikasi.

    ```
    CALL mysql.rds_start_replication;
    ```

Setelah Anda membuat replikasi antara instans DB MySQL sumber dan klaster DB Amazon Aurora, Anda dapat menambahkan Replika Aurora ke klaster DB Amazon Aurora Anda. Kemudian, Anda dapat terhubung ke Replika Aurora untuk menskalakan baca data Anda. Untuk informasi tentang cara membuat Replika Aurora, lihat Menambahkan Replika Aurora ke klaster DB.

Mengoptimalkan replikasi log biner

Dari informasi berikut, Anda dapat mempelajari cara mengoptimalkan performa replikasi log biner dan memecahkan masalah terkait di Aurora MySQL.

> **Tip**
>
> Diskusi ini mengasumsikan bahwa Anda sudah mengenal mekanisme replikasi log biner MySQL dan cara kerjanya. Untuk informasi latar belakang, lihat Replication Implementation dalam dokumentasi MySQL.

Replikasi log biner multithreaded (Aurora MySQL versi 3)

Dengan replikasi log biner multi-thread, thread SQL membaca peristiwa dari log relai dan mengantrekannya agar thread pekerja SQL dapat diterapkan. Thread pekerja SQL dikelola oleh thread koordinator. Peristiwa log biner diterapkan secara paralel jika memungkinkan.

Ketika instance Aurora MySQL dikonfigurasi untuk menggunakan replikasi log biner, secara default instance replika menggunakan replikasi single-threaded untuk versi MySQL Aurora yang lebih rendah dari 3,04. Untuk mengaktifkan replikasi multi-thread, Anda memperbarui parameter `replica_parallel_workers` ke nilai yang lebih besar dari nol di grup parameter kustom Anda.

Untuk Aurora MySQL versi 3.04 dan lebih tinggi, replikasi multithreaded secara default, dengan disetel ke. `replica_parallel_workers` `4` Anda dapat memodifikasi parameter ini di grup parameter kustom Anda.

Opsi konfigurasi berikut dapat membantu Anda menyesuaikan replikasi multi-thread. Untuk informasi penggunaan, lihat Replication and Binary Logging Options and Variables dalam Panduan Referensi MySQL.

Konfigurasi yang optimal akan bergantung pada beberapa faktor. Misalnya, performa replikasi log biner dipengaruhi oleh karakteristik beban kerja basis data Anda dan kelas instans DB tempat replika berjalan. Oleh karena itu, kami menyarankan Anda menguji secara menyeluruh semua perubahan pada parameter konfigurasi ini sebelum menerapkan pengaturan parameter baru ke instance produksi:

- `binlog_group_commit_sync_delay`
- `binlog_group_commit_sync_no_delay_count`
- `binlog_transaction_dependency_history_size`
- `binlog_transaction_dependency_tracking`
- `replica_preserve_commit_order`

- `replica_parallel_type`
- `replica_parallel_workers`

Di Aurora MySQL versi 3.06 dan lebih tinggi, Anda dapat meningkatkan kinerja replika log biner saat mereplikasi transaksi untuk tabel besar dengan lebih dari satu indeks sekunder. Fitur ini memperkenalkan kumpulan utas untuk menerapkan perubahan indeks sekunder secara paralel pada replika binlog. Fitur ini dikendalikan oleh parameter cluster `aurora_binlog_replication_sec_index_parallel_workers` DB, yang mengontrol jumlah total thread paralel yang tersedia untuk menerapkan perubahan indeks sekunder. Parameter diatur ke `0` (dinonaktifkan) secara default. Mengaktifkan fitur ini tidak memerlukan instance restart. Untuk mengaktifkan fitur ini, hentikan replikasi yang sedang berlangsung, atur jumlah thread parallel worker yang diinginkan, lalu mulai replikasi lagi.

Anda juga dapat menggunakan parameter ini sebagai variabel global, di mana *n* adalah jumlah thread pekerja paralel:

```
SET global aurora_binlog_replication_sec_index_parallel_workers=n;
```

Mengoptimalkan replikasi binlog (Aurora MySQL 2.10 dan lebih tinggi)

Di Aurora MySQL 2.10 dan lebih tinggi, Aurora secara otomatis menerapkan optimisasi yang dikenal sebagai cache I/O binlog ke replikasi log biner. Dengan membuat cache peristiwa binlog yang paling baru di-commit, optimisasi ini dirancang untuk meningkatkan performa thread dump binlog sambil membatasi dampak pada transaksi latar depan di instans sumber binlog.

> **Note**
>
> Memori yang digunakan untuk fitur ini tidak bergantung pada pengaturan `binlog_cache` MySQL.
> Fitur ini tidak berlaku untuk instans Aurora DB yang menggunakan kelas instans `db.t2` dan `db.t3`.

Anda tidak perlu menyesuaikan parameter konfigurasi apa pun untuk mengaktifkan optimisasi ini. Secara khusus, jika Anda menyesuaikan parameter konfigurasi `aurora_binlog_replication_max_yield_seconds` ke nilai non-nol di versi Aurora MySQL sebelumnya, atur kembali ke nol untuk Aurora MySQL 2.10 dan lebih tinggi.

Variabel status `aurora_binlog_io_cache_reads` dan `aurora_binlog_io_cache_read_requests` tersedia di Aurora MySQL 2.10 dan lebih tinggi. Variabel status ini membantu Anda untuk memantau seberapa sering data dibaca dari cache I/O binlog.

- `aurora_binlog_io_cache_read_requests` menunjukkan jumlah permintaan baca I/O binlog dari cache.
- `aurora_binlog_io_cache_reads`: menunjukkan jumlah baca I/O binlog yang mengambil informasi dari cache.

Kueri SQL berikut menghitung persentase permintaan baca binlog yang memanfaatkan informasi cache. Dalam hal ini, makin dekat rasionya ke 100, makin baik.

```
mysql> SELECT
  (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_reads')
  / (SELECT VARIABLE_VALUE FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME='aurora_binlog_io_cache_read_requests')
  * 100
  as binlog_io_cache_hit_ratio;
+---------------------------+
| binlog_io_cache_hit_ratio |
+---------------------------+
|         99.99847949080622 |
+---------------------------+
```

Fitur cache I/O binlog juga menyertakan metrik baru yang terkait dengan thread dump binlog. Thread dump adalah thread yang dibuat saat replika binlog baru terhubung ke instans sumber binlog.

Metrik thread dump dicetak ke log basis data setiap 60 detik dengan awalan `[Dump thread metrics]`. Metrik ini mencakup informasi untuk setiap replika binlog seperti `Secondary_id`, `Secondary_uuid`, nama file binlog, dan posisi yang dibaca setiap replika. Metrik ini juga mencakup `Bytes_behind_primary` yang merepresentasikan jarak dalam byte antara sumber replikasi dan replika. Metrik ini mengukur lag thread I/O replika. Angka tersebut berbeda dengan lag thread pengaplikasi SQL replika, yang direpresentasikan oleh metrik `seconds_behind_master` pada replika binlog. Anda dapat menentukan apakah replika binlog mengimbangi atau tertinggal di belakang sumber dengan memeriksa apakah jaraknya berkurang atau bertambah.

Mengoptimalkan replikasi binlog (Aurora MySQL versi 2 hingga 2.09)

Untuk mengoptimalkan replikasi log biner untuk Aurora MySQL, Anda perlu menyesuaikan parameter optimisasi tingkat klaster berikut. Parameter ini membantu Anda menentukan keseimbangan yang tepat antara latensi pada instans sumber binlog dan lag replikasi.

- `aurora_binlog_use_large_read_buffer`
- `aurora_binlog_read_buffer_size`
- `aurora_binlog_replication_max_yield_seconds`

> **Note**
>
> Untuk klaster yang kompatibel dengan MySQL 5.7, Anda dapat menggunakan parameter ini di Aurora MySQL versi 2 hingga 2.09.*. Di Aurora MySQL 2.10.0 dan lebih tinggi, parameter ini digantikan oleh optimisasi cache I/O binlog dan Anda tidak perlu menggunakannya.

Topik

- Gambaran umum buffer baca besar dan optimisasi hasil maks
- Parameter terkait
- Mengaktifkan mekanisme hasil maksimal untuk replikasi log biner
- Menonaktifkan optimisasi hasil maksimal replikasi log biner
- Menonaktifkan buffer baca besar

Gambaran umum buffer baca besar dan optimisasi hasil maks

Anda mungkin mengalami penurunan performa replikasi log biner saat thread dump log biner mengakses volume klaster Aurora sementara klaster ini memproses sejumlah besar transaksi. Anda dapat menggunakan parameter `aurora_binlog_use_large_read_buffer`, `aurora_binlog_replication_max_yield_seconds`, dan `aurora_binlog_read_buffer_size` untuk membantu meminimalkan jenis pertentangan ini.

Misalkan Anda memiliki situasi di mana `aurora_binlog_replication_max_yield_seconds` disetel ke nilai yang lebih besar dari 0 dan file binlog terkini thread dump berstatus aktif. Dalam hal ini, thread dump log biner menunggu hingga beberapa detik agar file binlog saat ini diisi oleh

transaksi. Periode tunggu ini menghindari pertentangan yang dapat timbul dari replikasi setiap binlog peristiwa secara individual. Namun, hal ini akan meningkatkan lag replika untuk replika log biner. Replika tersebut dapat tertinggal di belakang sumber dengan jumlah detik yang sama seperti pengaturan `aurora_binlog_replication_max_yield_seconds`.

File binlog terkini adalah file binlog yang sedang dibaca oleh thread dump untuk melakukan replikasi. Kami menganggap bahwa file binlog aktif jika file binlog ini memperbarui atau terbuka untuk diperbarui oleh transaksi masuk. Setelah Aurora MySQL mengisi file binlog aktif, MySQL membuat dan beralih ke file binlog baru. File binlog lama menjadi tidak aktif. File ini tidak diperbarui oleh transaksi masuk lagi.

> **Note**
>
> Sebelum menyesuaikan parameter ini, ukur latensi dan throughput transaksi Anda dari waktu ke waktu. Anda mungkin menemukan bahwa performa replikasi log biner stabil dan memiliki latensi rendah bahkan jika ada pertentangan sesekali.

`aurora_binlog_use_large_read_buffer`

Jika parameter ini diatur ke 1, Aurora MySQL mengoptimalkan replikasi log biner berdasarkan pengaturan parameter `aurora_binlog_read_buffer_size` dan `aurora_binlog_replication_max_yield_seconds`. Jika `aurora_binlog_use_large_read_buffer` adalah 0, Aurora MySQL mengabaikan nilai parameter `aurora_binlog_read_buffer_size` dan `aurora_binlog_replication_max_yield_seconds`.

`aurora_binlog_read_buffer_size`

Thread dump log biner dengan buffer baca yang lebih besar meminimalkan jumlah operasi I/O baca dengan membaca lebih banyak peristiwa untuk setiap I/O. Parameter `aurora_binlog_read_buffer_size` mengatur ukuran buffer baca. Buffer baca yang besar dapat mengurangi pertentangan log biner untuk beban kerja yang menghasilkan data binlog dalam jumlah besar.

> **Note**
>
> Parameter ini hanya berpengaruh ketika klaster juga memiliki pengaturan `aurora_binlog_use_large_read_buffer=1`.

Meningkatkan ukuran buffer baca tidak memengaruhi performa replikasi log biner. Thread dump log biner tidak menunggu pembaruan transaksi untuk mengisi buffer baca.

`aurora_binlog_replication_max_yield_seconds`

Jika beban kerja Anda memerlukan latensi transaksi yang rendah, dan Anda dapat menoleransi beberapa lag replikasi, Anda dapat meningkatkan parameter `aurora_binlog_replication_max_yield_seconds`. Parameter ini mengontrol properti hasil maksimum replikasi log biner di klaster Anda.

> **Note**
>
> Parameter ini hanya berpengaruh ketika klaster juga memiliki pengaturan `aurora_binlog_use_large_read_buffer=1`.

Aurora MySQL mengenali perubahan apa pun pada nilai parameter `aurora_binlog_replication_max_yield_seconds` segera. Anda tidak perlu memulai ulang instans DB. Namun, saat Anda mengaktifkan pengaturan ini, thread dump hanya mulai memberikan hasil saat file binlog saat ini mencapai ukuran maksimum 128 MB dan dirotasi ke file baru.

Parameter terkait

Gunakan parameter klaster DB berikut untuk mengaktifkan optimisasi binlog.

Parameter	Default	Nilai Valid	Deskripsi
`aurora_binlog_use_large_read_buffer`	1	0, 1	Opsi untuk mengaktifkan fitur peningkatan replikasi. Jika nilainya 1, thread dump log biner menggunakan `aurora_binlog_read_buffer_size` untuk replikasi log biner; jika tidak,

Parameter	Default	Nilai Valid	Deskripsi
			ukuran buffer default (8K) digunakan. Tidak digunakan di Aurora MySQL versi 3.
`aurora_binlog_read_buffer_size`	5242880	8192-536870912	Ukuran buffer baca yang digunakan oleh thread dump log biner saat parameter `aurora_binlog_use_large_read_buffer` disetel ke 1. Tidak digunakan di Aurora MySQL versi 3.

Parameter	Default	Nilai Valid	Deskripsi
`aurora_binlog_replication_max_yield_seconds`	0	0-36000	Untuk Aurora MySQL versi 2.07.*, nilai maksimum yang diterima adalah 45. Anda dapat menyetelnya ke nilai yang lebih tinggi pada versi 2.09 dan yang lebih baru. Untuk versi 2, parameter ini hanya berfungsi jika parameter `aurora_binlog_use_large_read_buffer` diatur menjadi 1.

Mengaktifkan mekanisme hasil maksimal untuk replikasi log biner

Anda dapat mengaktifkan optimisasi hasil maksimal replikasi log biner sebagai berikut. Hal ini akan meminimalkan latensi untuk transaksi pada instans sumber binlog. Namun, Anda mungkin mengalami lag replikasi yang lebih tinggi.

Untuk mengaktifkan optimisasi binlog hasil maksimal untuk klaster Aurora MySQL

1. Buat atau edit grup parameter klaster DB menggunakan pengaturan parameter berikut:
   - `aurora_binlog_use_large_read_buffer`: aktifkan dengan nilai `ON` atau 1.
   - `aurora_binlog_replication_max_yield_seconds`: tentukan nilai lebih besar dari 0.
2. Kaitkan grup parameter klaster DB dengan klaster Aurora MySQL yang berfungsi sebagai sumber binlog. Untuk melakukannya, ikuti prosedur dalam Bekerja dengan grup parameter.

3. Konfirmasikan bahwa perubahan parameter berlaku. Untuk melakukannya, jalankan kueri berikut pada instans sumber binlog.

```
SELECT @@aurora_binlog_use_large_read_buffer,
 @@aurora_binlog_replication_max_yield_seconds;
```

Output Anda harus seperti yang berikut ini.

```
+--------------------------------------
+-----------------------------------------------+
| @@aurora_binlog_use_large_read_buffer |
 @@aurora_binlog_replication_max_yield_seconds |
+--------------------------------------
+-----------------------------------------------+
|                                     1 |
  45 |
+--------------------------------------
+-----------------------------------------------+
```

Menonaktifkan optimisasi hasil maksimal replikasi log biner

Anda dapat menonaktifkan optimisasi hasil maksimal replikasi log biner sebagai berikut. Hal ini akan meminimalkan lag replikasi. Namun, Anda mungkin mengalami latensi yang lebih tinggi untuk transaksi pada instans sumber binlog.

Untuk menonaktifkan optimisasi binlog hasil maksimal untuk klaster Aurora MySQL

1. Pastikan grup parameter klaster DB yang terkait dengan klaster Aurora MySQL memiliki `aurora_binlog_replication_max_yield_seconds` yang diatur ke 0. Untuk informasi selengkapnya tentang cara mengatur parameter konfigurasi menggunakan grup parameter, lihat Bekerja dengan grup parameter.
2. Konfirmasikan bahwa perubahan parameter berlaku. Untuk melakukannya, jalankan kueri berikut pada instans sumber binlog.

```
SELECT @@aurora_binlog_replication_max_yield_seconds;
```

Output Anda harus seperti yang berikut ini.

```
+-----------------------------------------------+
```

```
| @@aurora_binlog_replication_max_yield_seconds |
+-----------------------------------------------+
|                                             0 |
+-----------------------------------------------+
```

Menonaktifkan buffer baca besar

Anda dapat menonaktifkan seluruh fitur buffer baca besar sebagai berikut.

Untuk menonaktifkan buffer baca log biner besar untuk klaster Aurora MySQL

1. Atur ulang `aurora_binlog_use_large_read_buffer` ke `OFF` atau 0.

   Pastikan grup parameter klaster DB yang terkait dengan klaster Aurora MySQL memiliki `aurora_binlog_use_large_read_buffer` yang diatur ke 0. Untuk informasi selengkapnya tentang cara mengatur parameter konfigurasi menggunakan grup parameter, lihat Bekerja dengan grup parameter.
2. Pada instans sumber binlog, jalankan kueri berikut.

   ```
   SELECT @@ aurora_binlog_use_large_read_buffer;
   ```

   Output Anda harus seperti yang berikut ini.

   ```
   +---------------------------------------+
   | @@aurora_binlog_use_large_read_buffer |
   +---------------------------------------+
   |                                     0 |
   +---------------------------------------+
   ```

Menyiapkan binlog yang ditingkatkan

Binlog yang ditingkatkan akan mengurangi overhead performa komputasi yang disebabkan oleh pengaktifan binlog, yang dapat mencapai hingga 50% dalam kasus tertentu. Dengan binlog yang ditingkatkan, overhead ini dapat dikurangi menjadi sekitar 13%. Untuk mengurangi overhead, binlog yang ditingkatkan menulis log biner dan transaksi ke penyimpanan secara paralel, yang meminimalkan data yang ditulis pada waktu commit transaksi.

Menggunakan binlog yang ditingkatkan juga meningkatkan waktu pemulihan basis data setelah mulai ulang dan failover hingga 99% dibandingkan dengan binlog MySQL komunitas. Binlog yang ditingkatkan kompatibel dengan beban kerja berbasis binlog yang ada, dan Anda berinteraksi dengannya dengan cara yang sama seperti Anda berinteraksi dengan binlog MySQL komunitas.

Binlog yang ditingkatkan tersedia di Aurora MySQL versi 3.03.1 dan lebih tinggi.

Topik

- Mengonfigurasi parameter binlog yang ditingkatkan
- Parameter terkait lainnya
- Perbedaan antara binlog yang ditingkatkan dan binlog MySQL komunitas
- CloudWatch Metrik Amazon untuk binlog yang disempurnakan
- Batasan binlog yang ditingkatkan

Mengonfigurasi parameter binlog yang ditingkatkan

Anda dapat beralih antara binlog MySQL komunitas dan binlog yang ditingkatkan dengan mengaktifkan/menonaktifkan parameter binlog yang ditingkatkan. Konsumen binlog yang ada dapat terus membaca dan mengonsumsi file binlog tanpa kesenjangan dalam urutan file binlog.

Untuk mengaktifkan binlog yang ditingkatkan

Parameter	Default	Deskripsi
`binlog_format`	–	Atur parameter `binlog_format` ke format pencatatan log biner pilihan Anda untuk mengaktifkan binlog yang ditingkatkan. Pastikan `binlog_format parameter` tidak diatur ke OFF. Untuk informasi selengkapnya, lihat Mengonfigurasi pencatatan log biner Aurora MySQL.
`aurora_enhanced_binlog`	`0`	Tetapkan nilai parameter ini ke 1 dalam grup parameter

Parameter	Default	Deskripsi
		klaster DB yang terkait dengan klaster Aurora MySQL. Ketika Anda mengubah nilai parameter ini, Anda harus melakukan boot ulang instans penulis ketika nilai `DBClusterParameterGroupStatus` ditampilkan sebagai `pending-reboot` .
`binlog_backup`	`1`	Nonaktifkan parameter ini untuk mengaktifkan binlog yang ditingkatkan. Untuk melakukannya, tetapkan nilai parameter ini ke `0`.
`binlog_replication_globaldb`	`1`	Nonaktifkan parameter ini untuk mengaktifkan binlog yang ditingkatkan. Untuk melakukannya, tetapkan nilai parameter ini ke `0`.

Important

Anda dapat menonaktifkan `binlog_replication_globaldb` parameter `binlog_backup` dan hanya ketika Anda menggunakan binlog yang ditingkatkan.

Untuk menonaktifkan binlog yang ditingkatkan

Parameter	Deskripsi
`aurora_enhanced_binlog`	Tetapkan nilai parameter ini ke `0` dalam grup parameter klaster DB yang terkait dengan klaster Aurora MySQL. Setiap kali Anda

Parameter	Deskripsi
	mengubah nilai parameter ini, Anda harus melakukan boot ulang instans penulis ketika nilai `DBClusterParameterGroupStatus` ditampilkan sebagai `pending-reboot` .
`binlog_backup`	Aktifkan parameter ini saat Anda menonaktifkan binlog yang ditingkatkan. Untuk melakukannya, tetapkan nilai parameter ini ke 1.
`binlog_replication_globaldb`	Aktifkan parameter ini saat Anda menonaktifkan binlog yang ditingkatkan. Untuk melakukannya, tetapkan nilai parameter ini ke 1.

Untuk memeriksa apakah binlog yang ditingkatkan diaktifkan, gunakan perintah berikut di klien MySQL:

```
mysql>show status like 'aurora_enhanced_binlog';

+------------------------+--------+
| Variable_name          | Value  |
+------------------------+--------+
| aurora_enhanced_binlog | ACTIVE |
+------------------------+--------+
1 row in set (0.00 sec)
```

Saat binlog yang ditingkatkan diaktifkan, output-nya akan menampilkan `ACTIVE` untuk `aurora_enhanced_binlog`.

Parameter terkait lainnya

Saat Anda mengaktifkan binlog yang ditingkatkan, parameter berikut akan terpengaruh:

- Parameter `max_binlog_size` terlihat, tetapi tidak dapat dimodifikasi. Nilai default-nya, `134217728`, secara otomatis disesuaikan menjadi `268435456` saat binlog yang ditingkatkan diaktifkan.

- Tidak seperti di binlog MySQL komunitas, `binlog_checksum` tidak bertindak sebagai parameter dinamis ketika binlog yang ditingkatkan diaktifkan. Agar perubahan pada parameter ini berlaku, Anda harus melakukan boot ulang klaster DB secara manual bahkan jika `ApplyMethod` adalah `immediate`.
- Nilai yang Anda tetapkan pada parameter `binlog_order_commits` tidak berpengaruh pada urutan commit saat binlog yang ditingkatkan diaktifkan. Commit selalu diperintahkan tanpa implikasi performa lebih lanjut.

Perbedaan antara binlog yang ditingkatkan dan binlog MySQL komunitas

Binlog yang ditingkatkan berinteraksi secara berbeda dengan klon, cadangan, dan basis data global Aurora jika dibandingkan dengan binlog MySQL komunitas. Sebaiknya Anda memahami perbedaan berikut sebelum menggunakan binlog yang ditingkatkan.

- File binlog yang ditingkatkan dari klaster DB sumber tidak tersedia di klaster DB yang dikloning.
- File binlog yang ditingkatkan tidak disertakan dalam cadangan Aurora. Oleh karena itu, file binlog yang ditingkatkan dari klaster DB sumber tidak tersedia setelah memulihkan klaster DB meskipun ada periode retensi yang ditetapkan padanya.
- Ketika digunakan dengan basis data global Aurora, file binlog yang ditingkatkan untuk klaster DB primer tidak direplikasi ke klaster DB di wilayah sekunder.

Contoh

Contoh berikut menggambarkan perbedaan antara binlog yang ditingkatkan dan binlog MySQL komunitas.

Pada klaster DB yang dipulihkan atau dikloning

Saat binlog yang ditingkatkan diaktifkan, file binlog historis tidak tersedia di klaster DB yang dipulihkan atau dikloning. Setelah operasi pemulihan atau kloning, jika binlog dihidupkan, cluster DB baru mulai menulis urutan file binlognya sendiri, mulai dari 1 (.000001)mysql-bin-changelog.

Untuk mengaktifkan binlog yang ditingkatkan setelah operasi pemulihan atau kloning, atur parameter klaster DB yang diperlukan pada klaster DB yang dipulihkan atau dikloning. Untuk informasi selengkapnya, lihat Mengonfigurasi parameter binlog yang ditingkatkan.

Example Operasi kloning atau pemulihan yang dilakukan saat binlog yang ditingkatkan diaktifkan

Klaster DB Sumber:

```
mysql> show binary logs;

+--------------------------+-----------+-----------+
| Log_name                 | File_size | Encrypted |
+--------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog turned on
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog turned on
+--------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

Pada klaster DB yang dipulihkan atau dikloning, file binlog tidak akan dicadangkan saat binlog yang ditingkatkan diaktifkan. Untuk menghindari diskontinuitas dalam data binlog, file binlog yang ditulis sebelum binlog yang ditingkatkan diaktifkan juga tidak akan tersedia.

```
mysql>show binary logs;

+--------------------------+-----------+-----------+
| Log_name                 | File_size | Encrypted |
+--------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        | --> New sequence of Binlog files
+--------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example Operasi kloning atau pemulihan yang dilakukan saat binlog yang ditingkatkan dinonaktifkan

Klaster DB sumber:

```
mysql>show binary logs;

+--------------------------+-----------+-----------+
| Log_name                 | File_size | Encrypted |
+--------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
```

```
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

Pada klaster DB yang dipulihkan atau dikloning, file binlog yang ditulis setelah binlog yang ditingkatkan dinonaktifkan akan tersedia.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Pada basis data global Amazon Aurora

Pada basis data global Amazon Aurora, data binlog klaster DB primer tidak direplikasi ke klaster DB sekunder. Setelah proses failover lintas Wilayah, data binlog tidak tersedia di klaster DB primer yang baru dipromosikan. Jika binlog dihidupkan, cluster DB yang baru dipromosikan memulai urutan file binlognya sendiri, mulai dari 1 (mysql-bin-changelog.000001).

Untuk mengaktifkan binlog yang ditingkatkan setelah failover, Anda harus mengatur parameter klaster DB yang diperlukan pada klaster DB sekunder. Untuk informasi selengkapnya, lihat Mengonfigurasi parameter binlog yang ditingkatkan.

Example Operasi failover basis data global dilakukan saat binlog yang ditingkatkan diaktifkan

Klaster DB primer lama (sebelum failover):

```
mysql>show binary logs;
```

```
+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        |
| mysql-bin-changelog.000003 |       156 | No        |
| mysql-bin-changelog.000004 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000005 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000006 |       156 | No        | --> Enhanced Binlog enabled
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

Klaster DB primer baru (setelah failover):

File binlog tidak direplikasi ke wilayah sekunder saat binlog yang ditingkatkan diaktifkan. Untuk menghindari diskontinuitas dalam data binlog, file binlog yang ditulis sebelum binlog yang ditingkatkan diaktifkan tidak akan tersedia.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        | --> Fresh sequence of Binlog
 files
+----------------------------+-----------+-----------+
1 row in set (0.00 sec)
```

Example Operasi failover basis data global dilakukan ketika binlog yang ditingkatkan dinonaktifkan

Klaster DB Sumber:

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000001 |       156 | No        |
| mysql-bin-changelog.000002 |       156 | No        | --> Enhanced Binlog enabled
```

```
| mysql-bin-changelog.000003 |       156 | No        | --> Enhanced Binlog enabled
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
6 rows in set (0.00 sec)
```

Klaster DB yang dipulihkan atau dikloning:

File binlog yang ditulis setelah binlog yang ditingkatkan dinonaktifkan akan direplikasi dan tersedia di klaster DB yang baru dipromosikan.

```
mysql>show binary logs;

+----------------------------+-----------+-----------+
| Log_name                   | File_size | Encrypted |
+----------------------------+-----------+-----------+
| mysql-bin-changelog.000004 |       156 | No        |
| mysql-bin-changelog.000005 |       156 | No        |
| mysql-bin-changelog.000006 |       156 | No        |
+----------------------------+-----------+-----------+
3 rows in set (0.00 sec)
```

CloudWatch Metrik Amazon untuk binlog yang disempurnakan

CloudWatch Metrik Amazon berikut diterbitkan hanya ketika binlog yang disempurnakan diaktifkan.

CloudWatch metrik	Deskripsi	Unit
ChangeLogBytesUsed	Jumlah penyimpanan yang digunakan oleh binlog yang ditingkatkan.	Byte
ChangeLogReadIOP	Jumlah operasi I/O baca yang dilakukan di binlog yang ditingkatkan dalam interval 5 menit.	Hitungan per 5 menit

CloudWatch metrik	Deskripsi	Unit
ChangeLogWriteIOP	Jumlah operasi I/O disk tulis yang dilakukan di binlog yang ditingkatkan dalam interval 5 menit.	Hitungan per 5 menit

Batasan binlog yang ditingkatkan

Batasan berikut berlaku untuk klaster Amazon Aurora DB saat binlog yang ditingkatkan diaktifkan.

- Binlog yang ditingkatkan hanya didukung di Aurora MySQL version3.03.1 dan yang lebih tinggi.
- File binlog yang ditingkatkan yang ditulis pada klaster DB primer tidak disalin ke klaster DB yang dikloning atau dipulihkan.
- Saat digunakan dengan basis data global Amazon Aurora, file binlog yang ditingkatkan untuk klaster DB primer tidak direplikasi ke klaster DB sekunder. Oleh karena itu, setelah proses failover, data binlog historis tidak tersedia di klaster DB primer yang baru.
- Parameter konfigurasi binlog berikut diabaikan:
  - `binlog_group_commit_sync_delay`
  - `binlog_group_commit_sync_no_delay_count`
  - `binlog_max_flush_queue_time`
- Anda tidak dapat menghapus atau mengganti nama tabel yang rusak dalam basis data. Untuk menjatuhkan tabel ini, Anda dapat menghubungi AWS Support.
- Cache I/O binlog dinonaktifkan saat binlog yang ditingkatkan diaktifkan. Untuk informasi selengkapnya, lihat Mengoptimalkan replikasi log biner.

> **Note**
>
> Binlog yang ditingkatkan memberikan peningkatan performa baca yang serupa dengan cache I/O binlog dan peningkatan performa tulis yang lebih baik.

- Fitur pelacakan mundur tidak didukung. Binlog yang ditingkatkan tidak dapat diaktifkan di klaster DB dalam kondisi berikut:
  - Klaster DB dengan fitur pelacakan mundur saat ini diaktifkan.
  - Cluster DB tempat fitur backtrack sebelumnya diaktifkan, tetapi sekarang dinonaktifkan.

- Klaster DB dipulihkan dari klaster DB sumber atau snapshot dengan fitur pelacakan mundur diaktifkan.

Menggunakan replikasi berbasis GTID untuk Amazon Aurora MySQL

Berikut ini, Anda dapat mempelajari cara menggunakan pengidentifikasi transaksi global (GTID) dengan replikasi log biner (binlog) di antara klaster Aurora MySQL dan sumber eksternal.

> **Note**
>
> Untuk Aurora, Anda hanya dapat menggunakan fitur ini dengan klaster Aurora MySQL yang menggunakan replikasi binlog ke atau dari basis data MySQL eksternal. Basis data lain mungkin berupa instans Amazon RDS MySQL, basis data MySQL on-premise, atau klaster DB Aurora di Wilayah AWS yang berbeda. Untuk mempelajari cara mengonfigurasi jenis replikasi semacam itu, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner).

Jika Anda menggunakan replikasi binlog dan tidak familier dengan replikasi berbasis GTID dengan MySQL, ketahui latar belakangnya di Replication with global transaction identifiers dalam dokumentasi MySQL.

Replikasi berbasis GTID didukung untuk Aurora MySQL versi 2 dan 3.

Topik

- Ikhtisar pengidentifikasi transaksi global (GTID)
- Parameter untuk replikasi berbasis GTID
- Mengonfigurasi replikasi berbasis GTID untuk klaster Aurora MySQL
- Menonaktifkan replikasi berbasis GTID untuk klaster DB Aurora MySQL

Ikhtisar pengidentifikasi transaksi global (GTID)

Pengidentifikasi transaksi global (GTID) adalah pengidentifikasi unik yang dibuat untuk transaksi MySQL yang dilakukan. Anda dapat menggunakan GTID agar pemecahan masalah pada replikasi binlog bisa dilakukan dengan lebih mudah dan sederhana.

> **Note**
>
> Saat Aurora menyinkronkan data antar-instans DB dalam sebuah klaster, mekanisme replikasi tersebut tidak melibatkan log biner (binlog). Untuk Aurora MySQL, replikasi berbasis GTID hanya berlaku saat Anda juga menggunakan replikasi binlog untuk mereplikasi ke dalam atau keluar klaster DB Aurora MySQL dari basis data eksternal yang kompatibel dengan MySQL.

MySQL menggunakan dua jenis transaksi untuk replikasi binlog:

- Transaksi GTID – Transaksi yang diidentifikasi oleh GTID.
- Transaksi anonim – Transaksi yang tidak memiliki GTID.

Dalam konfigurasi replikasi, GTID bersifat unik di semua instans DB. GTID menyederhanakan konfigurasi replikasi karena saat Anda menggunakannya, Anda tidak harus merujuk ke posisi file log. GTID juga mempermudah pelacakan transaksi yang direplikasi dan menentukan apakah instans sumber dan replika konsisten.

Replikasi berbasis GTID biasanya digunakan dengan Aurora saat mereplikasi dari basis data yang kompatibel dengan MySQL eksternal ke dalam klaster Aurora. Anda dapat menyiapkan konfigurasi replikasi ini sebagai bagian migrasi dari basis data on-premise atau basis data Amazon RDS ke Aurora MySQL. Jika basis data eksternal sudah menggunakan GTID, mengaktifkan replikasi berbasis GTID untuk klaster Aurora akan menyederhanakan proses replikasi.

Anda mengonfigurasi replikasi berbasis GTID untuk klaster Aurora MySQL dengan terlebih dahulu menetapkan parameter konfigurasi yang relevan dalam grup parameter klaster DB. Kemudian, kaitkan grup parameter tersebut dengan klaster.

Parameter untuk replikasi berbasis GTID

Gunakan parameter berikut untuk mengonfigurasi replikasi berbasis GTID.

Parameter	Nilai valid	Deskripsi
`gtid_mode`	`OFF`, `OFF_PERMISSIVE` , `ON_PERMISSIVE` , `ON`	`OFF` menentukan bahwa transaksi baru adalah transaksi anonim (yaitu, tidak memiliki GTID),

Parameter	Nilai valid	Deskripsi
		dan transaksi harus anonim agar dapat direplikasi. `OFF_PERMISSIVE` menentukan bahwa transaksi baru adalah transaksi anonim, tetapi semua transaksi dapat direplikasi. `ON_PERMISSIVE` menentukan bahwa transaksi baru adalah transaksi GTID, tetapi semua transaksi dapat direplikasi. `ON` menentukan bahwa transaksi baru adalah transaksi GTID, dan transaksi harus berupa transaksi GTID untuk bisa direplikasi.
`enforce_gtid_consistency`	`OFF`, `ON`, `WARN`	`OFF` memperbolehkan transaksi melanggar konsistensi GTID. `ON` mencegah transaksi melanggar konsistensi GTID. `WARN` memperbolehkan transaksi melanggar konsistensi GTID, tetapi menampilkan peringatan apabila terjadi pelanggaran.

> **Note**
>
> Dalam AWS Management Console, parameter `gtid_mode` muncul sebagai `gtid-mode`.

Untuk replikasi berbasis GTID, gunakan pengaturan ini pada grup parameter klaster DB untuk klaster DB Aurora MySQL Anda:

- `ON` dan hanya `ON_PERMISSIVE` berlaku untuk replikasi keluar dari klaster Aurora MySQL. Kedua nilai ini menyebabkan klaster DB Aurora Anda menggunakan GTID untuk transaksi yang direplikasi ke basis data eksternal. `ON` mengharuskan basis data eksternal juga menggunakan replikasi

berbasis GTID. `ON_PERMISSIVE` membuat replikasi berbasis GTID bersifat opsional di basis data eksternal.

- `OFF_PERMISSIVE`, jika diatur, artinya klaster DB Aurora Anda dapat menerima replikasi masuk dari basis data eksternal. Hal ini dapat dilakukan terlepas dari apakah basis data eksternal tersebut menggunakan replikasi berbasis GTID atau tidak.
- `OFF`, jika diatur, artinya klaster DB Aurora Anda hanya dapat menerima replikasi masuk dari basis data eksternal yang tidak menggunakan replikasi berbasis GTID.

> **Tip**
>
> Replikasi masuk adalah skenario replikasi binlog paling umum untuk klaster Aurora MySQL. Untuk replikasi masuk, kami sarankan agar Anda mengatur mode GTID ke `OFF_PERMISSIVE`. Pengaturan tersebut memungkinkan replikasi masuk dari basis data eksternal, terlepas dari pengaturan GTID di sumber replikasinya.

Untuk informasi selengkapnya tentang grup parameter, lihat Bekerja dengan grup parameter.

Mengonfigurasi replikasi berbasis GTID untuk klaster Aurora MySQL

Saat replikasi berbasis GTID diaktifkan untuk klaster DB Aurora MySQL, pengaturan GTID berlaku untuk replikasi binlog ke dalam dan keluar.

Untuk mengaktifkan replikasi berbasis GTID untuk klaster Aurora MySQL

1. Buat atau edit grup parameter klaster DB menggunakan pengaturan parameter berikut:
   - `gtid_mode` – `ON` atau `ON_PERMISSIVE`
   - `enforce_gtid_consistency` – `ON`
2. Kaitkan grup parameter klaster DB dengan klaster Aurora MySQL. Untuk melakukannya, ikuti prosedur dalam Bekerja dengan grup parameter.
3. (Opsional) Tentukan cara menetapkan GTID ke transaksi yang tidak menyertakannya. Untuk melakukannya, panggil prosedur yang tersimpan di mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL versi 3).

Menonaktifkan replikasi berbasis GTID untuk klaster DB Aurora MySQL

Anda dapat menonaktifkan replikasi berbasis GTID untuk klaster DB Aurora MySQL. Dengan begitu, artinya klaster Aurora tidak dapat melakukan replikasi binlog ke dalam atau keluar dengan basis data eksternal yang menggunakan replikasi berbasis GTID.

> **Note**
>
> Dalam prosedur berikut, replika baca berarti target replikasi dalam konfigurasi Aurora dengan replikasi binlog ke atau dari basis data eksternal. Jadi, itu bukan instans DB Aurora Replica hanya-baca. Misalnya, saat sebuah klaster Aurora menerima replikasi masuk dari sebuah sumber eksternal, instans primer Aurora bertindak sebagai replika baca untuk replikasi binlog.

Untuk detail selengkapnya tentang prosedur tersimpan yang disebutkan di bagian ini, lihat Prosedur tersimpan Aurora MySQL.

Untuk menonaktifkan replikasi berbasis GTID untuk klaster DB Aurora MySQL

1. Pada instans primer Aurora, jalankan prosedur berikut.

   ```
   CALL mysql.rds_set_master_auto_position(0); (Aurora MySQL version 2)
   CALL mysql.rds_set_source_auto_position(0); (Aurora MySQL version 3)
   ```

2. Atur ulang `gtid_mode` ke `ON_PERMISSIVE`.

   a. Pastikan grup parameter klaster DB yang terkait dengan klaster Aurora MySQL memiliki yang diatur ke `ON_PERMISSIVE`.

      Untuk informasi selengkapnya tentang cara mengatur parameter konfigurasi menggunakan grup parameter, lihat Bekerja dengan grup parameter.

   b. Mulai ulang klaster DB Aurora MySQL.

3. Atur ulang `gtid_mode` ke `OFF_PERMISSIVE`:

   a. Pastikan grup parameter klaster DB yang terkait dengan klaster Aurora MySQL memiliki yang diatur ke `OFF_PERMISSIVE`.

   b. Mulai ulang klaster DB Aurora MySQL.

4. a. Pada instans primer Aurora, jalankan perintah `SHOW MASTER STATUS`.

Output Anda semestinya serupa dengan yang berikut ini.

```
File                        Position
-----------------------------------
mysql-bin-changelog.000031      107
-----------------------------------
```

Catat file dan posisi dalam output Anda.

b. Pada setiap replika baca, gunakan informasi file dan posisi dari instans sumber dalam langkah sebelumnya untuk menjalankan kueri berikut.

```
SELECT MASTER_POS_WAIT('file', position);
```

Misalnya, jika nama file-nya adalah `mysql-bin-changelog.000031` dan posisinya adalah `107`, jalankan pernyataan berikut.

```
SELECT MASTER_POS_WAIT('mysql-bin-changelog.000031', 107);
```

Jika replika baca melewati posisi yang ditentukan, kueri akan segera ditampilkan. Jika tidak, fungsi akan menunggu. Saat kueri menampilkan semua replika baca, lanjutkan ke langkah berikutnya.

5. Atur ulang parameter GTID untuk menonaktifkan replikasi berbasis GTID:

   a. Pastikan grup parameter klaster DB yang terkait dengan klaster Aurora MySQL memiliki pengaturan parameter berikut ini:

   - `gtid_mode` – `OFF`
   - `enforce_gtid_consistency` – `OFF`

   b. Mulai ulang klaster DB Aurora MySQL.

Mengintegrasikan Amazon Aurora MySQL dengan layanan AWS lainnya

Amazon Aurora MySQL berintegrasi dengan layanan AWS lain sehingga Anda dapat memperluas klaster DB Aurora MySQL Anda untuk menggunakan kemampuan tambahan di AWS Cloud. Klaster DB Aurora MySQL Anda dapat menggunakan layanan AWS untuk melakukan hal berikut:

- Secara sinkron atau asinkron menginvokasi fungsi AWS Lambda menggunakan fungsi native `lambda_sync` atau `lambda_async`. Untuk informasi selengkapnya, lihat Menginvokasi fungsi Lambda dari klaster DB Amazon Aurora MySQL.
- Memuat data dari file teks atau XML yang disimpan dalam bucket Amazon Simple Storage Service (Amazon S3) ke dalam klaster DB Anda menggunakan perintah `LOAD DATA FROM S3` atau `LOAD XML FROM S3`. Untuk informasi selengkapnya, lihat Memuat data ke klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3.
- Simpan data ke berkas teks yang disimpan dalam bucket Amazon S3 dari klaster DB Anda menggunakan perintah `SELECT INTO OUTFILE S3`. Untuk informasi selengkapnya, lihat Menyimpan data dari klaster DB Amazon Aurora MySQL ke dalam file teks di bucket Amazon S3.
- Secara otomatis menambahkan atau menghapus Replika Aurora dengan Application Auto Scaling. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Auto Scaling dengan Replika Aurora.
- Lakukan analisis sentimen dengan Amazon Comprehend, atau berbagai macam algoritma pembelajaran mesin dengan. SageMaker Untuk informasi selengkapnya, lihat Menggunakan machine learning Amazon Aurora.

Aurora mengamankan kemampuan untuk mengakses layanan AWS lain menggunakan AWS Identity and Access Management (IAM). Anda memberikan izin untuk mengakses layanan AWS lainnya dengan membuat peran IAM dengan izin yang diperlukan, lalu mengaitkan peran tersebut dengan klaster DB Anda. Untuk detail dan instruksi tentang cara mengizinkan klaster DB Aurora MySQL Anda untuk mengakses layanan AWS lainnya atas nama Anda, lihat Mengotorisasi Amazon Aurora MySQL untuk mengakses layanan AWS lainnya atas nama Anda.

Mengotorisasi Amazon Aurora MySQL untuk mengakses layanan AWS lainnya atas nama Anda

Agar klaster DB Aurora MySQL Anda dapat mengakses layanan lainnya atas nama Anda, buat dan konfigurasikan peran AWS Identity and Access Management (IAM). Peran ini mengotorisasi pengguna basis data di klaster DB Anda untuk mengakses layanan AWS lainnya. Untuk informasi selengkapnya, lihat Menyiapkan peran IAM untuk mengakses layanan AWS.

Anda juga harus mengonfigurasi klaster DB Aurora Anda untuk mengizinkan koneksi keluar ke layanan AWS target. Untuk informasi selengkapnya, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora MySQL ke layanan AWS lainnya.

Jika Anda melakukannya, pengguna basis data dapat melakukan tindakan ini menggunakan layanan AWS lainnya:

- Secara sinkron atau asinkron menginvokasi fungsi AWS Lambda menggunakan fungsi native `lambda_sync` atau `lambda_async`. Atau, secara asinkron menginvokasi fungsi AWS Lambda menggunakan prosedur `mysql.lambda_async`. Untuk informasi selengkapnya, lihat Menginvokasi fungsi Lambda dengan fungsi native Aurora MySQL.
- Memuat data dari file teks atau XML yang disimpan dalam bucket Amazon S3 ke dalam klaster DB Anda dengan menggunakan pernyataan `LOAD DATA FROM S3` atau `LOAD XML FROM S3`. Untuk informasi selengkapnya, lihat Memuat data ke klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3.
- Menyimpan data dari klaster DB Anda ke dalam file teks yang disimpan dalam bucket Amazon S3 dengan menggunakan pernyataan `SELECT INTO OUTFILE S3`. Untuk informasi selengkapnya, lihat Menyimpan data dari klaster DB Amazon Aurora MySQL ke dalam file teks di bucket Amazon S3.
- Mengekspor data log ke Log Amazon CloudWatch MySQL. Untuk informasi selengkapnya, lihat Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch .
- Secara otomatis menambahkan atau menghapus Replika Aurora dengan Application Auto Scaling. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Auto Scaling dengan Replika Aurora.

Menyiapkan peran IAM untuk mengakses layanan AWS

Untuk mengizinkan klaster DB Aurora Anda mengakses layanan AWS lainnya, lakukan hal berikut:

1. Buat kebijakan IAM yang memberikan izin ke layanan AWS. Untuk informasi selengkapnya, lihat:
   - Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3
   - Membuat kebijakan IAM untuk mengakses sumber daya AWS Lambda
   - Membuat kebijakan IAM untuk mengakses sumber daya Log CloudWatch
   - Membuat kebijakan IAM untuk mengakses sumber daya AWS KMS
2. Buat peran IAM dan lampirkan kebijakan yang Anda buat. Untuk informasi selengkapnya, lihat Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.
3. Kaitkan peran IAM tersebut dengan klaster DB Aurora Anda. Untuk informasi selengkapnya, lihat Mengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL.

Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3

Aurora dapat mengakses sumber daya Amazon S3 untuk memuat data ke atau menyimpan data dari klaster DB Aurora. Namun, Anda harus terlebih dahulu membuat kebijakan IAM yang memberikan izin bucket dan objek yang memungkinkan Aurora mengakses Amazon S3.

Tabel berikut mencantumkan fitur Aurora yang dapat mengakses bucket Amazon S3 atas nama Anda, serta izin bucket dan objek minimum yang diperlukan setiap fitur.

Fitur	Izin bucket	Izin objek
`LOAD DATA FROM S3`	`ListBucket`	`GetObject` `GetObjectVersion`
`LOAD XML FROM S3`	`ListBucket`	`GetObject` `GetObjectVersion`

`SELECT INTO OUTFILE S3`	`ListBucket`	`AbortMultipartUpload`
		`DeleteObject`
		`GetObject`
		`ListMultipartUploadParts`

Fitur	Izin bucket	Izin objek
		`PutObject`

Kebijakan berikut menambahkan izin yang mungkin diperlukan Aurora untuk mengakses bucket Amazon S3 atas nama Anda.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAuroraToExampleBucket",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject",
                "s3:AbortMultipartUpload",
                "s3:ListBucket",
                "s3:DeleteObject",
                "s3:GetObjectVersion",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": [
                "arn:aws:s3:::example-bucket/*",
                "arn:aws:s3:::example-bucket"
            ]
        }
    ]
}
```

> **Note**
>
> Pastikan untuk menyertakan kedua entri tersebut untuk nilai `Resource`. Aurora memerlukan izin pada bucket itu sendiri dan semua objek di dalam bucket.
> Berdasarkan kasus penggunaan Anda, Anda mungkin tidak perlu menambahkan semua izin dalam contoh kebijakan. Selain itu, izin lain mungkin diperlukan. Misalnya, jika bucket Amazon S3 Anda dienkripsi, Anda perlu menambahkan izin `kms:Decrypt`.

Anda dapat menggunakan langkah-langkah berikut untuk membuat kebijakan IAM yang memberikan izin minimum yang diperlukan bagi Aurora untuk mengakses bucket Amazon S3 atas nama Anda. Untuk mengizinkan Aurora mengakses semua bucket Amazon S3 Anda, Anda dapat melewati langkah-langkah ini dan menggunakan kebijakan IAM standar `AmazonS3ReadOnlyAccess` atau `AmazonS3FullAccess` daripada membuatnya sendiri.

Untuk membuat kebijakan IAM untuk memberikan akses ke sumber daya Amazon S3 Anda

1. Buka Konsol Manajemen IAM.
2. Di panel navigasi, pilih Kebijakan.
3. Pilih Buat kebijakan.
4. Pada tab Editor visual, klik Pilih layanan, lalu pilih S3.
5. Untuk Tindakan, pilih Perluas semua, lalu pilih izin bucket dan izin objek yang diperlukan untuk kebijakan IAM.

   Izin objek adalah izin untuk operasi objek di Amazon S3, dan perlu diberikan untuk objek dalam bucket, bukan bucket itu sendiri. Untuk informasi selengkapnya tentang izin untuk operasi objek di Amazon S3, lihat Izin untuk operasi objek.
6. Pilih Sumber Daya, lalu pilih Tambahkan ARN untuk bucket.
7. Di kotak dialog Tambahkan ARN, berikan detail tentang sumber daya Anda, dan pilih Tambahkan.

   Tentukan ke bucket Amazon S3 mana akses akan diizinkan. Misalnya, jika Anda ingin mengizinkan Aurora mengakses bucket Amazon S3 bernama `example-bucket`, maka atur nilai Amazon Resource Name (ARN) ke `arn:aws:s3:::example-bucket`.
8. Jika sumber daya objek dicantumkan, pilih Tambahkan ARN untuk objek.
9. Di kotak dialog Tambahkan ARN, berikan detail tentang sumber daya Anda.

   Untuk bucket Amazon S3, tentukan ke bucket Amazon S3 mana akses akan diizinkan. Untuk objeknya, Anda dapat memilih Apa pun untuk memberikan izin ke objek apa pun di bucket.

> **Note**
>
> Anda dapat mengatur Amazon Resource Name (ARN) ke nilai ARN yang lebih spesifik untuk memungkinkan Aurora hanya mengakses file atau folder tertentu dalam bucket

Amazon S3. Untuk informasi selengkapnya tentang cara menentukan kebijakan akses untuk Amazon S3, lihat Mengelola izin akses ke sumber daya Amazon S3 Anda.

10. (Opsional) Pilih Tambahkan ARN untuk bucket guna menambahkan bucket Amazon S3 lainnya ke kebijakan, dan ulangi langkah sebelumnya untuk bucket.

> **Note**
>
> Anda dapat mengulanginya untuk menambahkan pernyataan izin bucket yang sesuai ke kebijakan Anda untuk setiap bucket Amazon S3 yang Anda ingin agar diakses Aurora. Secara opsional, Anda juga dapat memberikan akses ke semua bucket dan objek di Amazon S3.

11. Pilih Tinjau kebijakan.
12. Untuk Nama, masukkan nama untuk kebijakan IAM Anda, misalnya `AllowAuroraToExampleBucket`. Anda menggunakan nama ini saat membuat peran IAM untuk dikaitkan dengan klaster DB Aurora Anda. Anda juga dapat menambahkan nilai Deskripsi opsional.
13. Pilih Buat kebijakan.
14. Selesaikan langkah-langkah dalam Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.

Membuat kebijakan IAM untuk mengakses sumber daya AWS Lambda

Anda dapat membuat kebijakan IAM yang memberikan izin minimum yang diperlukan bagi Aurora untuk menginvokasi fungsi AWS Lambda atas nama Anda.

Kebijakan berikut menambahkan izin yang diperlukan oleh Aurora untuk menginvokasi fungsi AWS Lambda atas nama Anda.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAuroraToExampleFunction",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource":
 "arn:aws:lambda:<region>:<123456789012>:function:<example_function>"
```

```
        }
    ]
}
```

Anda dapat menggunakan langkah-langkah berikut untuk membuat kebijakan IAM yang memberikan izin minimum yang diperlukan bagi Aurora untuk menginvokasi fungsi AWS Lambda atas nama Anda. Untuk memungkinkan Aurora menginvokasi semua fungsi AWS Lambda Anda, Anda dapat melewati langkah-langkah ini dan menggunakan kebijakan `AWSLambdaRole` standar daripada membuatnya sendiri.

Untuk membuat kebijakan IAM untuk mengizinkan invokasi fungsi AWS Lambda Anda

1. Buka konsol IAM.
2. Di panel navigasi, pilih Kebijakan.
3. Pilih Buat kebijakan.
4. Pada tab Editor visual, klik Pilih layanan, lalu pilih Lambda.
5. Untuk Tindakan, pilih Perluas semua, lalu pilih izin AWS Lambda yang diperlukan untuk kebijakan IAM.

   Pastikan bahwa `InvokeFunction` dipilih. Ini adalah izin minimum yang diperlukan untuk memungkinkan Amazon Aurora menginvokasi fungsi AWS Lambda.
6. Pilih Sumber Daya dan pilih Tambahkan ARN untuk fungsi.
7. Di kotak dialog Tambahkan ARN, berikan detail tentang sumber daya Anda.

   Tentukan ke fungsi Lambda mana akses akan diizinkan. Misalnya, jika Anda ingin mengizinkan Aurora mengakses fungsi Lambda bernama `example_function`, atur nilai ARN ke `arn:aws:lambda:::function:example_function`.

   Untuk informasi selengkapnya tentang cara menentukan kebijakan akses untuk AWS Lambda, lihat Autentikasi dan kontrol akses untuk AWS Lambda.
8. Secara opsional, pilih Tambahkan izin tambahan untuk menambahkan fungsi AWS Lambda lain ke kebijakan, lalu ulangi langkah sebelumnya untuk fungsi tersebut.

> **Note**
>
> Anda dapat mengulanginya untuk menambahkan pernyataan izin fungsi yang sesuai ke kebijakan Anda untuk setiap fungsi AWS Lambda yang Anda ingin agar diakses Aurora.

9. Pilih Tinjau kebijakan.
10. Atur Nama untuk nama kebijakan IAM Anda, misalnya `AllowAuroraToExampleFunction`. Anda menggunakan nama ini saat membuat peran IAM untuk dikaitkan dengan klaster DB Aurora Anda. Anda juga dapat menambahkan nilai Deskripsi opsional.
11. Pilih Buat kebijakan.
12. Selesaikan langkah-langkah dalam Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.

Membuat kebijakan IAM untuk mengakses sumber daya Log CloudWatch

Aurora dapat mengakses Log CloudWatch untuk mengekspor data log audit dari klaster DB Aurora. Namun, Anda harus terlebih dahulu membuat kebijakan IAM yang menyediakan izin grup log dan log stream yang memungkinkan Aurora mengakses Log CloudWatch.

Kebijakan berikut menambahkan izin yang diperlukan Aurora untuk mengakses Log Amazon CloudWatch atas nama Anda, serta izin minimum yang diperlukan untuk membuat grup log dan mengekspor data.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:GetLogEvents",
                "logs:PutLogEvents"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*:log-stream:*"
        },
        {
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogGroupsAndStreams",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutRetentionPolicy",
                "logs:CreateLogGroup"
            ],
            "Resource": "arn:aws:logs:*:*:log-group:/aws/rds/*"
```

```
        }
    ]
}
```

Anda dapat mengubah ARN dalam kebijakan untuk membatasi akses ke Wilayah AWS dan akun tertentu.

Anda dapat menggunakan langkah-langkah berikut untuk membuat kebijakan IAM yang memberikan izin minimum yang diperlukan Aurora untuk mengakses Log CloudWatch atas nama Anda. Untuk memberi Aurora akses penuh ke Log CloudWatch, Anda dapat melewati langkah-langkah ini dan menggunakan kebijakan IAM `CloudWatchLogsFullAccess` standar daripada membuatnya sendiri. Untuk informasi selengkapnya, lihat Menggunakan kebijakan berbasis identitas (kebijakan IAM) untuk Log CloudWatch dalam Panduan Pengguna Amazon CloudWatch.

Untuk membuat kebijakan IAM untuk memberikan akses ke sumber daya Log CloudWatch Anda

1. Buka konsol IAM.
2. Di panel navigasi, pilih Kebijakan.
3. Pilih Buat kebijakan.
4. Di tab Editor visual, klik Pilih layanan, lalu pilih Log CloudWatch.
5. Untuk Tindakan, pilih Perluas semua (di sebelah kanan), lalu pilih izin Log Amazon CloudWatch yang diperlukan untuk kebijakan IAM.

   Pastikan izin berikut dipilih:

   - `CreateLogGroup`
   - `CreateLogStream`
   - `DescribeLogStreams`
   - `GetLogEvents`
   - `PutLogEvents`
   - `PutRetentionPolicy`
6. Pilih Sumber Daya dan pilih Tambahkan ARN untuk grup log.
7. Di kotak dialog Tambahkan ARN, masukkan nilai berikut:

   - Wilayah – Wilayah AWS atau *
   - Akun – Nomor akun atau *

- Nama Grup Log – `/aws/rds/*`

8. Di kotak dialog Tambahkan ARN, pilih Tambahkan.
9. Pilih Tambahkan ARN untuk log stream.
10. Di kotak dialog Tambahkan ARN, masukkan nilai berikut:
    - Wilayah – Wilayah AWS atau *
    - Akun – Nomor akun atau *
    - Nama Grup Log – `/aws/rds/*`
    - Nama Log Stream – *
11. Di kotak dialog Tambahkan ARN, pilih Tambahkan.
12. Pilih Tinjau kebijakan.
13. Atur Nama untuk nama kebijakan IAM Anda, misalnya `AmazonRDSCloudWatchLogs`. Anda menggunakan nama ini saat membuat peran IAM untuk dikaitkan dengan klaster DB Aurora Anda. Anda juga dapat menambahkan nilai Deskripsi opsional.
14. Pilih Buat kebijakan.
15. Selesaikan langkah-langkah dalam Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.

Membuat kebijakan IAM untuk mengakses sumber daya AWS KMS

Aurora dapat mengakses AWS KMS keys yang digunakan untuk mengenkripsi cadangan basis datanya. Namun, Anda harus terlebih dahulu membuat kebijakan IAM yang memberikan izin yang memungkinkan Aurora mengakses kunci KMS.

Kebijakan berikut menambahkan izin yang diperlukan Aurora untuk mengakses kunci KMS atas nama Anda.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:<region>:<123456789012>:key/<key-ID>"
```

```
        }
    ]
}
```

Anda dapat menggunakan langkah-langkah berikut untuk membuat kebijakan IAM yang memberikan izin minimum yang diperlukan Aurora untuk mengakses kunci KMS atas nama Anda.

Untuk membuat kebijakan IAM untuk memberikan akses ke kunci KMS Anda

1. Buka konsol IAM.
2. Di panel navigasi, pilih Kebijakan.
3. Pilih Buat kebijakan.
4. Di tab Editor visual, klik Pilih layanan, lalu pilih KMS.
5. Untuk Tindakan, pilih Tulis, lalu pilih Dekripsi.
6. Pilih Sumber Daya, lalu pilih Tambahkan ARN.
7. Di kotak dialog Tambahkan ARN, masukkan nilai berikut:
   - Wilayah – Ketik Wilayah AWS, seperti `us-west-2`.
   - Akun – Ketik nomor akun pengguna.
   - Nama Log Stream – Ketik pengidentifikasi kunci KMS.
8. Di kotak dialog Tambahkan ARN, pilih Tambahkan.
9. Pilih Tinjau kebijakan.
10. Atur Nama untuk nama kebijakan IAM Anda, misalnya `AmazonRDSKMSKey`. Anda menggunakan nama ini saat membuat peran IAM untuk dikaitkan dengan klaster DB Aurora Anda. Anda juga dapat menambahkan nilai Deskripsi opsional.
11. Pilih Buat kebijakan.
12. Selesaikan langkah-langkah dalam Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.

Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS

Setelah membuat kebijakan IAM untuk mengizinkan Aurora mengakses sumber daya AWS, Anda harus membuat peran IAM dan melampirkan kebijakan IAM ke peran IAM baru.

Agar peran IAM mengizinkan klaster Amazon RDS Anda berkomunikasi dengan layanan AWS lainnya atas nama Anda, lakukan langkah-langkah berikut.

Agar peran IAM mengizinkan Amazon RDS mengakses layanan AWS

1. Buka konsol IAM.
2. Di panel navigasi, pilih Peran.
3. Pilih Buat peran.
4. Di bagian Layanan AWS, pilih RDS.
5. Di bagian Pilih kasus penggunaan Anda, pilih RDS – Tambahkan Peran ke Basis Data.
6. Pilih Berikutnya.
7. Di halaman Kebijakan izin, masukkan nama kebijakan Anda di bidang Cari.
8. Saat muncul di daftar, pilih kebijakan yang Anda tentukan sebelumnya menggunakan petunjuk dalam salah satu bagian berikut:
   - Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3
   - Membuat kebijakan IAM untuk mengakses sumber daya AWS Lambda
   - Membuat kebijakan IAM untuk mengakses sumber daya Log CloudWatch
   - Membuat kebijakan IAM untuk mengakses sumber daya AWS KMS
9. Pilih Berikutnya.
10. Dalam Nama peran, masukkan nama untuk peran IAM Anda, misalnya `RDSLoadFromS3`. Anda juga dapat menambahkan nilai Deskripsi opsional.
11. Pilih Buat Peran.
12. Selesaikan langkah-langkah dalam Mengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL.

Mengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL

Untuk mengizinkan pengguna basis data di klaster DB Amazon Aurora untuk mengakses layanan AWS lainnya, Anda perlu mengaitkan peran IAM yang Anda buat di Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS dengan klaster DB tersebut. Anda juga dapat meminta AWS untuk membuat peran IAM baru dengan mengaitkan layanan secara langsung.

> **Note**
>
> Anda tidak dapat mengaitkan peran IAM dengan klaster DB Aurora Serverless v1. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Serverless v1.

Anda dapat mengaitkan peran IAM dengan klaster DB Aurora Serverless v2.

Untuk mengaitkan peran IAM dengan klaster DB, lakukan dua hal berikut:

1. Tambahkan peran ke daftar peran terkait untuk klaster DB dengan menggunakan konsol RDS, perintah add-role-to-db-cluster AWS CLI, atau operasi API RDS AddRoleToDBCluster.

   Anda dapat menambahkan maksimum lima peran IAM untuk setiap klaster DB Aurora.
2. Atur parameter tingkat klaster untuk layanan AWS terkait ke ARN untuk peran IAM terkait.

   Tabel berikut menjelaskan nama parameter tingkat klaster untuk peran IAM yang digunakan untuk mengakses layanan AWS lainnya.

Parameter tingkat klaster	Deskripsi
`aws_default_lambda_role`	Digunakan saat menginvokasi fungsi Lambda dari klaster DB Anda.
`aws_default_logs_role`	Parameter ini tidak lagi diperlukan untuk mengekspor data log dari klaster DB Anda ke Log Amazon CloudWatch. Aurora MySQL sekarang menggunakan peran terkait layanan untuk izin yang diperlukan. Untuk mengetahui informasi selengkapnya tentang peran terkait layanan, lihat Menggunakan peran tertaut layanan untuk Amazon Aurora.
`aws_default_s3_role`	Digunakan saat menginvokasi pernyataan `LOAD DATA FROM S3`, `LOAD XML FROM S3`, atau `SELECT INTO OUTFILE S3` dari klaster DB Anda. Di Aurora MySQL versi 2, peran IAM yang ditentukan dalam parameter ini akan digunakan jika peran IAM tidak ditentukan untuk `aurora_load_from_s3_role` atau

Parameter tingkat klaster	Deskripsi	
	`aurora_select_into_s3_role` untuk pernyataan yang sesuai. Di Aurora MySQL versi 3, peran IAM yang ditentukan untuk parameter ini selalu digunakan.	
`aurora_load_from_s3_role`	Digunakan saat menginvokasi pernyataan `LOAD DATA FROM S3` atau `LOAD XML FROM S3` dari klaster DB Anda. Jika peran IAM tidak ditentukan untuk parameter ini, peran IAM yang ditentukan dalam `aws_default_s3_role` akan digunakan. Di Aurora MySQL versi 3, parameter ini tidak tersedia.	
`aurora_select_into_s3_role`	Digunakan saat menginvokasi pernyataan `SELECT INTO OUTFILE S3` dari klaster DB Anda. Jika peran IAM tidak ditentukan untuk parameter ini, peran IAM yang ditentukan dalam `aws_default_s3_role` akan digunakan. Di Aurora MySQL versi 3, parameter ini tidak tersedia.	

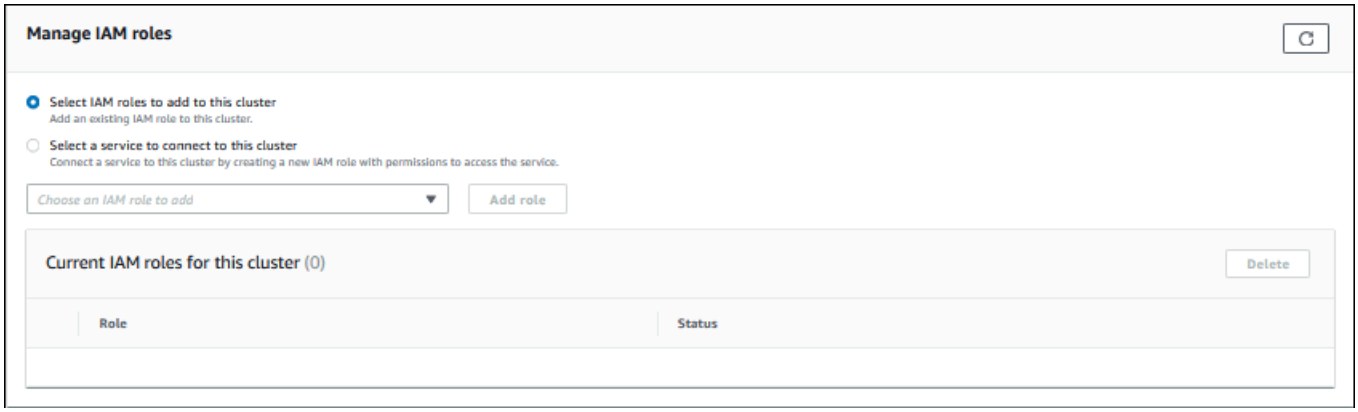
Untuk mengaitkan peran IAM untuk mengizinkan klaster Amazon RDS Anda berkomunikasi dengan layanan AWS lain atas nama Anda, lakukan langkah-langkah berikut.

Konsol

Untuk mengaitkan peran IAM dengan klaster DB Aurora menggunakan konsol

1. Buka konsol RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.

3. Pilih nama klaster DB Aurora yang ingin Anda kaitkan dengan peran IAM untuk menampilkan detailnya.
4. Pada tab Konektivitas & keamanan, di bagian Kelola peran IAM, lakukan salah satu hal berikut:
   - Pilih peran IAM untuk ditambahkan ke klaster ini (default)
   - Pilih layanan untuk dihubungkan ke klaster ini



5. Untuk menggunakan peran IAM yang ada, pilih peran IAM tersebut dari menu, lalu pilih Tambahkan peran.

   Jika penambahan peran berhasil, statusnya menunjukkan `Pending`, lalu `Available`.
6. Untuk menghubungkan layanan secara langsung:
   a. Klik Pilih layanan untuk dihubungkan ke klaster ini.
   b. Pilih layanannya dari menu, lalu pilih Hubungkan layanan.
   c. Untuk Hubungkan klaster ke ***`Nama Layanan`***, masukkan Amazon Resource Name (ARN) yang akan digunakan untuk terhubung ke layanan tersebut, lalu pilih Hubungkan layanan.

   AWS membuat peran IAM baru untuk menghubungkan ke layanan. Statusnya menunjukkan `Pending`, lalu `Available`.
7. (Opsional) Untuk berhenti mengaitkan peran IAM dengan klaster DB dan menghapus izin terkait, pilih peran dan pilih Hapus.

Untuk mengatur parameter tingkat klaster untuk peran IAM terkait

1. Di konsol RDS, pilih Grup parameter di panel navigasi.

2. Jika Anda sudah menggunakan grup parameter DB kustom, Anda dapat memilih grup tersebut untuk digunakan daripada membuat grup parameter klaster DB baru. Jika Anda menggunakan grup parameter klaster DB default, buat grup parameter klaster DB baru, seperti yang dijelaskan di langkah-langkah berikut:

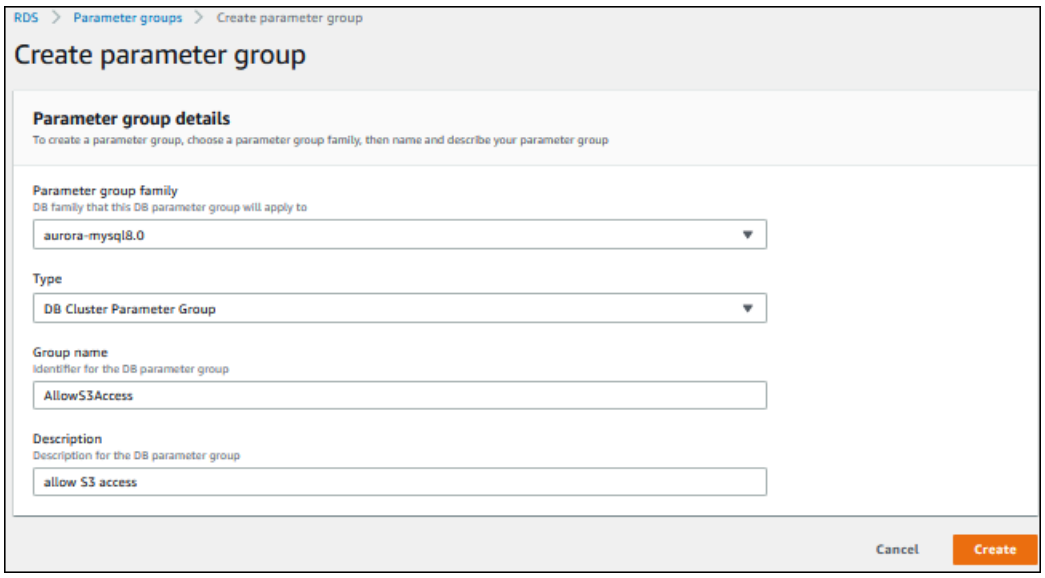
   a. Pilih Buat grup parameter.

   b. Untuk Rangkaian grup parameter, pilih `aurora-mysql8.0` untuk klaster DB yang kompatibel dengan Aurora MySQL 8.0, atau `aurora-mysql5.7` untuk klaster DB yang kompatibel dengan Aurora MySQL 5.7.

   c. Untuk Jenis, pilih Grup Parameter Klaster DB.

   d. Untuk Nama grup, ketik nama grup parameter klaster DB baru Anda.

   e. Untuk Deskripsi, ketik deskripsi untuk grup parameter klaster DB baru Anda.



   f. Pilih Buat.

3. Pada halaman Grup parameter, pilih grup parameter klaster DB Anda dan pilih Edit untuk Tindakan grup parameter.

4. Tetapkan parameter tingkat klaster yang sesuai ke nilai ARN milik peran IAM terkait.

   Misalnya, Anda hanya dapat mengatur parameter `aws_default_s3_role` ke `arn:aws:iam::123456789012:role/AllowS3Access`.

5. Pilih Simpan perubahan.

6. Untuk mengubah grup parameter klaster DB untuk klaster DB Anda, selesaikan langkah-langkah berikut:

a. Pilih Basis data, lalu pilih klaster DB Aurora Anda.

b. Pilih Ubah.

c. Gulir ke Opsi basis data dan tetapkan Grup parameter klaster DB ke grup parameter klaster DB.

d. Pilih Lanjutkan.

e. Verifikasi perubahan Anda, lalu pilih Terapkan segera.

f. Pilih Ubah klaster.

g. Pilih Basis data, lalu pilih instans primer untuk klaster DB Anda.

h. Untuk Tindakan, pilih Boot ulang.

   Saat instans di-boot ulang, peran IAM Anda dikaitkan dengan klaster DB Anda.

   Untuk informasi selengkapnya tentang grup parameter klaster, lihat Parameter konfigurasi Aurora MySQL.

CLI

Untuk mengaitkan peran IAM dengan klaster DB menggunakan AWS CLI

1. Panggil perintah `add-role-to-db-cluster` dari AWS CLI untuk menambahkan ARN untuk peran IAM Anda ke klaster DB, seperti yang ditunjukkan berikut.

```
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraS3Role
PROMPT> aws rds add-role-to-db-cluster --db-cluster-identifier my-cluster --role-
arn arn:aws:iam::123456789012:role/AllowAuroraLambdaRole
```

2. Jika Anda menggunakan grup parameter klaster DB default, buat grup parameter klaster DB baru. Jika Anda sudah menggunakan grup parameter DB kustom, Anda dapat menggunakan grup tersebut untuk digunakan daripada membuat grup parameter klaster DB baru.

   Untuk membuat grup parameter klaster DB baru, panggil perintah `create-db-cluster-parameter-group` dari AWS CLI, seperti yang ditunjukkan berikut.

```
PROMPT> aws rds create-db-cluster-parameter-group  --db-cluster-parameter-group-
name AllowAWSAccess \
```

```
    --db-parameter-group-family aurora5.7 --description "Allow access to Amazon S3
 and AWS Lambda"
```

Untuk klaster DB yang kompatibel dengan Aurora MySQL 5.7, tentukan `aurora-mysql5.7` untuk `--db-parameter-group-family`. Untuk klaster DB yang kompatibel dengan Aurora MySQL 8.0, tentukan `aurora-mysql8.0` untuk `--db-parameter-group-family`.

3. Tetapkan parameter atau parameter tingkat klaster yang sesuai dan nilai ARN peran IAM terkait di grup parameter klaster DB Anda, seperti yang ditunjukkan berikut.

```
PROMPT> aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name
 AllowAWSAccess \
    --parameters
 "ParameterName=aws_default_s3_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraS3Role,method=pending-reboot" \
    --parameters
 "ParameterName=aws_default_lambda_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraLambdaRole,method=pending-reboot"
```

4. Ubah klaster DB untuk menggunakan grup parameter klaster DB baru lalu boot ulang klaster, seperti yang ditunjukkan berikut.

```
PROMPT> aws rds modify-db-cluster --db-cluster-identifier my-cluster --db-cluster-
parameter-group-name AllowAWSAccess
PROMPT> aws rds reboot-db-instance --db-instance-identifier my-cluster-primary
```

Saat instans di-boot ulang, peran IAM Anda akan dikaitkan dengan klaster DB Anda.

Untuk informasi selengkapnya tentang grup parameter klaster, lihat Parameter konfigurasi Aurora MySQL.

Mengaktifkan komunikasi jaringan dari Amazon Aurora MySQL ke layanan AWS lainnya

Untuk menggunakan layanan AWS tertentu lainnya dengan Amazon Aurora, konfigurasi jaringan klaster DB Aurora Anda harus mengizinkan koneksi keluar ke titik akhir untuk layanan tersebut. Operasi berikut memerlukan konfigurasi jaringan ini.

- Menginvokasi fungsi AWS Lambda. Untuk mempelajari tentang fitur ini, lihat Menginvokasi fungsi Lambda dengan fungsi native Aurora MySQL.

- Mengakses file dari Amazon S3. Untuk mempelajari tentang fitur ini, lihat Memuat data ke klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3 dan Menyimpan data dari klaster DB Amazon Aurora MySQL ke dalam file teks di bucket Amazon S3.
- Mengakses titik akhir AWS KMS. Akses AWS KMS diperlukan untuk menggunakan stream aktivitas basis data dengan Aurora MySQL. Untuk mempelajari tentang fitur ini, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.
- Mengakses titik akhir SageMaker. Akses SageMaker diperlukan untuk menggunakan machine learning SageMaker dengan Aurora MySQL. Untuk mempelajari tentang fitur ini, lihat Menggunakan machine learning Amazon Aurora dengan Aurora MySQL.

Aurora menampilkan pesan kesalahan berikut jika tidak dapat terhubung ke titik akhir layanan.

```
ERROR 1871 (HY000): S3 API returned error: Network Connection
```

```
ERROR 1873 (HY000): Lambda API returned error: Network Connection. Unable to connect to
 endpoint
```

```
ERROR 1815 (HY000): Internal error: Unable to initialize S3Stream
```

Untuk stream aktivitas basis data yang menggunakan Aurora MySQL, stream aktivitas berhenti berfungsi jika klaster DB tidak dapat mengakses titik akhir AWS KMS. Aurora memberi tahu Anda tentang masalah ini menggunakan Peristiwa RDS.

Jika Anda menemukan pesan ini saat menggunakan layanan AWS terkait, periksa apakah klaster DB Aurora Anda bersifat publik atau privat. Jika klaster DB Aurora Anda bersifat privat, Anda harus mengonfigurasinya untuk mengizinkan koneksi.

Agar menjadi bersifat publik, klaster DB Aurora harus ditandai sebagai dapat diakses publik. Hal ini ditunjukkan dengan opsi Dapat Diakses Publik yang diatur ke Ya jika Anda melihat detail untuk klaster DB tersebut di AWS Management Console. Klaster DB ini juga harus berada di subnet publik Amazon VPC. Untuk informasi selengkapnya tentang instans DB yang dapat diakses publik, lihat Bekerja dengan instans DB dalam VPC. Untuk informasi selengkapnya tentang subnet Amazon VPC publik, lihat VPC dan subnet Anda.

Jika klaster DB Aurora Anda tidak dapat diakses publik dan berada di subnet publik VPC, berarti klaster DB ini bersifat privat. Anda mungkin memiliki klaster DB yang bersifat privat dan

ingin menggunakan salah satu fitur yang memerlukan konfigurasi jaringan ini. Jika demikian, konfigurasikan klaster agar dapat terhubung ke alamat Internet melalui Network Address Translation (NAT). Sebagai alternatif untuk Amazon S3, Amazon SageMaker, dan AWS Lambda, Anda dapat mengonfigurasi VPC agar titik akhir VPC untuk layanan lain dikaitkan dengan tabel rute klaster DB. Untuk detailnya, lihat Bekerja dengan instans DB dalam VPC. Untuk informasi selengkapnya tentang konfigurasi NAT di VPC Anda, lihat Gateway NAT. Untuk informasi selengkapnya tentang mengonfigurasi titik akhir VPC, lihat Titik akhir VPC. Anda juga dapat membuat titik akhir gateway S3 untuk mengakses bucket S3 Anda. Untuk informasi selengkapnya, lihat Titik akhir gateway untuk Amazon S3.

Anda mungkin juga harus membuka port efemeral untuk daftar kontrol akses (ACL) jaringan Anda dalam aturan keluar untuk grup keamanan VPC Anda. Untuk informasi selengkapnya tentang port efemeral untuk ACL jaringan, lihat Port Efemeral dalam Panduan Pengguna Amazon Virtual Private Cloud.

Topik terkait

- Mengintegrasikan Aurora dengan layanan AWS lainnya
- Mengelola klaster DB Amazon Aurora

Memuat data ke klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3

Anda dapat menggunakan pernyataan `LOAD DATA FROM S3` atau `LOAD XML FROM S3` untuk memuat data dari file yang tersimpan di bucket Amazon S3.

> **Note**
>
> Memuat data ke dalam tabel dari file teks tidak didukung untuk Aurora Serverless v1. Hal ini didukung untuk Aurora Serverless v2.

Memberi Aurora akses ke Amazon S3

Sebelum Anda dapat memuat data dari bucket Amazon S3, Anda harus terlebih dahulu memberi klaster DB Aurora MySQL Anda izin untuk mengakses Amazon S3.

Untuk memberi Aurora MySQL akses ke Amazon S3

1. Buat kebijakan AWS Identity and Access Management (IAM) yang memberikan izin bucket dan objek yang memungkinkan klaster DB Aurora MySQL Anda mengakses Amazon S3. Untuk petunjuk, lihat Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3.

   > **Note**
   >
   > Di Aurora MySQL versi 3.05 dan lebih tinggi, Anda dapat memuat objek yang dienkripsi menggunakan AWS KMS keys terkelola pelanggan. Untuk melakukannya, sertakan izin `kms:Decrypt` dalam kebijakan IAM Anda. Untuk informasi selengkapnya, lihat Membuat kebijakan IAM untuk mengakses sumber daya AWS KMS.
   > Anda tidak memerlukan izin ini untuk memuat objek yang dienkripsi menggunakan Kunci yang dikelola AWS atau kunci terkelola Amazon S3 (SSE-S3).

2. Buat peran IAM, lalu lampirkan kebijakan IAM yang Anda buat di Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3 ke peran IAM baru. Untuk petunjuk, lihat Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.
3. Pastikan klaster DB menggunakan grup parameter klaster DB kustom.

   Untuk informasi selengkapnya tentang membuat grup parameter klaster DB kustom, lihat Membuat grup parameter klaster DB.
4. Untuk Aurora MySQL versi 2, atur parameter klaster DB `aurora_load_from_s3_role` atau `aws_default_s3_role` ke Amazon Resource Name (ARN) milik peran IAM yang baru. Jika peran IAM tidak ditentukan untuk `aurora_load_from_s3_role`, Aurora menggunakan peran IAM yang ditentukan dalam `aws_default_s3_role`.

   Untuk Aurora MySQL versi 3, gunakan `aws_default_s3_role`.

   Jika klaster adalah bagian dari basis data global Aurora, atur parameter ini untuk setiap klaster Aurora dalam basis data global. Meskipun hanya klaster primer dalam basis data global Aurora yang dapat memuat data, klaster lain mungkin dipromosikan oleh mekanisme failover dan menjadi klaster primer.

   Untuk informasi selengkapnya tentang parameter klaster DB, lihat Parameter instans DB dan klaster DB Amazon Aurora.
5. Untuk mengizinkan pengguna basis data dalam klaster DB Aurora MySQL untuk mengakses Amazon S3, kaitkan peran yang Anda buat di Membuat peran IAM untuk mengizinkan Amazon

Aurora mengakses layanan AWS dengan klaster DB. Untuk basis data global Aurora, kaitkan peran dengan setiap klaster Aurora di basis data global. Untuk informasi tentang mengaitkan peran IAM dengan klaster DB, lihat Mengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL.

6. Konfigurasikan klaster DB Aurora MySQL Anda untuk memungkinkan koneksi keluar ke Amazon S3. Untuk petunjuk, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora MySQL ke layanan AWS lainnya.

   Jika klaster DB Anda tidak dapat diakses publik dan berada di subnet publik VPC, berarti klaster DB Anda bersifat privat. Anda dapat membuat titik akhir gateway S3 untuk mengakses bucket S3 Anda. Untuk informasi selengkapnya, lihat Titik akhir gateway untuk Amazon S3.

   Untuk basis data global Aurora, aktifkan koneksi keluar untuk setiap klaster Aurora di basis data global.

Memberikan hak akses untuk memuat data di Amazon Aurora MySQL

Pengguna basis data yang mengeluarkan pernyataan `LOAD XML FROM S3` atau `LOAD DATA FROM S3` harus memiliki peran atau hak akses tertentu untuk mengeluarkan pernyataan. Di Aurora MySQL versi 3, Anda memberikan peran `AWS_LOAD_S3_ACCESS`. Di Aurora MySQL versi 2, Anda memberikan hak akses `LOAD FROM S3`. Pengguna administratif untuk klaster DB akan diberi peran atau hak akses yang sesuai secara default. Anda dapat memberikan hak akses kepada pengguna lain dengan menggunakan salah satu pernyataan berikut.

Gunakan pernyataan berikut untuk Aurora MySQL versi 3:

```
GRANT AWS_LOAD_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> Saat Anda menggunakan teknik peran di Aurora MySQL versi 3, Anda juga dapat mengaktifkan peran dengan menggunakan pernyataan `SET ROLE role_name` atau `SET ROLE ALL`. Jika Anda tidak memahami sistem peran MySQL 8.0, Anda dapat mempelajari selengkapnya dalam Model hak akses berbasis peran. Untuk detail selengkapnya, lihat Using roles dalam Panduan Referensi MySQL.
>
> Hal ini hanya berlaku untuk sesi aktif saat ini. Ketika Anda terhubung kembali, Anda harus menjalankan pernyataan `SET ROLE` lagi untuk memberikan hak akses. Untuk informasi selengkapnya, lihat SET ROLE statement dalam Panduan Referensi MySQL.

Anda dapat menggunakan parameter klaster DB `activate_all_roles_on_login` untuk mengaktifkan semua peran secara otomatis saat pengguna terhubung ke instans DB. Ketika parameter ini ditetapkan, Anda tidak perlu memanggil pernyataan SET ROLE secara eksplisit untuk mengaktifkan peran. Untuk informasi selengkapnya, lihat activate_all_roles_on_login dalam Panduan Referensi MySQL.

Gunakan pernyataan berikut untuk Aurora MySQL versi 2:

```
GRANT LOAD FROM S3 ON *.* TO 'user'@'domain-or-ip-address'
```

Peran `AWS_LOAD_S3_ACCESS` dan hak akses `LOAD FROM S3` dikhususkan untuk Amazon Aurora dan tidak tersedia untuk basis data MySQL eksternal atau instans DB RDS for MySQL. Jika Anda telah mengatur replikasi antara klaster DB Aurora sebagai master replikasi dan basis data MySQL sebagai klien replikasi, maka pernyataan `GRANT` untuk peran atau hak akses menyebabkan replikasi berhenti dengan kesalahan. Anda dapat melewati kesalahan ini dengan aman untuk melanjutkan replikasi. Untuk melewati kesalahan pada instans RDS for MySQL, gunakan prosedur mysql_rds_skip_repl_error. Untuk melewati kesalahan pada basis data MySQL eksternal, gunakan variabel sistem slave_skip_errors (Aurora MySQL versi 2) atau variabel sistem replica_skip_errors (Aurora MySQL versi 3).

Menentukan jalur (URI) ke bucket Amazon S3

Sintaksis untuk menentukan jalur (URI) ke file yang tersimpan di bucket Amazon S3 adalah sebagai berikut.

```
s3-region://bucket-name/file-name-or-prefix
```

Jalur tersebut mencakup nilai-nilai berikut:

- `region` (opsional) – Wilayah AWS yang berisi bucket Amazon S3 yang dimuat. Nilai ini bersifat opsional. Jika Anda tidak menentukan nilai `region`, Aurora memuat file Anda dari Amazon S3 di wilayah yang sama dengan klaster DB Anda.
- `bucket-name` – Nama bucket Amazon S3 yang berisi data yang akan dimuat. Awalan objek yang mengidentifikasi jalur folder virtual didukung.
- `file-name-or-prefix` – Nama file teks atau file XML Amazon S3, atau awalan yang mengidentifikasi satu atau beberapa file teks atau file XML yang akan dimuat. Anda juga dapat menentukan file manifes yang mengidentifikasi satu atau beberapa file teks yang akan dimuat.

Untuk informasi selengkapnya tentang menggunakan file manifes untuk memuat file teks dari Amazon S3, lihat Menggunakan manifes untuk menentukan file data yang akan dimuat.

Untuk menyalin URI untuk file dalam bucket S3

1. Masuk ke AWS Management Console dan buka konsol Amazon S3 di https://console.aws.amazon.com/s3/.
2. Di panel navigasi, pilih Bucket, lalu pilih bucket yang URI-nya ingin Anda salin.
3. Pilih awalan atau file yang ingin Anda muat dari S3.
4. Pilih Salin URI S3.

LOAD DATA FROM S3

Anda dapat menggunakan pernyataan `LOAD DATA FROM S3` untuk memuat data dari format file teks apa pun yang didukung oleh pernyataan LOAD DATA INFILE MySQL, seperti data teks yang dipisahkan koma. file terkompresi tidak didukung.

> **Note**
>
> Pastikan klaster DB Aurora MySQL Anda memungkinkan koneksi keluar ke S3. Untuk informasi selengkapnya, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora MySQL ke layanan AWS lainnya.

Sintaksis

```
LOAD DATA [FROM] S3 [FILE | PREFIX | MANIFEST] 'S3-URI'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
```

```
    ]
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var,...)]
    [SET col_name = expr,...]
```

> **Note**
>
> Di Aurora MySQL versi 3.05 dan lebih tinggi, kata kunci `FROM` bersifat opsional.

Parameter

Pernyataan `LOAD DATA FROM S3` menggunakan parameter wajib dan opsional berikut. Anda dapat menemukan detail selengkapnya tentang beberapa parameter ini dalam LOAD DATA Statement dalam dokumentasi MySQL.

FILE | PREFIX | MANIFEST

Mengidentifikasi apakah akan memuat data dari satu file, dari semua file yang cocok dengan awalan tertentu, atau dari semua file dalam manifes yang ditentukan. `FILE` adalah opsi default.

S3-URI

Menentukan URI untuk teks atau file manifes yang akan dimuat, atau awalan Amazon S3 yang akan digunakan. Tentukan URI menggunakan sintaksis yang dijelaskan dalam Menentukan jalur (URI) ke bucket Amazon S3.

REPLACE | IGNORE

Menentukan tindakan yang harus diambil jika baris input memiliki nilai kunci unik yang sama dengan baris yang ada dalam tabel basis data.

- Tentukan `REPLACE` jika Anda ingin baris input menggantikan baris yang ada dalam tabel.
- Tentukan `IGNORE` jika Anda ingin membuang baris input.

INTO TABLE

Mengidentifikasi nama tabel basis data yang akan dimuati dengan baris input.

PARTITION

Mengharuskan semua baris input disisipkan ke partisi yang diidentifikasi oleh daftar nama partisi yang dipisahkan koma yang ditentukan. Jika baris input tidak dapat disisipkan ke salah satu partisi yang ditentukan, maka pernyataan ini akan gagal dan kesalahan akan ditampilkan.

CHARACTER SET

Mengidentifikasi kumpulan karakter data dalam file input.

FIELDS | COLUMNS

Mengidentifikasi cara bidang atau kolom dalam file input dibatasi. Bidang dibatasi tab secara default.

LINES

Mengidentifikasi cara baris dalam file input dibatasi. Baris dibatasi oleh karakter baris baru (`'\n'`) secara default.

IGNORE ***angka*** LINES | ROWS

Menentukan untuk mengabaikan jumlah tertentu baris di awal file input. Misalnya, Anda dapat menggunakan `IGNORE 1 LINES` untuk melewati baris header awal yang berisi nama kolom, atau `IGNORE 2 ROWS` untuk melewati dua baris pertama data dalam file input. Jika Anda juga menggunakan `PREFIX`, `IGNORE` akan melewati jumlah tertentu baris di awal file input pertama.

col_name_or_user_var, ...

– Menentukan daftar yang dipisahkan koma yang berisi satu atau beberapa nama kolom atau variabel pengguna yang mengidentifikasi kolom mana yang akan dimuat berdasarkan nama. Nama variabel pengguna yang digunakan untuk tujuan ini harus cocok dengan nama elemen dari file teks, yang diawali dengan @. Anda dapat menggunakan variabel pengguna untuk menyimpan nilai bidang terkait untuk digunakan kembali nanti.

Misalnya, pernyataan berikut memuat kolom pertama dari file input ke kolom pertama `table1`, dan menetapkan nilai kolom `table_column2` di `table1` ke nilai input kolom kedua dibagi 100.

```
LOAD DATA FROM S3 's3://mybucket/data.txt'
    INTO TABLE table1
    (column1, @var1)
    SET table_column2 = @var1/100;
```

SET

Menentukan daftar operasi penetapan yang dipisahkan koma yang mengatur nilai kolom dalam tabel ke nilai yang tidak disertakan dalam file input.

Misalnya, pernyataan berikut mengatur dua kolom pertama `table1` ke nilai di dua kolom pertama dari file input, lalu mengatur nilai `column3` di `table1` ke stempel waktu saat ini.

```
LOAD DATA FROM S3  's3://mybucket/data.txt'
    INTO TABLE table1
    (column1, column2)
    SET column3 = CURRENT_TIMESTAMP;
```

Anda dapat menggunakan subkueri di sisi kanan penetapan SET. Untuk subkueri yang menampilkan nilai yang akan ditetapkan ke kolom, Anda hanya dapat menggunakan subkueri skalar. Selain itu, Anda tidak dapat menggunakan subkueri untuk memilih dari tabel yang sedang dimuat.

Anda tidak dapat menggunakan kata kunci LOCAL dari pernyataan LOAD DATA FROM S3 jika memuat data dari bucket Amazon S3.

Menggunakan manifes untuk menentukan file data yang akan dimuat

Anda dapat menggunakan pernyataan LOAD DATA FROM S3 dengan kata kunci MANIFEST untuk menentukan file manifes dalam format JSON yang mencantumkan file teks yang akan dimuat ke dalam tabel di klaster DB Anda.

Skema JSON berikut menjelaskan format dan konten file manifes.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "additionalProperties": false,
    "definitions": {},
    "id": "Aurora_LoadFromS3_Manifest",
    "properties": {
        "entries": {
            "additionalItems": false,
            "id": "/properties/entries",
            "items": {
                "additionalProperties": false,
                "id": "/properties/entries/items",
                "properties": {
                    "mandatory": {
                        "default": "false",
                        "id": "/properties/entries/items/properties/mandatory",
                        "type": "boolean"
                    },
                    "url": {
                        "id": "/properties/entries/items/properties/url",
```

```
                        "maxLength": 1024,
                        "minLength": 1,
                        "type": "string"
                    }
                },
                "required": [
                    "url"
                ],
                "type": "object"
            },
            "type": "array",
            "uniqueItems": true
        }
    },
    "required": [
        "entries"
    ],
    "type": "object"
}
```

Setiap `url` dalam manifes harus menentukan URL dengan nama bucket dan jalur objek lengkap untuk file tersebut, bukan hanya awalan. Anda dapat menggunakan manifes untuk memuat file dari bucket yang berbeda-beda, wilayah yang berbeda-beda, atau file yang tidak memiliki awalan yang sama. Jika wilayah tidak ditentukan dalam URL, wilayah klaster DB Aurora target akan digunakan. Contoh berikut menunjukkan file manifes yang memuat empat file dari bucket yang berbeda.

```
{
  "entries": [
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":true
    },
    {
      "url":"s3-us-west-2://aurora-bucket-usw2/2013-10-05-customerdata",
      "mandatory":true
    },
    {
      "url":"s3://aurora-bucket/2013-10-04-customerdata",
      "mandatory":false
    },
    {
      "url":"s3://aurora-bucket/2013-10-05-customerdata"
    }
```

```
  ]
}
```

Flag `mandatory` opsional menentukan apakah `LOAD DATA FROM S3` harus menampilkan kesalahan jika file tidak ditemukan. Flag `mandatory` diatur secara default ke `false`. Terlepas dari cara `mandatory` diatur, `LOAD DATA FROM S3` akan diterminasi jika tidak ada file yang ditemukan.

File manifes dapat memiliki ekstensi. Contoh berikut menjalankan pernyataan `LOAD DATA FROM S3` dengan manifes di contoh sebelumnya, yang bernama **`customer.manifest`**.

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/customer.manifest'
    INTO TABLE CUSTOMER
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, EMAIL);
```

Setelah pernyataan selesai, entri untuk setiap file yang berhasil dimuat akan ditulis ke tabel `aurora_s3_load_history`.

Memverifikasi file yang dimuat menggunakan tabel aurora_s3_load_history

Setiap pernyataan `LOAD DATA FROM S3` yang berhasil akan memperbarui tabel `aurora_s3_load_history` dalam skema `mysql` dengan entri untuk setiap file yang dimuat.

Setelah Anda menjalankan pernyataan `LOAD DATA FROM S3`, Anda dapat memverifikasi file mana yang dimuat dengan mengkueri tabel `aurora_s3_load_history`. Untuk melihat file yang dimuat dari satu iterasi pernyataan, gunakan klausa `WHERE` untuk memfilter catatan di URI Amazon S3 untuk file manifes yang digunakan dalam pernyataan. Jika Anda telah menggunakan file manifes yang sama sebelumnya, filter hasilnya menggunakan bidang `timestamp`.

```
select * from mysql.aurora_s3_load_history where load_prefix = 'S3_URI';
```

Tabel berikut menjelaskan bidang dalam tabel `aurora_s3_load_history`.

Bidang	Deskripsi
`load_prefix`	URI yang ditentukan dalam pernyataan pemuatan. URI ini dapat dipetakan ke salah satu hal berikut:

Bidang	Deskripsi
	• File data tunggal untuk pernyataan `LOAD DATA FROM S3 FILE` • Awalan Amazon S3 yang dipetakan ke beberapa file data untuk pernyataan `LOAD DATA FROM S3 PREFIX` • File manifes tunggal yang berisi nama file yang akan dimuat untuk pernyataan `LOAD DATA FROM S3 MANIFEST`
`file_name`	Nama file yang dimuat ke Aurora dari Amazon S3 menggunakan URI yang diidentifikasi dalam bidang `load_prefix` .
`version_number`	Nomor versi file yang diidentifikasi oleh bidang `file_name` yang dimuat, jika bucket Amazon S3 memiliki nomor versi.
`bytes_loaded`	Ukuran file yang dimuat, dalam byte.
`load_timestamp`	Stempel waktu saat pernyataan `LOAD DATA FROM S3` selesai.

Contoh

Pernyataan berikut memuat data dari bucket Amazon S3 yang berada di wilayah yang sama dengan klaster DB Aurora. Pernyataan tersebut membaca data yang dipisahkan koma di file `customerdata.txt` yang ada di bucket Amazon S3 `dbbucket`, lalu memuat data tersebut ke tabel `store-schema.customer-table`.

```
LOAD DATA FROM S3 's3://dbbucket/customerdata.csv'
    INTO TABLE store-schema.customer-table
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, ADDRESS, EMAIL, PHONE);
```

Pernyataan berikut memuat data dari bucket Amazon S3 yang berada di wilayah yang berbeda dari klaster DB Aurora. Pernyataan tersebut membaca data yang dipisahkan koma dari semua file yang cocok dengan awalan objek `employee-data` di bucket Amazon S3 `my-data` di wilayah `us-west-2`, lalu memuat data tersebut ke dalam tabel `employees`.

```
LOAD DATA FROM S3 PREFIX 's3-us-west-2://my-data/employee_data'
```

```
    INTO TABLE employees
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (ID, FIRSTNAME, LASTNAME, EMAIL, SALARY);
```

Pernyataan berikut memuat data dari file yang ditentukan dalam file manifes JSON bernama q1_sales.json ke dalam tabel `sales`.

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/q1_sales.json'
    INTO TABLE sales
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    (MONTH, STORE, GROSS, NET);
```

LOAD XML FROM S3

Anda dapat menggunakan pernyataan `LOAD XML FROM S3` untuk memuat data dari file XML yang tersimpan di bucket Amazon S3 dalam salah satu dari tiga format XML yang berbeda:

- Nama kolom sebagai atribut elemen `<row>`. Nilai atribut mengidentifikasi konten bidang tabel.

  ```
  <row column1="value1" column2="value2" .../>
  ```

- Nama kolom sebagai elemen turunan dari elemen `<row>`. Nilai elemen turunan mengidentifikasi konten bidang tabel.

  ```
  <row>
    <column1>value1</column1>
    <column2>value2</column2>
  </row>
  ```

- Nama kolom di atribut `name` dari elemen `<field>` dalam elemen `<row>`. Nilai elemen `<field>` mengidentifikasi konten bidang tabel.

  ```
  <row>
    <field name='column1'>value1</field>
    <field name='column2'>value2</field>
  </row>
  ```

Sintaksis

```
LOAD XML FROM S3 'S3-URI'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name,...)]
    [CHARACTER SET charset_name]
    [ROWS IDENTIFIED BY '<element-name>']
    [IGNORE number {LINES | ROWS}]
    [(field_name_or_user_var,...)]
    [SET col_name = expr,...]
```

Parameter

Pernyataan `LOAD XML FROM S3` menggunakan parameter wajib dan opsional berikut. Anda dapat menemukan detail selengkapnya tentang beberapa parameter ini dalam LOAD XML Statement dalam dokumentasi MySQL.

FILE | PREFIX

Mengidentifikasi apakah akan memuat data dari satu file, atau dari semua file yang cocok dengan awalan yang diberikan. `FILE` adalah opsi default.

REPLACE | IGNORE

Menentukan tindakan yang harus diambil jika baris input memiliki nilai kunci unik yang sama dengan baris yang ada dalam tabel basis data.

- Tentukan `REPLACE` jika Anda ingin baris input menggantikan baris yang ada dalam tabel.
- Tentukan `IGNORE` jika Anda ingin membuang baris input. `IGNORE` adalah opsi default.

INTO TABLE

Mengidentifikasi nama tabel basis data yang akan dimuati dengan baris input.

PARTITION

Mengharuskan semua baris input disisipkan ke partisi yang diidentifikasi oleh daftar nama partisi yang dipisahkan koma yang ditentukan. Jika baris input tidak dapat disisipkan ke salah satu partisi yang ditentukan, maka pernyataan ini akan gagal dan kesalahan akan ditampilkan.

CHARACTER SET

Mengidentifikasi kumpulan karakter data dalam file input.

ROWS IDENTIFIED BY

Mengidentifikasi nama elemen yang mengidentifikasi baris dalam file input. Default-nya adalah `<row>`.

IGNORE *angka* LINES | ROWS

Menentukan untuk mengabaikan jumlah tertentu baris di awal file input. Misalnya, Anda dapat menggunakan `IGNORE 1 LINES` untuk melewati baris pertama dalam file teks, atau `IGNORE 2 ROWS` untuk melewati dua baris pertama data dalam XML input.

field_name_or_user_var, ...

Menentukan daftar yang dipisahkan koma yang berisi satu atau beberapa nama elemen XML atau variabel pengguna yang mengidentifikasi elemen mana yang akan dimuat berdasarkan nama. Nama variabel pengguna yang digunakan untuk tujuan ini harus cocok dengan nama elemen dari file XML, yang diawali dengan @. Anda dapat menggunakan variabel pengguna untuk menyimpan nilai bidang terkait untuk digunakan kembali nanti.

Misalnya, pernyataan berikut memuat kolom pertama dari file input ke kolom pertama `table1`, dan menetapkan nilai kolom `table_column2` di `table1` ke nilai input kolom kedua dibagi 100.

```
LOAD XML FROM S3 's3://mybucket/data.xml'
   INTO TABLE table1
   (column1, @var1)
   SET table_column2 = @var1/100;
```

SET

Menentukan daftar operasi penetapan yang dipisahkan koma yang mengatur nilai kolom dalam tabel ke nilai yang tidak disertakan dalam file input.

Misalnya, pernyataan berikut mengatur dua kolom pertama `table1` ke nilai di dua kolom pertama dari file input, lalu mengatur nilai `column3` di `table1` ke stempel waktu saat ini.

```
LOAD XML FROM S3 's3://mybucket/data.xml'
   INTO TABLE table1
   (column1, column2)
   SET column3 = CURRENT_TIMESTAMP;
```

Anda dapat menggunakan subkueri di sisi kanan penetapan SET. Untuk subkueri yang menampilkan nilai yang akan ditetapkan ke kolom, Anda hanya dapat menggunakan subkueri

skalar. Selain itu, Anda tidak dapat menggunakan subkueri untuk memilih dari tabel yang sedang dimuat.

Menyimpan data dari klaster DB Amazon Aurora MySQL ke dalam file teks di bucket Amazon S3

Anda dapat menggunakan pernyataan `SELECT INTO OUTFILE S3` untuk mengkueri data dari klaster DB Amazon Aurora MySQL dan menyimpannya langsung ke dalam file teks yang tersimpan di bucket Amazon S3. Anda dapat menggunakan fungsionalitas ini agar tidak perlu membawa data ke klien terlebih dahulu, lalu menyalinnya dari klien ke Amazon S3. Pernyataan `LOAD DATA FROM S3` dapat menggunakan file yang dibuat oleh pernyataan ini untuk memuat data ke klaster DB Aurora. Untuk informasi selengkapnya, lihat Memuat data ke klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3.

Anda dapat mengenkripsi bucket Amazon S3 menggunakan kunci yang dikelola Amazon S3 (SSE-S3) atau Kunci yang dikelola AWS (SSE-SMS: AWS KMS key atau kunci yang dikelola pelanggan).

Fitur ini tidak didukung untuk klaster DB Aurora Serverless v1. Hal ini didukung untuk klaster DB Aurora Serverless v2.

> **Note**
>
> Anda dapat menyimpan data snapshot klaster DB ke Amazon S3 menggunakan AWS Management Console, AWS CLI, atau API Amazon RDS. Untuk informasi selengkapnya, lihat Mengekspor data snapshot klaster DB ke Amazon S3.

Memberi Aurora MySQL akses ke Amazon S3

Sebelum Anda dapat menyimpan data ke bucket Amazon S3, Anda harus terlebih dahulu memberi klaster DB Aurora MySQL Anda izin untuk mengakses Amazon S3.

Untuk memberi Aurora MySQL akses ke Amazon S3

1. Buat kebijakan AWS Identity and Access Management (IAM) yang memberikan izin bucket dan objek yang memungkinkan klaster DB Aurora MySQL Anda mengakses Amazon S3. Untuk petunjuk, lihat Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3.

> **Note**
>
> Di Aurora MySQL versi 3.05 dan lebih tinggi, Anda dapat mengenkripsi objek menggunakan kunci yang dikelola pelanggan AWS KMS. Untuk melakukannya, sertakan izin `kms:GenerateDataKey` dalam kebijakan IAM Anda. Untuk informasi selengkapnya, lihat Membuat kebijakan IAM untuk mengakses sumber daya AWS KMS. Anda tidak memerlukan izin ini untuk mengenkripsi objek menggunakan Kunci yang dikelola AWS atau kunci terkelola Amazon S3 (SSE-S3).

2. Buat peran IAM, lalu lampirkan kebijakan IAM yang Anda buat di Membuat kebijakan IAM untuk mengakses sumber daya Amazon S3 ke peran IAM baru. Untuk petunjuk, lihat Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.
3. Untuk Aurora MySQL versi 2, atur parameter klaster DB `aurora_select_into_s3_role` atau `aws_default_s3_role` ke Amazon Resource Name (ARN) milik peran IAM yang baru. Jika peran IAM tidak ditentukan untuk `aurora_select_into_s3_role`, Aurora menggunakan peran IAM yang ditentukan dalam `aws_default_s3_role`.

   Untuk Aurora MySQL versi 3, gunakan `aws_default_s3_role`.

   Jika klaster adalah bagian dari basis data global Aurora, atur parameter ini untuk setiap klaster Aurora dalam basis data global.

   Untuk informasi selengkapnya tentang parameter klaster DB, lihat Parameter instans DB dan klaster DB Amazon Aurora.
4. Untuk mengizinkan pengguna basis data dalam klaster DB Aurora MySQL untuk mengakses Amazon S3, kaitkan peran yang Anda buat di Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS dengan klaster DB.

   Untuk basis data global Aurora, kaitkan peran dengan setiap klaster Aurora di basis data global.

   Untuk informasi tentang mengaitkan peran IAM dengan klaster DB, lihat Mengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL.
5. Konfigurasikan klaster DB Aurora MySQL Anda untuk memungkinkan koneksi keluar ke Amazon S3. Untuk petunjuk, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora MySQL ke layanan AWS lainnya.

Untuk basis data global Aurora, aktifkan koneksi keluar untuk setiap klaster Aurora di basis data global.

Memberikan hak akses untuk menyimpan data di Aurora MySQL

Pengguna basis data yang mengeluarkan pernyataan `SELECT INTO OUTFILE S3` harus memiliki peran atau hak akses tertentu. Di Aurora MySQL versi 3, Anda memberikan peran `AWS_SELECT_S3_ACCESS`. Di Aurora MySQL versi 2, Anda memberikan hak akses `SELECT INTO S3`. Pengguna administratif untuk klaster DB akan diberi peran atau hak akses yang sesuai secara default. Anda dapat memberikan hak akses kepada pengguna lain dengan menggunakan salah satu pernyataan berikut.

Gunakan pernyataan berikut untuk Aurora MySQL versi 3:

```
GRANT AWS_SELECT_S3_ACCESS TO 'user'@'domain-or-ip-address'
```

> **Tip**
>
> Saat Anda menggunakan teknik peran di Aurora MySQL versi 3, Anda juga dapat mengaktifkan peran dengan menggunakan pernyataan `SET ROLE role_name` atau `SET ROLE ALL`. Jika Anda tidak memahami sistem peran MySQL 8.0, Anda dapat mempelajari selengkapnya dalam Model hak akses berbasis peran. Untuk detail selengkapnya, lihat Using roles dalam Panduan Referensi MySQL.
>
> Hal ini hanya berlaku untuk sesi aktif saat ini. Ketika Anda terhubung kembali, Anda harus menjalankan pernyataan `SET ROLE` lagi untuk memberikan hak akses. Untuk informasi selengkapnya, lihat SET ROLE statement dalam Panduan Referensi MySQL.
>
> Anda dapat menggunakan parameter klaster DB `activate_all_roles_on_login` untuk mengaktifkan semua peran secara otomatis saat pengguna terhubung ke instans DB. Ketika parameter ini ditetapkan, Anda tidak perlu memanggil pernyataan SET ROLE secara eksplisit untuk mengaktifkan peran. Untuk informasi selengkapnya, lihat activate_all_roles_on_login dalam Panduan Referensi MySQL.

Gunakan pernyataan berikut untuk Aurora MySQL versi 2:

```
GRANT SELECT INTO S3 ON *.* TO 'user'@'domain-or-ip-address'
```

Peran `AWS_SELECT_S3_ACCESS` dan hak akses `SELECT INTO S3` dikhususkan untuk Amazon Aurora MySQL dan tidak tersedia untuk basis data MySQL atau instans DB RDS for MySQL. Jika Anda telah mengatur replikasi antara klaster DB Aurora MySQL sebagai master replikasi dan basis data MySQL sebagai klien replikasi, maka pernyataan `GRANT` untuk peran atau hak akses menyebabkan replikasi berhenti dengan kesalahan. Anda dapat melewati kesalahan ini dengan aman untuk melanjutkan replikasi. Untuk melewati kesalahan pada instans DB RDS for MySQL, gunakan prosedur mysql_rds_skip_repl_error. Untuk melewati kesalahan pada basis data MySQL eksternal, gunakan variabel sistem slave_skip_errors (Aurora MySQL versi 2) atau variabel sistem replica_skip_errors (Aurora MySQL versi 3).

Menentukan jalur ke bucket Amazon S3

Sintaksis untuk menentukan jalur untuk menyimpan data dan file manifes di bucket Amazon S3 serupa dengan yang digunakan dalam pernyataan `LOAD DATA FROM S3 PREFIX`, seperti yang ditunjukkan berikut ini.

```
s3-region://bucket-name/file-prefix
```

Jalur tersebut mencakup nilai-nilai berikut:

- `region` (opsional) – Wilayah AWS yang berisi bucket Amazon S3 untuk menyimpan data. Nilai ini bersifat opsional. Jika Anda tidak menentukan nilai `region`, Aurora akan menyimpan file Anda ke Amazon S3 di wilayah yang sama dengan klaster DB Anda.
- `bucket-name` – Nama bucket Amazon S3 untuk menyimpan data. Awalan objek yang mengidentifikasi jalur folder virtual didukung.
- `file-prefix` – Awalan objek Amazon S3 yang mengidentifikasi file yang akan disimpan di Amazon S3.

File data yang dibuat oleh pernyataan `SELECT INTO OUTFILE S3` akan menggunakan jalur berikut, dengan *`00000`* merepresentasikan bilangan bulat 5 digit berbasis nol.

```
s3-region://bucket-name/file-prefix.part_00000
```

Misalnya, pernyataan `SELECT INTO OUTFILE S3` menetapkan `s3-us-west-2://bucket/prefix` sebagai jalur untuk menyimpan file data dan membuat tiga file data. Bucket Amazon S3 yang ditentukan berisi file data berikut.

- s3-us-west-2://bucket/prefix.part_00000

- s3-us-west-2://bucket/prefix.part_00001
- s3-us-west-2://bucket/prefix.part_00002

Membuat manifes untuk menampilkan daftar file data

Anda dapat menggunakan pernyataan `SELECT INTO OUTFILE S3` dengan opsi `MANIFEST ON` untuk membuat file manifes dalam format JSON yang menampilkan daftar file teks yang dibuat oleh pernyataan tersebut. Pernyataan `LOAD DATA FROM S3` dapat menggunakan file manifes untuk memuat file data kembali ke klaster DB Aurora MySQL. Untuk informasi selengkapnya tentang menggunakan manifes untuk memuat file data dari Amazon S3 ke dalam klaster DB Aurora MySQL, lihat Menggunakan manifes untuk menentukan file data yang akan dimuat.

File data yang disertakan dalam manifes yang dibuat oleh pernyataan `SELECT INTO OUTFILE S3` akan dicantumkan dalam urutan pembuatannya oleh pernyataan tersebut. Misalnya, pernyataan `SELECT INTO OUTFILE S3` menetapkan `s3-us-west-2://bucket/prefix` sebagai jalur untuk menyimpan file data serta membuat tiga file data dan sebuah file manifes. Bucket Amazon S3 yang ditentukan berisi file manifes bernama `s3-us-west-2://bucket/prefix.manifest`, yang berisi informasi berikut.

```
{
  "entries": [
    {
      "url":"s3-us-west-2://bucket/prefix.part_00000"
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00001"
    },
    {
      "url":"s3-us-west-2://bucket/prefix.part_00002"
    }
  ]
}
```

SELECT INTO OUTFILE S3

Anda dapat menggunakan pernyataan `SELECT INTO OUTFILE S3` untuk meminta data dari klaster DB dan menyimpannya secara langsung ke dalam file teks yang dibatasi yang tersimpan di bucket Amazon S3.

File terkompresi tidak didukung. File terenkripsi didukung mulai dari Aurora MySQL versi 2.09.0.

Sintaksis

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
        [HIGH_PRIORITY]
        [STRAIGHT_JOIN]
        [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
        [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
        [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
        [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
         [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
    [export_options]
    [MANIFEST {ON | OFF}]
    [OVERWRITE {ON | OFF}]
    [ENCRYPTION {ON | OFF | SSE_S3 | SSE_KMS ['cmk_id']}]

export_options:
    [FORMAT {CSV|TEXT} [HEADER]]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
]
```

Parameter

Pernyataan `SELECT INTO OUTFILE S3` menggunakan parameter wajib dan opsional berikut yang khusus untuk Aurora.

s3-uri

Menentukan URI untuk awalan Amazon S3 yang akan digunakan. Gunakan sintaks yang dijelaskan dalam Menentukan jalur ke bucket Amazon S3.

FORMAT {CSV|TEXT} [HEADER]

Secara opsional menyimpan data dalam format CSV.

Opsi `TEXT` adalah default dan menghasilkan format ekspor MySQL yang ada.

Opsi `CSV` menghasilkan nilai data yang dipisahkan koma. Format CSV mengikuti spesifikasi dalam RFC-4180. Jika Anda menentukan kata kunci opsional `HEADER`, file output akan berisi satu baris header. Label di baris header sesuai dengan nama kolom dari pernyataan `SELECT`. Anda dapat menggunakan file CSV untuk model data pelatihan untuk digunakan dengan layanan ML AWS. Untuk informasi selengkapnya tentang menggunakan data Aurora yang diekspor dengan layanan ML AWS, lihat Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model (Lanjutan).

MANIFEST {ON | OFF}

Menunjukkan apakah file manifes dibuat di Amazon S3. File manifes adalah file JavaScript Object Notation (JSON) yang dapat digunakan untuk memuat data ke klaster DB Aurora dengan pernyataan `LOAD DATA FROM S3 MANIFEST`. Untuk informasi selengkapnya tentang `LOAD DATA FROM S3 MANIFEST`, lihat Memuat data ke klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3.

Jika `MANIFEST ON` ditentukan dalam kueri, file manifes dibuat di Amazon S3 setelah semua file data dibuat dan diunggah. file manifes dibuat menggunakan jalur berikut:

```
s3-region://bucket-name/file-prefix.manifest
```

Untuk informasi selengkapnya tentang format konten file manifes, lihat Membuat manifes untuk menampilkan daftar file data.

OVERWRITE {ON | OFF}

Menunjukkan apakah file yang ada di bucket Amazon S3 yang ditentukan akan ditimpa. Jika `OVERWRITE ON` ditentukan, file yang ada yang cocok dengan awalan file di URI yang ditentukan di `s3-uri` akan ditimpa. Jika tidak, kesalahan akan muncul.

ENCRYPTION {ON | OFF | SSE_S3 | SSE_KMS ['*cmk_id*']}

Menunjukkan apakah akan menggunakan enkripsi di sisi server dengan kunci yang dikelola Amazon S3 (SSE-S3) atau AWS KMS keys (SSE-KMS, termasuk Kunci yang dikelola AWS dan kunci yang dikelola pelanggan). Pengaturan `SSE_KMS` dan `SSE_S3` tersedia di Aurora MySQL versi 3.05 dan lebih tinggi.

Anda juga dapat menggunakan variabel sesi `aurora_select_into_s3_encryption_default` bukan klausa `ENCRYPTION`, seperti yang ditunjukkan pada contoh berikut. Gunakan salah satu klausa SQL atau variabel sesi, tetapi tidak keduanya.

```
set session set session aurora_select_into_s3_encryption_default={ON | OFF | SSE_S3
 | SSE_KMS};
```

Pengaturan `SSE_KMS` dan `SSE_S3` tersedia di Aurora MySQL versi 3.05 dan lebih tinggi.

Jika Anda mengatur `aurora_select_into_s3_encryption_default` ke nilai berikut:

- `OFF` – Kebijakan enkripsi default bucket S3 akan diikuti. Nilai default `aurora_select_into_s3_encryption_default` adalah `OFF`.
- `ON` atau `SSE_S3` – Objek S3 dienkripsi menggunakan kunci terkelola Amazon S3 (SSE-S3).
- `SSE_KMS` – Objek S3 dienkripsi menggunakan AWS KMS key.

  Dalam hal ini, Anda juga menyertakan variabel sesi `aurora_s3_default_cmk_id`, misalnya:

  ```
  set session aurora_select_into_s3_encryption_default={SSE_KMS};
  set session aurora_s3_default_cmk_id={NULL | 'cmk_id'};
  ```

  - Saat `aurora_s3_default_cmk_id` bernilai `NULL`, objek S3 dienkripsi menggunakan Kunci yang dikelola AWS.
  - Jika `aurora_s3_default_cmk_id` berupa string `cmk_id` yang tidak kosong, objek S3 dienkripsi menggunakan kunci yang dikelola pelanggan.

    Nilai `cmk_id` tidak boleh berupa string kosong.

Saat Anda menggunakan perintah `SELECT INTO OUTFILE S3`, Aurora menentukan enkripsi sebagai berikut:

- Jika klausa `ENCRYPTION` ada dalam perintah SQL, Aurora hanya mengandalkan nilai `ENCRYPTION`, dan tidak menggunakan variabel sesi.

- Jika klausa `ENCRYPTION` tidak ada, Aurora mengandalkan nilai variabel sesi.

Untuk informasi selengkapnya, lihat Menggunakan enkripsi di sisi server dengan kunci terkelola Amazon S3 (SSE-S3) dan Menggunakan enkripsi di sisi server dengan kunci AWS KMS (SSE-KMS) dalam Panduan Pengguna Amazon Simple Storage Service.

Anda dapat menemukan detail selengkapnya tentang parameter lain dalam SELECT statement dan LOAD DATA statement dalam dokumentasi MySQL.

Pertimbangan

Jumlah file yang ditulis ke bucket Amazon S3 bergantung pada jumlah data yang dipilih oleh pernyataan `SELECT INTO OUTFILE S3` dan ambang batas ukuran file untuk Aurora MySQL. Ambang batas ukuran file default adalah 6 gigabyte (GB). Jika data yang dipilih oleh pernyataan kurang dari ambang batas ukuran file, satu file akan dibuat; jika tidak, banyak file akan dibuat. Pertimbangan lain untuk file yang dibuat oleh pernyataan ini mencakup hal-hal berikut:

- Aurora MySQL menjamin bahwa baris dalam file data tidak dipisahkan melintasi batas file. Untuk banyak file, ukuran setiap file data kecuali yang terakhir biasanya hampir sama dengan ambang batas ukuran file. Namun, terkadang jika ambang batas ukuran file tidak tercapai, akibatnya baris akan terbagi menjadi dua file data. Dalam kasus ini, Aurora MySQL membuat file data yang menjaga baris tetap utuh, tetapi mungkin lebih besar dari ambang ukuran file.
- Karena setiap pernyataan `SELECT` di Aurora MySQL dijalankan sebagai transaksi atomik, pernyataan `SELECT INTO OUTFILE S3` yang memilih set data besar mungkin akan berjalan selama beberapa waktu. Jika pernyataan gagal karena alasan apa pun, Anda mungkin perlu memulai kembali dan mengeluarkan pernyataan tersebut lagi. Namun, jika pernyataan gagal, file yang sudah diunggah ke Amazon S3 tetap berada di bucket Amazon S3 yang ditentukan. Anda dapat menggunakan pernyataan lain untuk mengunggah data yang tersisa daripada memulai dari awal lagi.
- Jika jumlah data yang akan dipilih berukuran besar (lebih dari 25 GB), kami menyarankan Anda menggunakan beberapa pernyataan `SELECT INTO OUTFILE S3` untuk menyimpan data ke Amazon S3. Setiap pernyataan harus memilih bagian data yang berbeda untuk disimpan, dan juga menentukan `file_prefix` yang berbeda dalam parameter `s3-uri` untuk digunakan saat menyimpan file data. Dengan mempartisi data yang akan dipilih menggunakan beberapa pernyataan, kesalahan di satu pernyataan akan lebih mudah untuk dipulihkan. Jika terjadi kesalahan untuk satu pernyataan, hanya sebagian data yang perlu dipilih kembali dan diunggah

ke Amazon S3. Penggunaan banyak pernyataan juga membantu menghindari satu transaksi yang berjalan lama, yang dapat meningkatkan performa.

- Jika beberapa pernyataan `SELECT INTO OUTFILE S3` yang menggunakan `file_prefix` yang sama di parameter `s3-uri` dijalankan secara paralel untuk memilih data yang diunggah ke dalam Amazon S3, perilakunya tidak akan ditentukan.
- Metadata, seperti skema tabel atau metadata file, tidak diunggah oleh Aurora MySQL ke Amazon S3.
- Dalam beberapa kasus, Anda mungkin perlu menjalankan kembali kueri `SELECT INTO OUTFILE S3`, seperti untuk memulihkan dari kegagalan. Dalam kasus ini, Anda harus menghapus file data apa pun yang ada di bucket Amazon S3 dengan awalan file yang sama yang ditentukan di `s3-uri`, atau menyertakan `OVERWRITE ON` di kueri `SELECT INTO OUTFILE S3`.

Pernyataan `SELECT INTO OUTFILE S3` menampilkan nomor kesalahan MySQL yang biasa beserta respons berhasil atau gagal. Jika Anda tidak memiliki akses ke nomor dan respons kesalahan MySQL, cara termudah untuk menentukan apakah pernyataan tersebut telah selesai adalah dengan menentukan `MANIFEST ON` dalam pernyataan tersebut. File manifes adalah file terakhir yang ditulis oleh pernyataan tersebut. Dengan kata lain, jika Anda memiliki file manifes, berarti pernyataan tersebut telah selesai.

Saat ini, tidak ada cara untuk memantau secara langsung progres pernyataan `SELECT INTO OUTFILE S3` saat dijalankan. Namun, misalkan Anda menulis data dalam jumlah besar dari Aurora MySQL ke Amazon S3 menggunakan pernyataan ini, dan Anda mengetahui ukuran data yang dipilih oleh pernyataan tersebut. Dalam kasus ini, Anda dapat memperkirakan progresnya dengan memantau pembuatan file data di Amazon S3.

Untuk melakukannya, Anda dapat menggunakan fakta bahwa sebuah file data akan dibuat di bucket Amazon S3 yang ditentukan untuk setiap 6 GB data yang dipilih oleh pernyataan tersebut. Bagilah ukuran data yang dipilih dengan 6 GB untuk mendapatkan perkiraan jumlah file data yang akan dibuat. Anda kemudian dapat memperkirakan progres pernyataan dengan memantau jumlah file yang diunggah ke Amazon S3 saat pernyataan berjalan.

Contoh

Pernyataan berikut memilih semua data di tabel `employees` dan menyimpan data ke dalam bucket Amazon S3 yang berada di wilayah yang berbeda dari klaster DB Aurora MySQL. Pernyataan tersebut membuat file data yang setiap bidangnya diterminasi dengan karakter koma (`,`) dan setiap barisnya diterminasi dengan karakter baris baru (`\n`). Pernyataan tersebut menampilkan kesalahan

jika file yang cocok dengan awalan file `sample_employee_data` ada di bucket Amazon S3 yang ditentukan.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n';
```

Pernyataan berikut memilih semua data di tabel `employees` dan menyimpan data ke dalam bucket Amazon S3 yang berada di wilayah yang sama dengan klaster DB Aurora MySQL. Pernyataan tersebut membuat file data yang setiap bidangnya diterminasi dengan karakter koma (`,`) dan setiap barisnya diterminasi dengan karakter baris baru (`\n`), serta membuat juga sebuah file manifes. Pernyataan tersebut menampilkan kesalahan jika file yang cocok dengan awalan file `sample_employee_data` ada di bucket Amazon S3 yang ditentukan.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON;
```

Pernyataan berikut memilih semua data di tabel `employees` dan menyimpan data ke dalam bucket Amazon S3 yang berada di wilayah yang berbeda dari klaster DB Aurora. Pernyataan tersebut membuat file data yang setiap bidangnya diterminasi dengan karakter koma (`,`) dan setiap barisnya diterminasi dengan karakter baris baru (`\n`). Pernyataan tersebut menimpa file apa pun yang ada yang cocok dengan awalan file `sample_employee_data` di bucket Amazon S3 yang ditentukan.

```
SELECT * FROM employees INTO OUTFILE S3 's3-us-west-2://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    OVERWRITE ON;
```

Pernyataan berikut memilih semua data di tabel `employees` dan menyimpan data ke dalam bucket Amazon S3 yang berada di wilayah yang sama dengan klaster DB Aurora MySQL. Pernyataan tersebut membuat file data yang setiap bidangnya diterminasi dengan karakter koma (`,`) dan setiap barisnya diterminasi dengan karakter baris baru (`\n`), serta membuat juga sebuah file manifes. Pernyataan tersebut menimpa file apa pun yang ada yang cocok dengan awalan file `sample_employee_data` di bucket Amazon S3 yang ditentukan.

```
SELECT * FROM employees INTO OUTFILE S3 's3://aurora-select-into-s3-pdx/
sample_employee_data'
    FIELDS TERMINATED BY ','
    LINES TERMINATED BY '\n'
    MANIFEST ON
    OVERWRITE ON;
```

Menginvokasi fungsi Lambda dari klaster DB Amazon Aurora MySQL

Anda dapat menginvokasi fungsi AWS Lambda dari klaster DB Amazon Aurora MySQL yang kompatibel dengan fungsi native `lambda_sync` atau `lambda_async`. Sebelum menginvokasi fungsi Lambda dari Aurora MySQL, klaster DB Aurora harus memiliki akses ke Lambda. Untuk detail tentang pemberian akses ke Aurora MySQL, lihat Memberi Aurora akses ke Lambda. Untuk informasi tentang fungsi tersimpan `lambda_sync` dan `lambda_async`, lihat Menginvokasi fungsi Lambda dengan fungsi native Aurora MySQL.

Anda juga dapat memanggil fungsi AWS Lambda dengan menggunakan prosedur tersimpan. Namun, penggunaan prosedur tersimpan sudah ditiadakan. Sebaiknya gunakan fungsi native Aurora MySQL jika Anda menggunakan salah satu dari versi Aurora MySQL berikut:

- Aurora MySQL versi 2, untuk klaster yang kompatibel dengan MySQL 5.7.
- Aurora MySQL versi 3.01 dan lebih tinggi, untuk klaster yang kompatibel dengan MySQL 8.0. Prosedur tersimpan tidak tersedia di Aurora MySQL versi 3.

Topik

- Memberi Aurora akses ke Lambda
- Menginvokasi fungsi Lambda dengan fungsi native Aurora MySQL
- Menginvokasi fungsi Lambda dengan prosedur tersimpan Aurora MySQL (sudah ditiadakan)

Memberi Aurora akses ke Lambda

Sebelum Anda dapat menginvokasi fungsi Lambda dari klaster DB Aurora MySQL, pastikan untuk terlebih dahulu memberikan izin klaster untuk mengakses Lambda.

Untuk memberi Aurora MySQL akses ke Lambda

1. Buat kebijakan AWS Identity and Access Management (IAM) yang memberikan izin yang memungkinkan klaster DB Aurora MySQL Anda menginvokasi fungsi Lambda. Untuk petunjuk, lihat Membuat kebijakan IAM untuk mengakses sumber daya AWS Lambda.
2. Buat peran IAM, lalu lampirkan kebijakan IAM yang Anda buat di Membuat kebijakan IAM untuk mengakses sumber daya AWS Lambda ke peran IAM baru. Untuk petunjuk, lihat Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS.
3. Atur parameter klaster DB `aws_default_lambda_role` ke Amazon Resource Name (ARN) dari peran IAM baru.

   Jika klaster tersebut adalah bagian dari basis data global Aurora, terapkan pengaturan yang sama untuk setiap klaster Aurora di basis data global.

   Untuk informasi selengkapnya tentang parameter klaster DB, lihat Parameter instans DB dan klaster DB Amazon Aurora.
4. Untuk mengizinkan pengguna basis data di klaster DB Aurora MySQL untuk menginvokasi fungsi Lambda, kaitkan peran yang Anda buat di Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS dengan klaster DB. Untuk informasi tentang mengaitkan peran IAM dengan klaster DB, lihat Mengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL.

   Jika klaster adalah bagian dari basis data global Aurora, kaitkan peran dengan setiap klaster Aurora di basis data global.
5. Konfigurasikan klaster DB Aurora MySQL Anda untuk memungkinkan koneksi keluar ke Lambda. Untuk petunjuk, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora MySQL ke layanan AWS lainnya.

   Jika klaster adalah bagian dari basis data global Aurora, aktifkan koneksi keluar untuk setiap klaster Aurora di basis data global.

Menginvokasi fungsi Lambda dengan fungsi native Aurora MySQL

> **Note**
>
> Anda dapat memanggil fungsi native `lambda_sync` dan `lambda_async` saat Anda menggunakan Aurora MySQL versi 2, atau Aurora MySQL versi 3.01 dan lebih tinggi. Untuk

informasi selengkapnya tentang versi Aurora MySQL, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL.

Anda dapat menginvokasi fungsi AWS Lambda dari klaster DB Aurora MySQL dengan memanggil fungsi native `lambda_sync` dan `lambda_async`. Pendekatan ini dapat berguna saat Anda ingin mengintegrasikan basis data Anda yang berjalan di Aurora MySQL dengan layanan AWS lainnya. Misalnya, Anda mungkin ingin mengirim notifikasi menggunakan Amazon Simple Notification Service (Amazon SNS) setiap kali sebuah baris dimasukkan ke dalam tabel tertentu di basis data Anda.

Daftar Isi

- Menggunakan fungsi native untuk menginvokasi fungsi Lambda
  - Memberikan peran dalam Aurora MySQL versi 3
  - Memberikan hak akses di Aurora MySQL versi 2
  - Sintaksis untuk fungsi lambda_sync
  - Parameter untuk fungsi lambda_sync
  - Contoh untuk fungsi lambda_sync
  - Sintaksis untuk fungsi lambda_async
  - Parameter untuk fungsi lambda_async
  - Contoh untuk fungsi lambda_async
  - Menginvokasi fungsi Lambda di dalam pemicu

Menggunakan fungsi native untuk menginvokasi fungsi Lambda

Fungsi `lambda_sync` dan `lambda_async` adalah fungsi native bawaan yang menginvokasi fungsi Lambda secara sinkron atau asinkron. Ketika Anda harus mengetahui hasil dari fungsi Lambda sebelum beralih ke tindakan lain, gunakan fungsi sinkron `lambda_sync`. Ketika Anda tidak harus mengetahui hasil dari fungsi Lambda sebelum beralih ke tindakan lain, gunakan fungsi asinkron `lambda_async`.

Memberikan peran dalam Aurora MySQL versi 3

Di Aurora MySQL versi 3, pengguna yang menginvokasi fungsi native harus diberi peran `AWS_LAMBDA_ACCESS`. Untuk memberikan peran ini kepada pengguna, hubungkan ke instans DB sebagai pengguna administratif, dan jalankan pernyataan berikut.

```
GRANT AWS_LAMBDA_ACCESS TO user@domain-or-ip-address
```

Anda dapat mencabut peran ini dengan menjalankan pernyataan berikut.

```
REVOKE AWS_LAMBDA_ACCESS FROM user@domain-or-ip-address
```

> **Tip**
>
> Saat Anda menggunakan teknik peran di Aurora MySQL versi 3, Anda juga dapat mengaktifkan peran dengan menggunakan pernyataan `SET ROLE role_name` atau `SET ROLE ALL`. Jika Anda tidak memahami sistem peran MySQL 8.0, Anda dapat mempelajari selengkapnya dalam Model hak akses berbasis peran. Untuk detail selengkapnya, lihat Using roles dalam Panduan Referensi MySQL.
>
> Hal ini hanya berlaku untuk sesi aktif saat ini. Ketika Anda terhubung kembali, Anda harus menjalankan pernyataan `SET ROLE` lagi untuk memberikan hak akses. Untuk informasi selengkapnya, lihat SET ROLE statement dalam Panduan Referensi MySQL.
>
> Anda dapat menggunakan parameter klaster DB `activate_all_roles_on_login` untuk mengaktifkan semua peran secara otomatis saat pengguna terhubung ke instans DB. Ketika parameter ini ditetapkan, Anda tidak perlu memanggil pernyataan SET ROLE secara eksplisit untuk mengaktifkan peran. Untuk informasi selengkapnya, lihat activate_all_roles_on_login dalam Panduan Referensi MySQL.

Jika Anda mendapatkan kesalahan seperti berikut ketika Anda mencoba menginvokasi fungsi Lambda, kemudian jalankan pernyataan `SET ROLE`.

```
SQL Error [1227] [42000]: Access denied; you need (at least one of) the Invoke Lambda
 privilege(s) for this operation
```

Memberikan hak akses di Aurora MySQL versi 2

Di Aurora MySQL versi 2, pengguna yang menginvokasi fungsi native harus diberi hak akses `INVOKE LAMBDA`. Untuk memberikan hak akses ini kepada pengguna, hubungkan ke instans DB sebagai pengguna administratif, lalu jalankan pernyataan berikut.

```
GRANT INVOKE LAMBDA ON *.* TO user@domain-or-ip-address
```

Anda dapat mencabut hak akses ini dengan menjalankan pernyataan berikut.

```
REVOKE INVOKE LAMBDA ON *.* FROM user@domain-or-ip-address
```

Sintaksis untuk fungsi lambda_sync

Anda menginvokasi fungsi `lambda_sync` secara sinkron dengan jenis invokasi `RequestResponse`. Fungsi ini menampilkan hasil invokasi Lambda dalam payload JSON. Fungsi tersebut memiliki sintaksis berikut.

```
lambda_sync (
  lambda_function_ARN,
  JSON_payload
)
```

Parameter untuk fungsi lambda_sync

Fungsi `lambda_sync` memiliki parameter berikut.

lambda_function_ARN

Amazon Resource Name (ARN) dari fungsi Lambda yang akan diinvokasi.

JSON_payload

Payload untuk fungsi Lambda yang diinvokasi, dalam format JSON.

> **Note**
>
> Aurora MySQL versi 3 mendukung fungsi penguraian JSON dari MySQL 8.0. Namun, Aurora MySQL versi 2 tidak menyertakan fungsi-fungsi tersebut. Penguraian JSON tidak diperlukan saat fungsi Lambda menghasilkan nilai atomik, seperti angka atau string.

Contoh untuk fungsi lambda_sync

Kueri berikut berdasarkan `lambda_sync` menginvokasi fungsi Lambda `BasicTestLambda` secara sinkron menggunakan fungsi ARN. Payload untuk fungsi tersebut adalah `{"operation": "ping"}`.

```
SELECT lambda_sync(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
```

```
    '{"operation": "ping"}');
```

Sintaksis untuk fungsi lambda_async

Anda menginvokasi fungsi `lambda_async` secara asinkron dengan jenis invokasi `Event`. Fungsi ini menampilkan hasil invokasi Lambda dalam payload JSON. Fungsi tersebut memiliki sintaksis berikut.

```
lambda_async (
  lambda_function_ARN,
  JSON_payload
)
```

Parameter untuk fungsi lambda_async

Fungsi `lambda_async` memiliki parameter berikut.

lambda_function_ARN

Amazon Resource Name (ARN) dari fungsi Lambda yang akan diinvokasi.

JSON_payload

Payload untuk fungsi Lambda yang diinvokasi, dalam format JSON.

> **Note**
>
> Aurora MySQL versi 3 mendukung fungsi penguraian JSON dari MySQL 8.0. Namun, Aurora MySQL versi 2 tidak menyertakan fungsi-fungsi tersebut. Penguraian JSON tidak diperlukan saat fungsi Lambda menghasilkan nilai atomik, seperti angka atau string.

Contoh untuk fungsi lambda_async

Kueri yang didasarkan pada `lambda_async` berikut akan menginvokasi fungsi Lambda `BasicTestLambda` secara asinkron menggunakan fungsi ARN. Payload untuk fungsi tersebut adalah `{"operation": "ping"}`.

```
SELECT lambda_async(
    'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
    '{"operation": "ping"}');
```

Menginvokasi fungsi Lambda di dalam pemicu

Anda dapat menggunakan pemicu untuk memanggil Lambda pada pernyataan pengubahan data. Contoh berikut menggunakan fungsi native `lambda_async` dan menyimpan hasilnya dalam variabel.

```
mysql>SET @result=0;
mysql>DELIMITER //
mysql>CREATE TRIGGER myFirstTrigger
      AFTER INSERT
          ON Test_trigger FOR EACH ROW
      BEGIN
      SELECT lambda_async(
          'arn:aws:lambda:us-east-1:123456789012:function:BasicTestLambda',
          '{"operation": "ping"}')
          INTO @result;
      END; //
mysql>DELIMITER ;
```

> **Note**
>
> Pemicu tidak dijalankan sekali per pernyataan SQL, tetapi sekali per baris yang diubah, satu baris dalam satu waktu. Saat pemicu berjalan, prosesnya sinkron. Pernyataan modifikasi data hanya menampilkan hasil saat pemicu selesai.
> Berhati-hatilah saat menginvokasi fungsi AWS Lambda dari pemicu pada tabel yang mengalami lalu lintas tulis tinggi. Pemicu `INSERT`, `UPDATE`, dan `DELETE` diaktifkan per baris. Beban kerja sarat penulisan pada tabel dengan pemicu `INSERT`, `UPDATE`, atau `DELETE` akan menghasilkan banyak panggilan ke fungsi AWS Lambda Anda.

Menginvokasi fungsi Lambda dengan prosedur tersimpan Aurora MySQL (sudah ditiadakan)

Anda dapat menginvokasi fungsi AWS Lambda dari klaster DB Aurora MySQL dengan memanggil prosedur `mysql.lambda_async`. Pendekatan ini dapat berguna saat Anda ingin mengintegrasikan basis data Anda yang berjalan di Aurora MySQL dengan layanan AWS lainnya. Misalnya, Anda mungkin ingin mengirim notifikasi menggunakan Amazon Simple Notification Service (Amazon SNS) setiap kali sebuah baris dimasukkan ke dalam tabel tertentu di basis data Anda.

Daftar Isi

- Pertimbangan versi Aurora MySQL
- Menggunakan prosedur mysql.lambda_async untuk menginvokasi fungsi Lambda (sudah ditiadakan)
  - Sintaksis
  - Parameter
  - Contoh

Pertimbangan versi Aurora MySQL

Mulai dari Aurora MySQL versi 2, Anda dapat menggunakan metode fungsi native, bukan prosedur tersimpan ini untuk menginvokasi fungsi Lambda. Untuk informasi selengkapnya tentang fungsi native, lihat Menggunakan fungsi native untuk menginvokasi fungsi Lambda.

Di Aurora MySQL versi 2, prosedur tersimpan `mysql.lambda_async` tidak didukung lagi. Sebaiknya gunakan fungsi Lambda native sebagai gantinya.

Di Aurora MySQL versi 3, prosedur tersimpan tidak tersedia.

Menggunakan prosedur mysql.lambda_async untuk menginvokasi fungsi Lambda (sudah ditiadakan)

Prosedur `mysql.lambda_async` adalah prosedur tersimpan bawaan yang menginvokasi fungsi Lambda secara asinkron. Untuk menggunakan prosedur ini, pengguna basis data Anda harus memiliki hak akses `EXECUTE` pada prosedur tersimpan `mysql.lambda_async`.

Sintaksis

Prosedur `mysql.lambda_async` memiliki sintaksis berikut.

```
CALL mysql.lambda_async (
  lambda_function_ARN,
  lambda_function_input
)
```

Parameter

Prosedur `mysql.lambda_async` memiliki parameter berikut.

lambda_function_ARN

Amazon Resource Name (ARN) dari fungsi Lambda yang akan diinvokasi.

lambda_function_input

String input, dalam format JSON, untuk fungsi Lambda yang diinvokasi.

Contoh

Sebagai praktik terbaik, sebaiknya selesaikan panggilan ke prosedur `mysql.lambda_async` dalam prosedur tersimpan yang dapat dipanggil dari sumber berbeda seperti pemicu atau kode klien. Pendekatan ini dapat membantu menghindari masalah ketidakcocokan impedansi dan memudahkan Anda untuk menginvokasi fungsi Lambda.

> **Note**
>
> Berhati-hatilah saat menginvokasi fungsi AWS Lambda dari pemicu pada tabel yang mengalami lalu lintas tulis tinggi. Pemicu `INSERT`, `UPDATE`, dan `DELETE` diaktifkan per baris. Beban kerja sarat penulisan pada tabel dengan pemicu `INSERT`, `UPDATE`, atau `DELETE` akan menghasilkan banyak panggilan ke fungsi AWS Lambda Anda.
> Meskipun panggilan ke prosedur `mysql.lambda_async` bersifat asinkron, pemicunya bersifat sinkron. Pernyataan yang menghasilkan sejumlah besar aktivasi pemicu tidak menunggu panggilan ke fungsi AWS Lambda selesai, tetapi menunggu pemicu selesai sebelum mengembalikan kontrol ke klien.

Example Contoh: Invokasi fungsi AWS Lambda untuk mengirim email

Contoh berikut membuat prosedur tersimpan yang dapat Anda panggil dalam kode basis data Anda untuk mengirim email menggunakan fungsi Lambda.

Fungsi AWS Lambda

```
import boto3

ses = boto3.client('ses')

def SES_send_email(event, context):

    return ses.send_email(
        Source=event['email_from'],
        Destination={
```

```
            'ToAddresses': [
            event['email_to'],
            ]
        },

        Message={
            'Subject': {
            'Data': event['email_subject']
            },
            'Body': {
                'Text': {
                    'Data': event['email_body']
                }
            }
        }
    )
```

Prosedur Tersimpan

```
DROP PROCEDURE IF EXISTS SES_send_email;
DELIMITER ;;
  CREATE PROCEDURE SES_send_email(IN email_from VARCHAR(255),
                                  IN email_to VARCHAR(255),
                                  IN subject VARCHAR(255),
                                  IN body TEXT) LANGUAGE SQL
  BEGIN
    CALL mysql.lambda_async(
         'arn:aws:lambda:us-west-2:123456789012:function:SES_send_email',
         CONCAT('{"email_to" : "', email_to,
             '", "email_from" : "', email_from,
             '", "email_subject" : "', subject,
             '", "email_body" : "', body, '"}')
     );
  END
  ;;
DELIMITER ;
```

Panggil Prosedur Tersimpan untuk Menginvokasi Fungsi AWS Lambda

```
mysql> call SES_send_email('example_from@amazon.com', 'example_to@amazon.com', 'Email
 subject', 'Email content');
```

Example Contoh: Invokasi fungsi AWS Lambda untuk menerbitkan peristiwa dari pemicu

Contoh berikut membuat prosedur tersimpan yang menerbitkan peristiwa dengan menggunakan Amazon SNS. Kode ini memanggil prosedur dari pemicu ketika baris ditambahkan ke tabel.

Fungsi AWS Lambda

```
import boto3

sns = boto3.client('sns')

def SNS_publish_message(event, context):

    return sns.publish(
        TopicArn='arn:aws:sns:us-west-2:123456789012:Sample_Topic',
        Message=event['message'],
        Subject=event['subject'],
        MessageStructure='string'
    )
```

Prosedur Tersimpan

```
DROP PROCEDURE IF EXISTS SNS_Publish_Message;
DELIMITER ;;
CREATE PROCEDURE SNS_Publish_Message (IN subject VARCHAR(255),
                                      IN message TEXT) LANGUAGE SQL
BEGIN
  CALL mysql.lambda_async('arn:aws:lambda:us-
west-2:123456789012:function:SNS_publish_message',
     CONCAT('{ "subject" : "', subject,
            '", "message" : "', message, '" }')
     );
END
;;
DELIMITER ;
```

Tabel

```
CREATE TABLE 'Customer_Feedback' (
  'id' int(11) NOT NULL AUTO_INCREMENT,
  'customer_name' varchar(255) NOT NULL,
  'customer_feedback' varchar(1024) NOT NULL,
  PRIMARY KEY ('id')
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Pemicu

```
DELIMITER ;;
CREATE TRIGGER TR_Customer_Feedback_AI
  AFTER INSERT ON Customer_Feedback
  FOR EACH ROW
BEGIN
  SELECT CONCAT('New customer feedback from ', NEW.customer_name),
 NEW.customer_feedback INTO @subject, @feedback;
  CALL SNS_Publish_Message(@subject, @feedback);
END
;;
DELIMITER ;
```

Sisipkan Baris ke dalam Tabel untuk Memicu Notifikasi

```
mysql> insert into Customer_Feedback (customer_name, customer_feedback) VALUES ('Sample
 Customer', 'Good job guys!');
```

Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch

Anda dapat mengonfigurasi cluster DB MySQL Aurora Anda untuk mempublikasikan data log umum, lambat, audit, dan kesalahan ke grup log di Amazon Logs. CloudWatch Dengan CloudWatch Log, Anda dapat melakukan analisis real-time dari data log, dan menggunakannya CloudWatch untuk membuat alarm dan melihat metrik. Anda dapat menggunakan CloudWatch Log untuk menyimpan catatan log Anda dalam penyimpanan yang sangat tahan lama.

Untuk mempublikasikan CloudWatch log ke Log, masing-masing log harus diaktifkan. Log kesalahan diaktifkan secara default, tetapi Anda harus mengaktifkan jenis log lain secara eksplisit. Untuk informasi tentang cara mengaktifkan log dalam MySQL, lihat Selecting general query and slow query log output destinations dalam dokumentasi MySQL. Untuk informasi selengkapnya tentang mengaktifkan log audit Aurora MySQL, lihat Mengaktifkan Audit Lanjutan.

> **Note**
>
> Ketahui hal-hal berikut:
>
> - Anda tidak dapat mempublikasikan CloudWatch log ke Log untuk wilayah China (Ningxia).

- Jika pengeksporan data log dinonaktifkan, Aurora tidak menghapus grup log atau log stream yang ada. Jika mengekspor data log dinonaktifkan, data log yang ada tetap tersedia di CloudWatch Log, tergantung pada penyimpanan log, dan Anda masih dikenakan biaya untuk data log audit yang disimpan. Anda dapat menghapus aliran log dan grup log menggunakan konsol CloudWatch Log, APIAWS CLI, atau CloudWatch Logs.
- Cara alternatif untuk mempublikasikan log audit ke CloudWatch Log adalah dengan mengaktifkan Audit Lanjutan, lalu membuat grup parameter cluster DB kustom dan menyetel `server_audit_logs_upload` parameter ke. `1` Default untuk parameter klaster DB `server_audit_logs_upload` adalah `0`. Untuk informasi tentang mengaktifkan Audit Lanjutan, lihat Menggunakan Audit Lanjutan dengan klaster DB Amazon Aurora MySQL.

  Jika Anda menggunakan metode alternatif ini, Anda harus memiliki peran IAM untuk mengakses CloudWatch Log dan mengatur parameter `aws_default_logs_role` tingkat cluster ke ARN untuk peran ini. Untuk informasi tentang membuat peran, lihat Menyiapkan peran IAM untuk mengakses layanan AWS. Namun, jika Anda memiliki peran `AWSServiceRoleForRDS` terkait layanan, peran tersebut menyediakan akses ke CloudWatch Log dan mengganti peran yang ditentukan khusus. Untuk informasi tentang peran terkait layanan untuk Amazon RDS, lihat Menggunakan peran tertaut layanan untuk Amazon Aurora.
- Jika Anda tidak ingin mengekspor log audit ke CloudWatch Log, pastikan semua metode mengekspor log audit dinonaktifkan. Metode ini adalah AWS Management Console, AWS CLI, API RDS, dan parameter `server_audit_logs_upload`.
- Prosedurnya sedikit berbeda untuk klaster Aurora Serverless v1 daripada untuk klaster dengan instans terprovisi atau instans Aurora Serverless v2. Klaster Aurora Serverless v1 secara otomatis mengunggah semua jenis log yang Anda aktifkan melalui parameter konfigurasi. Oleh karena itu, Anda mengaktifkan atau menonaktifkan unggahan log untuk klaster Nirserver dengan mengaktifkan dan menonaktifkan jenis log di grup parameter klaster DB. Anda tidak mengubah pengaturan klaster itu sendiri melalui AWS Management Console, AWS CLI, atau API RDS. Untuk informasi tentang mengaktifkan dan menonaktifkan log MySQL untuk klaster Aurora Serverless v1, lihat Grup parameter untuk Aurora Serverless v1.

Konsol

Anda dapat menerbitkan log Aurora MySQL untuk klaster yang disediakan ke Log dengan konsol. CloudWatch

Untuk menerbitkan log Aurora MySQL dari konsol

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB Aurora MySQL yang data log-nya ingin Anda terbitkan.
4. Pilih Ubah.
5. Di bagian Log ekspor, pilih log yang ingin Anda mulai terbitkan ke CloudWatch Log.
6. Pilih Lanjutkan, lalu pilih Modifikasi Klaster DB di halaman ringkasan.

AWS CLI

Anda dapat menerbitkan log Aurora MySQL untuk klaster terprovisi dengan AWS CLI. Untuk melakukannya, Anda menjalankan modify-db-clusterAWS CLIperintah dengan opsi berikut:

- `--db-cluster-identifier` – Pengidentifikasi klaster DB.
- `--cloudwatch-logs-export-configuration`—Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk diekspor ke CloudWatch Log untuk cluster DB.

Anda juga dapat menerbitkan log Aurora MySQL dengan menjalankan salah satu dari perintah AWS CLI berikut:

- create-db-cluster
- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

Jalankan salah satu perintah AWS CLI ini dengan opsi berikut:

- `--db-cluster-identifier` – Pengidentifikasi klaster DB.
- `--engine` – Mesin basis data.
- `--enable-cloudwatch-logs-exports`—Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk diekspor ke CloudWatch Log untuk cluster DB.

Opsi lain mungkin diperlukan tergantung pada perintah AWS CLI yang Anda jalankan.

Example

Perintah berikut memodifikasi cluster Aurora MySQL DB yang ada untuk mempublikasikan file log ke Log. CloudWatch

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit"]}'
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":
["error","general","slowquery","audit"]}'
```

Example

Perintah berikut membuat cluster Aurora MySQL DB untuk mempublikasikan file log ke Log. CloudWatch

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine aurora \
    --enable-cloudwatch-logs-exports '["error","general","slowquery","audit"]'
```

Untuk Windows:

```
aws rds create-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine aurora ^
    --enable-cloudwatch-logs-exports '["error","general","slowquery","audit"]'
```

API RDS

Anda dapat menerbitkan log Aurora MySQL untuk klaster terprovisi dengan API RDS. Untuk melakukannya, Anda perlu menjalankan operasi ModifyDBCluster dengan opsi berikut:

- `DBClusterIdentifier` – Pengidentifikasi klaster DB.
- `CloudwatchLogsExportConfiguration`—Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk diekspor ke CloudWatch Log untuk cluster DB.

Anda juga dapat menerbitkan log Aurora MySQL menggunakan API RDS dengan menjalankan salah satu dari operasi API RDS berikut:

- CreateDBCluster
- DimemulihkanB S3 ClusterFrom
- DimemulihkanB ClusterFromSnapshot
- DimemulihkanB ClusterToPointInTime

Jalankan operasi API RDS dengan parameter berikut:

- `DBClusterIdentifier` – Pengidentifikasi klaster DB.
- `Engine` – Mesin basis data.
- `EnableCloudwatchLogsExports`—Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk diekspor ke CloudWatch Log untuk cluster DB.

Parameter lain mungkin diperlukan tergantung pada perintah AWS CLI yang Anda jalankan.

Memantau peristiwa log di Amazon CloudWatch

Setelah mengaktifkan peristiwa log Aurora MySQL, Anda dapat memantau peristiwa di Amazon Logs. CloudWatch Grup log baru secara otomatis dibuat untuk klaster DB Aurora menggunakan awalan berikut, dengan *`cluster-name`* merepresentasikan nama klaster DB, dan *`log_type`* merepresentasikan jenis log.

```
/aws/rds/cluster/cluster-name/log_type
```

Misalnya, jika Anda mengonfigurasi fungsi ekspor untuk menyertakan log kueri yang lambat untuk klaster DB bernama `mydbcluster`, data kueri yang lambat disimpan di grup log `/aws/rds/cluster/mydbcluster/slowquery`.

Peristiwa dari semua instans di klaster Anda didorong ke grup log menggunakan log stream yang berbeda-beda. Perilakunya bergantung pada mana yang berlaku dari kondisi berikut ini:

- Grup log dengan nama yang ditentukan tersedia.

  Aurora menggunakan grup log yang ada untuk mengekspor data log untuk klaster. Untuk membuat grup log dengan periode retensi log, filter metrik, dan akses pelanggan standar, Anda dapat menggunakan konfigurasi otomatis, seperti AWS CloudFormation.
- Grup log dengan nama yang ditentukan tidak ada.

  Ketika entri log yang cocok terdeteksi dalam file log untuk contoh, Aurora MySQL membuat grup log baru di Log secara otomatis. CloudWatch Grup log menggunakan periode retensi log default Tidak Pernah Kedaluwarsa.

  Untuk mengubah periode penyimpanan log, gunakan konsol CloudWatch Log, APIAWS CLI, atau CloudWatch Logs. Untuk informasi selengkapnya tentang mengubah periode penyimpanan CloudWatch log di Log, lihat Mengubah penyimpanan data CloudWatch log di Log.

Untuk mencari informasi dalam peristiwa log untuk klaster DB, gunakan konsol CloudWatch Log, APIAWS CLI, atau CloudWatch Logs. Untuk informasi selengkapnya tentang mencari dan memfilter data log, lihat Menelusuri dan memfilter data log.

Mode lab Amazon Aurora MySQL

Mode lab Aurora digunakan untuk mengaktifkan fitur Aurora yang tersedia dalam versi basis data Aurora saat ini, tetapi tidak diaktifkan secara default. Meskipun fitur mode lab Aurora tidak disarankan untuk digunakan dalam klaster DB produksi, Anda dapat menggunakan mode lab Aurora untuk mengaktifkan fitur ini untuk klaster DB di lingkungan pengembangan dan pengujian Anda. Untuk informasi selengkapnya tentang fitur Aurora yang tersedia saat mode lab Aurora diaktifkan, lihat Fitur mode lab Aurora.

Parameter `aurora_lab_mode` adalah parameter tingkat instans yang berada dalam grup parameter default. Parameter ini diatur ke `0` (dinonaktifkan) dalam grup parameter default. Untuk mengaktifkan mode lab Aurora, buat grup parameter kustom, atur parameter `aurora_lab_mode` ke `1` (diaktifkan) dalam grup parameter kustom tersebut, dan ubah satu atau beberapa instans DB dalam klaster Aurora Anda agar menggunakan grup parameter kustom tersebut. Kemudian, hubungkan ke titik akhir instans yang sesuai untuk mencoba fitur mode lab. Untuk informasi tentang mengubah grup parameter DB, lihat Memodifikasi parameter dalam grup parameter DB. Untuk informasi tentang grup parameter dan Amazon Aurora, lihat Parameter konfigurasi Aurora MySQL.

Fitur mode lab Aurora

Tabel berikut mencantumkan fitur Aurora yang saat ini tersedia saat mode lab Aurora diaktifkan. Anda harus mengaktifkan mode lab Aurora sebelum salah satu fitur ini dapat digunakan.

Fitur	Deskripsi
Batching Pemindaian	Batching pemindaian Aurora MySQL mempercepat kueri dalam memori berorient asi pemindaian secara signifikan. Fitur ini meningkatkan performa pemindaian lengkap tabel, pemindaian lengkap indeks, dan pemindaian rentang indeks berdasarkan pemrosesan batch.
Hash Join	Fitur ini dapat meningkatkan performa kueri saat Anda perlu menggabungkan data dalam jumlah besar dengan menggunakan equijoin. Anda dapat menggunakan fitur ini tanpa mode lab di Aurora MySQL versi 2. Untuk informasi

Fitur	Deskripsi
	selengkapnya tentang penggunaan fitur ini, lihat Mengoptimalkan kueri join MySQL Aurora besar dengan hash join.
DDL Cepat	Fitur ini memungkinkan Anda menjalankan operasi `ALTER TABLE tbl_name ADD COLUMN col_name column_definition` hampir seketika. Operasi tersebut akan diselesaikan tanpa perlu menyalin tabel dan tanpa memengaruhi pernyataan DML lainnya secara signifikan. Karena tidak menggunakan penyimpanan sementara untuk salinan tabel, pernyataan DDL menjadi praktis bahkan untuk tabel besar di kelas instans kecil. DDL Cepat saat ini hanya mendukung penambahan kolom yang dapat dibuat null, tanpa nilai default, di akhir tabel. Untuk informasi selengkapnya tentang penggunaan fitur ini, lihat Mengubah tabel di Amazon Aurora menggunakan DDL Cepat.

Praktik terbaik dengan Amazon Aurora MySQL

Topik ini mencakup informasi tentang praktik terbaik dan opsi untuk menggunakan atau memigrasi data ke klaster DB Amazon Aurora MySQL. Informasi dalam topik ini merangkum dan mengulangi beberapa pedoman dan prosedur yang dapat Anda temukan di Mengelola klaster DB Amazon Aurora.

Daftar Isi

- Menentukan ke instans DB mana Anda terhubung
- Praktik terbaik untuk performa dan penskalaan Aurora MySQL
  - Menggunakan kelas instans T untuk pengembangan dan pengujian
  - Mengoptimalkan kueri join yang diindeks MySQL Aurora dengan prefetch kunci asinkron
    - Mengaktifkan prefetch kunci asinkron
    - Mengoptimalkan kueri untuk prefetch kunci asinkron
  - Mengoptimalkan kueri join MySQL Aurora besar dengan hash join
    - Mengaktifkan hash join
    - Mengoptimalkan kueri untuk hash join
  - Menggunakan Amazon Aurora untuk menskalakan baca untuk basis data MySQL Anda
  - Mengoptimalkan operasi stempel waktu
- Praktik terbaik untuk ketersediaan tinggi Aurora MySQL
  - Menggunakan Amazon Aurora untuk Pemulihan Bencana dengan basis data MySQL Anda
  - Bermigrasi dari MySQL ke Amazon Aurora MySQL dengan waktu henti yang lebih singkat
  - Menghindari performa lambat, pengaktifan ulang otomatis, dan failover untuk instans Aurora MySQL DB
- Rekomendasi untuk Aurora MySQL
  - Menggunakan replikasi multithread di Aurora MySQL versi 3
  - Memanggil AWS Lambda fungsi menggunakan fungsi MySQL asli
  - Menghindari transaksi XA dengan Amazon Aurora MySQL
  - Mempertahankan kunci asing tetap aktif selama pernyataan DML
  - Mengonfigurasi seberapa sering buffer log di-flush
  - Meminimalkan dan memecahkan masalah deadlock Aurora MySQL
  - Meminimalkan deadlock InnoDB

- Memantau deadlock InnoDB

Menentukan ke instans DB mana Anda terhubung

Untuk menentukan ke instans DB mana sebuah koneksi terhubung di klaster DB Aurora MySQL, periksa variabel global `innodb_read_only`, seperti yang ditunjukkan dalam contoh berikut.

```
SHOW GLOBAL VARIABLES LIKE 'innodb_read_only';
```

Variabel `innodb_read_only` diatur ke `0N` jika Anda terhubung ke instans DB pembaca. Pengaturan ini berstatus `0FF` jika Anda terhubung ke instans DB penulis, seperti instans primer dalam klaster terprovisi.

Pendekatan ini dapat membantu jika Anda ingin menambahkan logika ke kode aplikasi Anda untuk menyeimbangkan beban kerja atau untuk memastikan bahwa operasi tulis menggunakan koneksi yang tepat.

Praktik terbaik untuk performa dan penskalaan Aurora MySQL

Anda dapat menerapkan praktik terbaik berikut untuk meningkatkan performa dan skalabilitas klaster Aurora MySQL Anda.

Topik

- Menggunakan kelas instans T untuk pengembangan dan pengujian
- Mengoptimalkan kueri join yang diindeks MySQL Aurora dengan prefetch kunci asinkron
- Mengoptimalkan kueri join MySQL Aurora besar dengan hash join
- Menggunakan Amazon Aurora untuk menskalakan baca untuk basis data MySQL Anda
- Mengoptimalkan operasi stempel waktu

Menggunakan kelas instans T untuk pengembangan dan pengujian

Instans Amazon Aurora MySQL yang menggunakan kelas instans DB `db.t2`, `db.t3` atau `db.t4g` paling cocok untuk aplikasi yang tidak mendukung beban kerja yang tinggi untuk waktu yang lama. Instans T dirancang untuk memberikan performa dasar sedang dan kemampuan untuk melakukan burst performa yang secara signifikan lebih tinggi sesuai dengan yang dibutuhkan beban kerja Anda.

Instans ini ditujukan untuk beban kerja yang tidak menggunakan CPU penuh secara sering atau konsisten, tetapi kadang-kadang memerlukan burst. Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Instans performa yang dapat melonjak.

Jika klaster Aurora Anda lebih besar dari 40 TB, jangan gunakan kelas instans T. Ketika basis data Anda memiliki volume data yang besar, overhead memori untuk mengelola objek skema dapat melebihi kapasitas instans T.

Jangan aktifkan Skema Performa MySQL pada instans T Amazon Aurora MySQL. Jika Skema Performa diaktifkan, instans ini dapat kehabisan memori.

> **Tip**
>
> Jika basis data Anda terkadang idle, tetapi di lain waktu memiliki beban kerja yang besar, Anda dapat menggunakan Aurora Serverless v2 sebagai alternatif untuk instans T. Dengan Aurora Serverless v2, Anda menentukan rentang kapasitas dan Aurora akan secara otomatis menaikkan atau menurunkan skala basis data Anda tergantung pada beban kerja saat ini. Untuk detail penggunaan, lihat Menggunakan Aurora Serverless v2. Untuk versi mesin basis data yang dapat Anda gunakan dengan Aurora Serverless v2, lihat Persyaratan dan batasan untuk Aurora Serverless v2.

Saat Anda menggunakan instans T sebagai instans DB di klaster DB Aurora MySQL kami menyarankan hal berikut:

- Gunakan kelas instans DB yang sama untuk semua instans di klaster DB Anda. Misalnya, jika Anda menggunakan `db.t2.medium` untuk instans penulis Anda, maka kami sarankan Anda menggunakan `db.t2.medium` untuk instans pembaca Anda juga.
- Jangan sesuaikan pengaturan konfigurasi terkait memori apa pun, seperti `innodb_buffer_pool_size`. Aurora menggunakan serangkaian nilai default yang sudah sangat disesuaikan untuk buffer memori pada instans T. Default khusus ini diperlukan agar Aurora dapat berjalan pada instans yang dibatasi memori. Jika Anda mengubah pengaturan terkait memori pada instance T, Anda jauh lebih mungkin menghadapi out-of-memory kondisi, bahkan jika perubahan Anda dimaksudkan untuk meningkatkan ukuran buffer.
- Pantau Saldo Kredit CPU Anda (`CPUCreditBalance`) untuk memastikannya berada pada tingkat yang layak. Artinya, kredit CPU diakumulasi pada kecepatan yang sama dengan yang digunakan.

Ketika Anda telah menghabiskan kredit CPU untuk instans, Anda akan mengalami penurunan yang cepat pada CPU yang tersedia serta peningkatan latensi baca dan tulis untuk instans. Situasi ini menyebabkan penurunan yang parah dalam performa keseluruhan instans.

Jika saldo kredit CPU Anda tidak berada pada tingkat yang layak, maka kami menyarankan untuk memodifikasi instans DB Anda agar menggunakan salah satu kelas instans DB R yang didukung (penskalaan komputasi).

Untuk informasi selengkapnya tentang pemantauan metrik, lihat Melihat metrik di konsol Amazon RDS.

- Pantau lag replika (`AuroraReplicaLag`) antara instans penulis dan instans pembaca.

  Jika instans pembaca kehabisan kredit CPU sebelum instans penulis, lag yang dihasilkan dapat menyebabkan instans pembaca sering diaktifkan ulang. Hasil ini biasa terjadi jika aplikasi memiliki beban operasi baca berat yang didistribusikan di antara instans pembaca, pada waktu yang sama saat instans penulis memiliki beban operasi tulis minimal.

  Jika Anda melihat peningkatan berkelanjutan dalam lag replika, pastikan saldo kredit CPU Anda untuk instans pembaca di klaster DB Anda tidak habis.

  Jika saldo kredit CPU Anda tidak berada pada tingkat yang layak, maka kami menyarankan untuk memodifikasi instans DB Anda agar menggunakan salah satu kelas instans DB R yang didukung (penskalaan komputasi).

- Pertahankan jumlah penyisipan per transaksi di bawah 1 juta untuk klaster DB yang memiliki pencatatan log biner.

  Jika grup parameter cluster DB untuk cluster DB Anda memiliki `binlog_format` parameter yang disetel ke nilai selain`OFF`, maka cluster DB Anda mungkin mengalami out-of-memory kondisi jika cluster DB menerima transaksi yang berisi lebih dari 1 juta baris untuk disisipkan. Anda dapat memantau metrik memori yang dapat dibebaskan (`FreeableMemory`) untuk menentukan apakah klaster DB Anda kehabisan memori. Kemudian, Anda memeriksa metrik operasi tulis (`VolumeWriteIOPS`) untuk melihat apakah instans penulis menerima beban berat dari operasi tulis. Jika demikian, kami menyarankan agar Anda memperbarui aplikasi Anda untuk membatasi jumlah penyisipan dalam transaksi hingga kurang dari 1 juta. Alternatifnya, Anda dapat memodifikasi instans Anda untuk menggunakan salah satu dari kelas instans DB R yang didukung (penskalaan komputasi).

Mengoptimalkan kueri join yang diindeks MySQL Aurora dengan prefetch kunci asinkron

Aurora MySQL dapat menggunakan fitur prefetch kunci asinkron (AKP) untuk meningkatkan performa kueri yang menggabungkan tabel di seluruh indeks. Fitur ini meningkatkan performa dengan mengantisipasi baris yang diperlukan untuk menjalankan kueri yang mengharuskan kueri JOIN menggunakan fitur optimisasi Batched Key Access (BKA) dan Multi-Range Read (MRR). Untuk informasi selengkapnya tentang BKA dan MRR, lihat Block nested-loop and batched key access joins dan Multi-range read optimization dalam dokumentasi MySQL.

Untuk memanfaatkan fitur AKP, kueri harus menggunakan BKA dan MRR. Biasanya, kueri tersebut terjadi saat klausa JOIN dari kueri menggunakan indeks sekunder, tetapi juga memerlukan beberapa kolom dari indeks primer. Misalnya, Anda dapat menggunakan AKP ketika klausa JOIN merepresentasikan equijoin pada nilai indeks antara tabel luar kecil dan tabel dalam besar, dan indeksnya sangat selektif pada tabel yang lebih besar. AKP beroperasi bersama dengan BKA dan MRR untuk melakukan pencarian indeks sekunder hingga primer selama evaluasi klausa JOIN. AKP mengidentifikasi baris yang diperlukan untuk menjalankan kueri selama evaluasi klausa JOIN. Kemudian, fitur ini menggunakan thread latar belakang untuk secara asinkron memuat halaman yang berisi baris tersebut ke dalam memori sebelum menjalankan kueri.

AKP tersedia untuk Aurora MySQL versi 2.10 dan lebih tinggi, serta versi 3. Untuk informasi selengkapnya tentang versi Aurora MySQL, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL.

Mengaktifkan prefetch kunci asinkron

Anda dapat mengaktifkan fitur AKP dengan mengatur `aurora_use_key_prefetch`, sebuah variabel server MySQL, ke `on`. Secara default, nilai ini diatur ke `on`. Namun, AKP tidak dapat diaktifkan sampai Anda juga mengaktifkan algoritma BKA Join dan menonaktifkan fungsionalitas MRR berbasis biaya. Untuk melakukannya, Anda harus menetapkan nilai berikut untuk `optimizer_switch`, sebuah variabel server MySQL:

- Atur `batched_key_access` ke `on`. Nilai ini mengontrol penggunaan algoritma BKA Join. Secara default, nilai ini diatur ke `off`.
- Atur `mrr_cost_based` ke `off`. Nilai ini mengontrol penggunaan fungsionalitas MRR berbasis biaya. Secara default, nilai ini diatur ke `on`.

Saat ini, Anda dapat mengatur nilai ini hanya di tingkat sesi. Contoh berikut menggambarkan cara menetapkan nilai-nilai ini untuk mengaktifkan AKP untuk sesi saat ini dengan mengeksekusi pernyataan SET.

```
mysql> set @@session.aurora_use_key_prefetch=on;
mysql> set @@session.optimizer_switch='batched_key_access=on,mrr_cost_based=off';
```

Demikian pula, Anda dapat menggunakan pernyataan SET untuk menonaktifkan AKP dan algoritma BKA Join serta mengaktifkan ulang fungsi MRR berbasis biaya untuk sesi saat ini, seperti yang ditunjukkan dalam contoh berikut.

```
mysql> set @@session.aurora_use_key_prefetch=off;
mysql> set @@session.optimizer_switch='batched_key_access=off,mrr_cost_based=on';
```

Untuk informasi selengkapnya tentang switch pengoptimisasi batched_key_access dan mrr_cost_based lihat Switchable optimizations dalam dokumentasi MySQL.

Mengoptimalkan kueri untuk prefetch kunci asinkron

Anda dapat mengonfirmasi apakah kueri dapat memanfaatkan fitur AKP. Untuk melakukannya, gunakan pernyataan `EXPLAIN` untuk membuat profil kueri sebelum menjalankannya. Pernyataan `EXPLAIN` memberikan informasi tentang rencana eksekusi yang akan digunakan untuk kueri tertentu.

Pada output pernyataan `EXPLAIN`, kolom `Extra` menjelaskan informasi tambahan, termasuk rencana eksekusi. Jika fitur AKP berlaku pada tabel yang digunakan dalam kueri, kolom ini mencakup salah satu nilai berikut:

- `Using Key Prefetching`
- `Using join buffer (Batched Key Access with Key Prefetching)`

Contoh berikut menunjukkan penggunaan `EXPLAIN` untuk melihat rencana eksekusi untuk kueri yang dapat memanfaatkan AKP.

```
mysql> explain select sql_no_cache
    ->     ps_partkey,
    ->     sum(ps_supplycost * ps_availqty) as value
    -> from
```

```
    ->     partsupp,
    ->     supplier,
    ->     nation
    -> where
    ->     ps_suppkey = s_suppkey
    ->     and s_nationkey = n_nationkey
    ->     and n_name = 'ETHIOPIA'
    -> group by
    ->     ps_partkey having
    ->         sum(ps_supplycost * ps_availqty) > (
    ->             select
    ->                 sum(ps_supplycost * ps_availqty) * 0.0000003333
    ->             from
    ->                 partsupp,
    ->                 supplier,
    ->                 nation
    ->             where
    ->                 ps_suppkey = s_suppkey
    ->                 and s_nationkey = n_nationkey
    ->                 and n_name = 'ETHIOPIA'
    ->         )
    -> order by
    ->     value desc;
+----+-------------+----------+------+-----------------------+---------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
| id | select_type | table    | type | possible_keys         | key           | key_len
 | ref                               | rows | filtered | Extra
                              |
+----+-------------+----------+------+-----------------------+---------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
|  1 | PRIMARY     | nation   | ALL  | PRIMARY               | NULL          | NULL
 | NULL                              |   25 |   100.00 | Using where; Using temporary;
 Using filesort                |
|  1 | PRIMARY     | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                              |
|  1 | PRIMARY     | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
|  2 | SUBQUERY    | nation   | ALL  | PRIMARY               | NULL          | NULL
 | NULL                              |   25 |   100.00 | Using where
                              |
```

```
|  2 | SUBQUERY    | supplier | ref  | PRIMARY,i_s_nationkey | i_s_nationkey | 5
 | dbt3_scale_10.nation.n_nationkey | 2057 |   100.00 | Using index
                              |
|  2 | SUBQUERY    | partsupp | ref  | i_ps_suppkey          | i_ps_suppkey  | 4
 | dbt3_scale_10.supplier.s_suppkey |   42 |   100.00 | Using join buffer (Batched Key
 Access with Key Prefetching) |
+----+-------------+----------+------+-----------------------+---------------
+---------+-----------------------------------+------+----------
+-------------------------------------------------------------+
6 rows in set, 1 warning (0.00 sec)
```

Untuk informasi selengkapnya tentang format output `EXPLAIN`, lihat Extended EXPLAIN output format dalam dokumentasi MySQL.

Mengoptimalkan kueri join MySQL Aurora besar dengan hash join

Saat Anda harus menggabungkan sejumlah besar data dengan menggunakan equijoin, hash join dapat meningkatkan performa kueri. Anda dapat mengaktifkan hash join untuk Aurora MySQL.

Kolom hash join dapat berupa ekspresi rumit apa pun. Dalam kolom hash join, Anda dapat membandingkan jenis data dengan cara berikut:

- Anda dapat membandingkan apa pun dalam kategori jenis data numerik yang tepat, seperti `int`, `bigint`, `numeric`, dan `bit`.
- Anda dapat membandingkan apa pun dalam kategori perkiraan jenis data numerik, seperti `float` dan `double`.
- Anda dapat membandingkan item di seluruh jenis string jika jenis string memiliki kumpulan karakter dan kolasi yang sama.
- Anda dapat membandingkan item dengan jenis data tanggal dan stempel waktu jika jenisnya sama.

> **Note**
>
> Anda tidak dapat membandingkan jenis data dalam kategori yang berbeda.

Batasan berikut berlaku untuk hash join untuk Aurora MySQL:

- Outer join kiri-kanan tidak didukung untuk Aurora MySQL versi 2, tetapi didukung untuk versi 3.

- Semijoin seperti subkueri tidak didukung, kecuali jika subkueri dimaterialkan terlebih dahulu.
- Pembaruan atau penghapusan multi-tabel tidak didukung.

> **Note**
>
> Pembaruan atau penghapusan satu tabel didukung.

- Kolom jenis data spasial dan BLOB tidak dapat menjadi kolom join dalam hash join.

Mengaktifkan hash join

Untuk mengaktifkan hash join:

- Aurora MySQL versi 2 - Atur parameter DB atau parameter klaster DB `aurora_disable_hash_join` ke `0`. Menonaktifkan `aurora_disable_hash_join` akan menetapkan nilai `optimizer_switch` ke `hash_join=on`.
- Aurora MySQL versi 3 - Atur parameter server MySQL `optimizer_switch` ke `block_nested_loop=on`.

Hash join diaktifkan secara default di Aurora MySQL versi 3 dan dinonaktifkan secara default di Aurora MySQL versi 2. Contoh berikut mengilustrasikan cara mengaktifkan hash join untuk Aurora MySQL versi 3. Anda dapat mengeluarkan pernyataan `select @@optimizer_switch` terlebih dahulu untuk melihat apa saja pengaturan lain yang ada dalam string parameter `SET`. Memperbarui satu pengaturan dalam parameter `optimizer_switch` tidak akan menghapus atau memodifikasi pengaturan lainnya.

```
mysql> SET optimizer_switch='block_nested_loop=on';
```

> **Note**
>
> Untuk Aurora MySQL versi 3, dukungan hash join tersedia di semua versi minor dan diaktifkan secara default.
> Untuk Aurora MySQL versi 2, dukungan hash join tersedia di semua versi minor. Di Aurora MySQL versi 2, fitur hash join selalu dikendalikan oleh nilai `aurora_disable_hash_join`.

Dengan pengaturan ini, pengoptimisasi memilih untuk menggunakan hash join berdasarkan biaya, karakteristik kueri, dan ketersediaan sumber daya. Jika estimasi biaya salah, Anda dapat memaksa pengoptimisasi untuk memilih hash join. Caranya adalah dengan mengatur `hash_join_cost_based`, sebuah variabel server MySQL, ke `off`. Contoh berikut mengilustrasikan cara memaksa pengoptimisasi untuk memilih hash join.

```
mysql> SET optimizer_switch='hash_join_cost_based=off';
```

> **Note**
>
> Pengaturan ini mengganti keputusan pengoptimisasi berbasis biaya. Meskipun pengaturan ini dapat berguna untuk pengujian dan pengembangan, kami menyarankan Anda untuk tidak menggunakannya dalam produksi.

Mengoptimalkan kueri untuk hash join

Untuk mengetahui apakah kueri dapat memanfaatkan hash join, gunakan pernyataan `EXPLAIN` untuk membuat profil kueri terlebih dahulu. Pernyataan `EXPLAIN` memberikan informasi tentang rencana eksekusi yang akan digunakan untuk kueri tertentu.

Pada output pernyataan `EXPLAIN`, kolom `Extra` menjelaskan informasi tambahan, termasuk rencana eksekusi. Jika hash join berlaku pada tabel yang digunakan dalam kueri, kolom ini akan mencakup nilai seperti yang berikut ini:

- `Using where; Using join buffer (Hash Join Outer table` *`table1_name`*`)`
- `Using where; Using join buffer (Hash Join Inner table` *`table2_name`*`)`

Contoh berikut menunjukkan penggunaan EXPLAIN untuk melihat rencana eksekusi untuk kueri hash join.

```
mysql> explain SELECT sql_no_cache * FROM hj_small, hj_big, hj_big2
    ->     WHERE hj_small.col1 = hj_big.col1 and hj_big.col1=hj_big2.col1 ORDER BY 1;
+----+-------------+----------+------+---------------+------+---------+------+------
+----------------------------------------------------------------+
| id | select_type | table    | type | possible_keys | key  | key_len | ref  | rows |
 Extra                                                          |
```

```
+----+-------------+----------+------+---------------+------+---------+------+------
+-------------------------------------------------------------------+
|  1 | SIMPLE      | hj_small | ALL  | NULL          | NULL | NULL    | NULL |    6 |
 Using temporary; Using filesort                                    |
|  1 | SIMPLE      | hj_big   | ALL  | NULL          | NULL | NULL    | NULL |   10 |
 Using where; Using join buffer (Hash Join Outer table hj_big)  |
|  1 | SIMPLE      | hj_big2  | ALL  | NULL          | NULL | NULL    | NULL |   15 |
 Using where; Using join buffer (Hash Join Inner table hj_big2) |
+----+-------------+----------+------+---------------+------+---------+------+------
+-------------------------------------------------------------------+
3 rows in set (0.04 sec)
```

Dalam output ini, `Hash Join Inner table` adalah tabel yang digunakan untuk membuat tabel hash, dan `Hash Join Outer table` adalah tabel yang digunakan untuk menyelidiki tabel hash.

Untuk informasi selengkapnya tentang format output `EXPLAIN` yang diperluas, lihat Extended EXPLAIN Output Format dalam dokumentasi produk MySQL.

Di Aurora MySQL 2.08 dan yang lebih tinggi, Anda dapat menggunakan petunjuk SQL untuk memengaruhi apakah kueri menggunakan hash join atau tidak, dan tabel mana yang digunakan untuk sisi build dan probe dari join. Untuk detailnya, lihat Petunjuk Aurora MySQL.

Menggunakan Amazon Aurora untuk menskalakan baca untuk basis data MySQL Anda

Anda dapat menggunakan Amazon Aurora dengan instans DB MySQL Anda untuk memanfaatkan kemampuan penskalaan baca Amazon Aurora dan memperluas beban kerja baca untuk instans DB MySQL Anda. Untuk menggunakan Aurora untuk menskalakan baca instans DB MySQL Anda, buat klaster DB Aurora MySQL dan jadikan sebagai replika baca dari instans DB MySQL Anda. Kemudian, hubungkan ke klaster Aurora MySQL untuk memproses kueri baca. Basis data sumber dapat berupa instans DB RDS for MySQL, atau sebuah basis data MySQL yang dijalankan secara eksternal di luar Amazon RDS. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora untuk menskalakan baca untuk basis data MySQL Anda.

Mengoptimalkan operasi stempel waktu

Ketika nilai variabel sistem `time_zone` diatur ke `SYSTEM`, setiap panggilan fungsi MySQL yang memerlukan perhitungan zona waktu akan membuat panggilan pustaka sistem. Saat Anda menjalankan pernyataan SQL yang menampilkan atau mengubah nilai `TIMESTAMP` tersebut pada konkurensi tinggi, Anda mungkin akan mengalami latensi, pertentangan kunci, dan penggunaan CPU yang meningkat. Untuk informasi selengkapnya, lihat time_zone dalam dokumentasi MySQL.

Untuk menghindari perilaku ini, sebaiknya Anda mengubah nilai parameter klaster DB `time_zone` menjadi `UTC`. Untuk informasi selengkapnya, lihat Mengubah parameter dalam grup parameter klaster DB.

Meskipun parameter `time_zone` bersifat dinamis (tidak memerlukan pengaktifan ulang server basis data), nilai baru hanya digunakan untuk koneksi baru. Untuk memastikan bahwa semua koneksi diperbarui untuk menggunakan nilai `time_zone` baru, kami sarankan Anda mendaur ulang koneksi aplikasi Anda setelah memperbarui parameter klaster DB.

Praktik terbaik untuk ketersediaan tinggi Aurora MySQL

Anda dapat menerapkan praktik terbaik berikut untuk meningkatkan ketersediaan klaster Aurora MySQL Anda.

Topik

- Menggunakan Amazon Aurora untuk Pemulihan Bencana dengan basis data MySQL Anda
- Bermigrasi dari MySQL ke Amazon Aurora MySQL dengan waktu henti yang lebih singkat
- Menghindari performa lambat, pengaktifan ulang otomatis, dan failover untuk instans Aurora MySQL DB

Menggunakan Amazon Aurora untuk Pemulihan Bencana dengan basis data MySQL Anda

Anda dapat menggunakan Amazon Aurora dengan instans DB MySQL Anda untuk membuat cadangan di luar lokasi untuk pemulihan bencana. Untuk menggunakan Aurora untuk pemulihan bencana instans DB MySQL Anda, buat klaster DB Amazon Aurora dan jadikan sebagai replika baca instans DB MySQL Anda. Hal ini berlaku untuk satu instans DB RDS for MySQL, atau basis data MySQL yang dijalankan secara eksternal di luar Amazon RDS.

> **Important**
>
> Saat Anda mengatur replikasi i antara instans DB MySQL dan klaster DB Amazon Aurora MySQL, Anda harus memantau replikasi untuk memastikan kondisinya tetap baik dan memperbaikinya jika perlu.

Untuk petunjuk tentang cara membuat klaster DB Amazon Aurora MySQL dan menjadikannya sebagai replika baca dari instans DB MySQL Anda, ikuti prosedur dalam Menggunakan Amazon Aurora untuk menskalakan baca untuk basis data MySQL Anda.

Untuk informasi selengkapnya tentang model pemulihan bencana, lihat Cara memilih opsi pemulihan bencana terbaik untuk klaster MySQL Amazon Aurora Anda.

Bermigrasi dari MySQL ke Amazon Aurora MySQL dengan waktu henti yang lebih singkat

Saat mengimpor data dari basis data MySQL yang mendukung aplikasi aktif ke klaster DB Amazon Aurora MySQL, Anda mungkin ingin mengurangi waktu gangguan layanan saat Anda bermigrasi. Untuk melakukannya, Anda dapat menggunakan prosedur yang didokumentasikan dalam Mengimpor data ke instans MySQL atau MariaDB DB dengan waktu henti yang lebih singkat dalam Panduan Pengguna Amazon Relational Database Service. Prosedur ini terutama dapat membantu jika Anda beroperasi dengan basis data yang sangat besar. Anda dapat menggunakan prosedur ini untuk mengurangi biaya impor dengan meminimalkan jumlah data yang diteruskan melalui jaringan ke AWS.

Prosedur ini mencantumkan langkah-langkah untuk mentransfer salinan data basis data Anda ke instans Amazon EC2 dan mengimpor data ke instans DB RDS for MySQL baru. Karena Amazon Aurora kompatibel dengan MySQL, Anda dapat menggunakan klaster DB Amazon Aurora untuk instans DB Amazon RDS MySQL target.

Menghindari performa lambat, pengaktifan ulang otomatis, dan failover untuk instans Aurora MySQL DB

Jika Anda menjalankan beban kerja berat atau beban kerja yang melonjak melampaui sumber daya yang dialokasikan untuk instans DB Anda, Anda dapat menghabiskan sumber daya yang Anda gunakan untuk menjalankan aplikasi dan basis data Aurora. Untuk mendapatkan metrik pada instans database Anda seperti pemanfaatan CPU, penggunaan memori, dan jumlah koneksi database yang digunakan, Anda dapat merujuk ke metrik yang disediakan oleh Amazon, Performance CloudWatch Insights, dan Enhanced Monitoring. Untuk informasi selengkapnya tentang pemantauan instans DB, lihat Memantau metrik di klaster Amazon Aurora.

Jika beban kerja Anda menghabiskan sumber daya yang Anda gunakan, instans DB Anda mungkin akan menjadi lambat, diaktifkan ulang, atau bahkan melakukan failover ke instans DB lain. Untuk menghindari hal ini, pantau pemanfaatan sumber daya Anda, periksa beban kerja yang berjalan pada

instans DB Anda, dan buat pengoptimalan jika diperlukan. Jika pengoptimalan tidak meningkatkan metrik instans dan mengurangi kehabisan sumber daya, pertimbangkan untuk menaikkan skala instans DB Anda sebelum mencapai batasnya. Untuk informasi selengkapnya tentang kelas instans DB yang tersedia dan spesifikasinya, lihat Kelas instans DB Aurora.

Rekomendasi untuk Aurora MySQL

Fitur-fitur berikut tersedia di Aurora MySQL untuk kompatibilitas MySQL. Namun, fitur-fitur ini memiliki masalah performa, skalabilitas, stabilitas, atau kompatibilitas di lingkungan Aurora. Oleh karena itu, kami menyarankan Anda mengikuti pedoman tertentu dalam penggunaan fitur-fitur ini. Misalnya, kami sarankan Anda tidak menggunakan fitur tertentu untuk deployment Aurora produksi.

Topik

- Menggunakan replikasi multithread di Aurora MySQL versi 3
- Memanggil AWS Lambda fungsi menggunakan fungsi MySQL asli
- Menghindari transaksi XA dengan Amazon Aurora MySQL
- Mempertahankan kunci asing tetap aktif selama pernyataan DML
- Mengonfigurasi seberapa sering buffer log di-flush
- Meminimalkan dan memecahkan masalah deadlock Aurora MySQL

Menggunakan replikasi multithread di Aurora MySQL versi 3

Secara default, Aurora menggunakan replikasi thread tunggal saat klaster DB Aurora MySQL digunakan sebagai replika baca untuk replikasi log biner.

Meskipun Aurora MySQL tidak melarang replikasi multithread, fitur ini hanya didukung di Aurora MySQL versi 3.

Aurora MySQL versi 2 mewarisi beberapa masalah terkait replikasi multithread dari MySQL. Untuk versi tersebut, kami sarankan Anda tidak menggunakan replikasi multithread dalam produksi.

Jika Anda menggunakan replikasi multithread, kami sarankan agar Anda menguji setiap penggunaan secara menyeluruh.

Untuk informasi selengkapnya tentang replikasi di Amazon Aurora, lihat Replikasi dengan Amazon Aurora. Untuk informasi tentang replikasi multithread di Aurora MySQL versi 3, lihat Replikasi log biner multithreaded (Aurora MySQL versi 3).

Memanggil AWS Lambda fungsi menggunakan fungsi MySQL asli

Sebaiknya gunakan fungsi MySQL native `lambda_sync` dan `lambda_async` untuk menginvokasi fungsi Lambda.

Jika Anda menggunakan prosedur `mysql.lambda_async` yang sudah dihentikan, kami sarankan agar Anda menggabungkan panggilan ke prosedur `mysql.lambda_async` dalam prosedur tersimpan. Anda dapat memanggil prosedur tersimpan ini dari berbagai sumber, seperti pemicu atau kode klien. Pendekatan ini dapat membantu menghindari masalah ketidakcocokan impedansi dan memudahkan pemrogram basis data Anda untuk menginvokasi fungsi Lambda.

Untuk informasi selengkapnya tentang menginvokasi fungsi Lambda dari Amazon Aurora, lihat Menginvokasi fungsi Lambda dari klaster DB Amazon Aurora MySQL.

Menghindari transaksi XA dengan Amazon Aurora MySQL

Sebaiknya jangan gunakan transaksi eXtended Architecture (XA) dengan Aurora MySQL karena dapat menyebabkan waktu pemulihan yang lama jika XA berada di status `PREPARED`. Jika Anda harus menggunakan transaksi XA dengan Aurora MySQL, ikuti praktik terbaik ini:

- Jangan tinggalkan transaksi XA terbuka pada status `PREPARED`.
- Pertahankan transaksi XA sekecil mungkin.

Untuk informasi selengkapnya tentang menggunakan transaksi XA dengan MySQL, lihat XA transactions dalam dokumentasi MySQL.

Mempertahankan kunci asing tetap aktif selama pernyataan DML

Kami sangat menyarankan agar Anda tidak menjalankan pernyataan bahasa definisi data (DDL) saat variabel `foreign_key_checks` diatur menjadi `0` (nonaktif).

Jika Anda perlu menyisipkan atau memperbarui baris yang memerlukan pelanggaran sementara terhadap kunci asing, ikuti langkah-langkah berikut:

1. Atur `foreign_key_checks` ke `0`.
2. Membuat perubahan bahasa manipulasi data (DML).
3. Pastikan bahwa perubahan yang telah Anda selesaikan tidak melanggar batasan kunci asing.
4. Atur `foreign_key_checks` ke `1` (aktif).

Selain itu, ikuti praktik terbaik lainnya untuk batasan kunci asing:

- Pastikan bahwa aplikasi klien tidak mengatur variabel `foreign_key_checks` ke `0` sebagai bagian dari variabel `init_connect`.
- Jika pemulihan dari cadangan logis seperti `mysqldump` gagal atau tidak selesai, pastikan bahwa `foreign_key_checks` diatur ke `1` sebelum memulai operasi lain pada sesi yang sama. Cadangan logis mengatur `foreign_key_checks` ke `0` saat dimulai.

Mengonfigurasi seberapa sering buffer log di-flush

Di MySQL Community Edition, untuk membuat transaksi durabel, buffer log InnoDB harus di-flush ke penyimpanan yang durabel. Anda menggunakan parameter `innodb_flush_log_at_trx_commit` untuk mengonfigurasi seberapa sering buffer log di-flush ke disk.

Saat Anda mengatur parameter `innodb_flush_log_at_trx_commit` ke nilai default 1, buffer log akan di-flush pada setiap commit transaksi. Pengaturan ini membantu menjaga basis data memenuhi persyaratan ACID. Kami menyarankan Anda untuk tetap mempertahankan pengaturan default 1.

Mengubah `innodb_flush_log_at_trx_commit` ke nilai nondefault dapat membantu mengurangi latensi bahasa manipulasi data (DHTML), tetapi mengorbankan daya tahan catatan log. Kurangnya durabilitas ini membuat basis data tidak memenuhi persyaratan ACID. Kami menyarankan agar basis data Anda memenuhi persyaratan ACID untuk menghindari risiko data jika server diaktifkan ulang. Untuk informasi selengkapnya tentang parameter ini, lihat innodb_flush_log_at_trx_commit dalam dokumentasi MySQL.

Di Aurora MySQL, pemrosesan log redo dialihkan ke lapisan penyimpanan, jadi tidak ada flushing ke file log yang terjadi pada instans DB. Ketika sebuah penulisan dikeluarkan, log redo akan dikirim dari instans DB penulis langsung ke volume klaster Aurora. Satu-satunya penulisan yang melintasi jaringan adalah catatan log redo. Tidak ada halaman yang akan ditulis dari tingkat basis data.

Secara default, setiap thread yang melakukan transaksi menunggu konfirmasi dari volume cluster Aurora. Konfirmasi ini menunjukkan bahwa catatan ini dan semua catatan log redo sebelumnya telah ditulis dan mencapai kuorum. Mempersistensi catatan log dan mencapai kuorum akan membuat transaksi durabel, baik melalui commit otomatis maupun commit eksplisit. Untuk informasi selengkapnya tentang arsitektur penyimpanan Aurora, lihat Amazon Aurora storage demystified.

Aurora MySQL tidak mem-flush log ke file data seperti yang dilakukan MySQL Community Edition. Namun, Anda dapat menggunakan parameter `innodb_flush_log_at_trx_commit` untuk melonggarkan batasan durabilitas saat menulis catatan log redo ke volume klaster Aurora.

Untuk Aurora MySQL versi 2:

- `innodb_flush_log_at_trx_commit=` 0 atau 2 — Database tidak menunggu konfirmasi bahwa catatan log ulang ditulis ke volume cluster Aurora.
- `innodb_flush_log_at_trx_commit=` 1 — Database menunggu konfirmasi bahwa catatan redo log ditulis ke volume cluster Aurora.

Untuk Aurora MySQL versi 3:

- `innodb_flush_log_at_trx_commit=` 0 — Database tidak menunggu konfirmasi bahwa catatan redo log ditulis ke volume cluster Aurora.
- `innodb_flush_log_at_trx_commit=` 1 atau 2 — Database menunggu konfirmasi bahwa catatan log ulang ditulis ke volume cluster Aurora.

Oleh karena itu, untuk mendapatkan perilaku nondefault yang sama di Aurora MySQL versi 3 yang Anda lakukan dengan nilai yang disetel ke 0 atau 2 di Aurora MySQL versi 2, atur parameter ke 0.

Meskipun dapat menurunkan latensi DML ke klien, pengaturan ini juga dapat mengakibatkan kehilangan data jika terjadi failover atau pengaktifan ulang. Oleh karena itu, sebaiknya pertahankan parameter `innodb_flush_log_at_trx_commit` yang diatur ke nilai default 1.

Meskipun kehilangan data dapat terjadi di MySQL Community Edition dan Aurora MySQL, perilaku di setiap basis data berbeda karena arsitekturnya yang berbeda. Perbedaan arsitektur ini dapat menyebabkan berbagai tingkat kehilangan data. Untuk memastikan bahwa basis data Anda memenuhi persyaratan ACID, selalu atur `innodb_flush_log_at_trx_commit` ke 1.

> **Note**
>
> Di Aurora MySQL versi 3, sebelum Anda dapat mengubah `innodb_flush_log_at_trx_commit` ke nilai selain 1, Anda harus terlebih dahulu mengubah nilai `innodb_trx_commit_allow_data_loss` menjadi 1. Dengan melakukannya, berarti Anda memahami adanya risiko kehilangan data.

Meminimalkan dan memecahkan masalah deadlock Aurora MySQL

Pengguna yang menjalankan beban kerja yang secara rutin mengalami pelanggaran batasan pada indeks sekunder unik atau kunci asing, saat memodifikasi catatan pada halaman data yang sama

secara konkuren, mungkin akan lebih sering mengalami deadlock dan kehabisan waktu tunggu kunci. Deadlock dan kehabisan waktu ini disebabkan oleh perbaikan bug MySQL Community Edition.

Perbaikan ini disertakan dalam MySQL Community Edition versi 5.7.26 dan lebih tinggi, serta di-backport ke Aurora MySQL versi 2.10.3 dan yang lebih tinggi. Perbaikan ini diperlukan untuk memberlakukan kemampuan serialisasi, dengan menerapkan penguncian tambahan untuk jenis operasi bahasa manipulasi data (DML) ini, pada perubahan yang dibuat terhadap catatan dalam tabel InnoDB. Masalah ini terungkap sebagai bagian dari penyelidikan masalah deadlock yang ditimbulkan oleh perbaikan bug MySQL Community Edition sebelumnya.

Perbaikan ini mengubah penanganan internal untuk rollback sebagian pembaruan tuple (baris) di mesin penyimpanan InnoDB. Operasi yang menghasilkan pelanggaran batasan pada kunci asing atau indeks sekunder unik menyebabkan rollback sebagian. Ini termasuk, tetapi tidak terbatas pada, pernyataan `INSERT...ON DUPLICATE KEY UPDATE`, `REPLACE INTO`, dan `INSERT IGNORE` konkuren (upsert).

Dalam konteks ini, rollback sebagian tidak mengacu pada rollback transaksi tingkat aplikasi, melainkan rollback InnoDB internal terhadap perubahan ke indeks berklaster ketika pelanggaran batasan ditemui. Misalnya, nilai kunci duplikat ditemukan selama operasi upsert.

Dalam operasi penyisipan normal, InnoDB secara atomis membuat entri indeks berklaster dan sekunder untuk setiap indeks. Jika InnoDB mendeteksi nilai duplikat pada indeks sekunder unik selama operasi upsert, entri yang disisipkan dalam indeks berklaster harus dikembalikan (rollback sebagian), dan pembaruan kemudian harus diterapkan ke baris duplikat yang ada. Selama langkah rollback sebagian internal ini, InnoDB harus mengunci setiap catatan yang dilihat sebagai bagian dari operasi. Perbaikan ini memastikan kemampuan serialisasi transaksi dengan menerapkan penguncian tambahan setelah rollback sebagian.

Meminimalkan deadlock InnoDB

Anda dapat mengambil pendekatan berikut untuk mengurangi frekuensi deadlock dalam instans basis data Anda. Contoh lainnya dapat ditemukan dalam dokumentasi MySQL.

1. Untuk mengurangi kemungkinan deadlock, lakukan transaksi segera setelah melakukan serangkaian perubahan terkait. Anda dapat melakukannya dengan memecah transaksi besar (beberapa pembaruan baris di antara commit) menjadi lebih kecil. Jika Anda memasukkan baris secara batch, cobalah untuk mengurangi ukuran penyisipan batch, terutama saat menggunakan operasi upsert yang disebutkan sebelumnya.

Untuk mengurangi jumlah kemungkinan rollback sebagian, Anda dapat mencoba beberapa pendekatan berikut:

a. Ganti operasi penyisipan batch dengan memasukkan satu baris pada satu waktu. Hal ini dapat mengurangi jumlah waktu saat kunci ditahan oleh transaksi yang mungkin memiliki konflik.

b. Alih-alih menggunakan `REPLACE INTO`, tulis ulang pernyataan SQL sebagai transaksi multipernyataan seperti berikut ini:

```
BEGIN;
DELETE conflicting rows;
INSERT new rows;
COMMIT;
```

c. Alih-alih menggunakan `INSERT...ON DUPLICATE KEY UPDATE`, tulis ulang pernyataan SQL sebagai transaksi multipernyataan seperti berikut ini:

```
BEGIN;
SELECT rows that conflict on secondary indexes;
UPDATE conflicting rows;
INSERT new rows;
COMMIT;
```

2. Hindari transaksi yang berjalan lama, aktif atau idle, yang mungkin menahan kunci. Hal ini termasuk sesi klien MySQL interaktif yang mungkin terbuka untuk waktu yang lama dengan transaksi yang tidak di-commit. Saat mengoptimalkan ukuran transaksi atau ukuran batch, dampaknya dapat bervariasi tergantung pada sejumlah faktor seperti konkurensi, jumlah duplikat, dan struktur tabel. Setiap perubahan harus diterapkan dan diuji berdasarkan beban kerja Anda.
3. Dalam beberapa situasi, deadlock dapat terjadi ketika dua transaksi mencoba mengakses set data yang sama, baik dalam satu maupun beberapa tabel, dalam urutan yang berbeda-beda. Untuk mencegah hal ini, Anda dapat memodifikasi transaksi untuk mengakses data dalam urutan yang sama, sehingga menserialisasi akses. Misalnya, buat antrean transaksi yang akan diselesaikan. Pendekatan ini dapat membantu menghindari deadlock ketika beberapa transaksi terjadi secara bersamaan.
4. Menambahkan indeks yang dipilih dengan cermat ke tabel Anda dapat meningkatkan selektivitas dan mengurangi kebutuhan untuk mengakses baris, sehingga mengurangi penguncian.
5. Jika Anda mengalami penguncian celah, Anda dapat memodifikasi tingkat isolasi transaksi menjadi `READ COMMITTED` untuk sesi atau transaksi agar mencegahnya. Untuk informasi

selengkapnya tentang tingkat isolasi InnoDB dan perilakunya, lihat Transaction isolation levels dalam dokumentasi MySQL.

> **Note**
>
> Meskipun Anda dapat mengambil tindakan pencegahan untuk mengurangi kemungkinan deadlock terjadi, deadlock adalah perilaku basis data yang sudah diperkirakan dan tetap dapat terjadi. Aplikasi harus memiliki logika yang diperlukan untuk menangani deadlock ketika terjadi. Misalnya, terapkan logika percobaan ulang dan backing-off dalam aplikasi. Yang terbaik adalah mengatasi akar penyebab masalahnya, tetapi jika deadlock memang terjadi, aplikasi memiliki opsi untuk menunggu dan mencoba lagi.

Memantau deadlock InnoDB

Deadlock dapat terjadi di MySQL ketika transaksi aplikasi mencoba mengambil kunci tingkat tabel dan tingkat baris dengan cara yang menghasilkan peristiwa tunggu sirkular. Deadlock InnoDB yang sesekali tidak selalu menjadi masalah karena mesin penyimpanan InnoDB mendeteksi kondisi ini dengan segera dan melakukan rollback terhadap salah satu transaksi secara otomatis. Jika Anda sering mengalami deadlock, kami sarankan untuk meninjau dan memodifikasi aplikasi Anda untuk mengurangi masalah performa dan menghindari deadlock. Ketika deteksi deadlock diaktifkan (default), InnoDB secara otomatis mendeteksi deadlock transaksi dan melakukan rollback transaksi untuk mengatasi deadlock. InnoDB mencoba memilih transaksi kecil untuk di-rollback. Ukuran transaksi akan ditentukan berdasarkan jumlah baris yang disisipkan, diperbarui, atau dihapus.

- Pernyataan `SHOW ENGINE` – Pernyataan `SHOW ENGINE INNODB STATUS \G` berisi detail deadlock terbaru yang ditemui pada basis data sejak pengaktifan ulang terakhir.
- Log kesalahan MySQL – Jika Anda sering mengalami deadlock dengan output pernyataan `SHOW ENGINE` yang tidak memadai, Anda dapat mengaktifkan parameter klaster DB innodb_print_all_deadlocks.

  Ketika parameter ini diaktifkan, informasi tentang semua deadlock dalam transaksi pengguna InnoDB dicatat dalam log kesalahan Aurora MySQL.
- CloudWatch Metrik Amazon — Kami juga menyarankan Anda memantau kebuntuan secara proaktif menggunakan metrik. CloudWatch `Deadlocks` Untuk informasi selengkapnya, lihat Metrik tingkat instans untuk Amazon Aurora.

- CloudWatch Log Amazon — Dengan CloudWatch Log, Anda dapat melihat metrik, menganalisis data log, dan membuat alarm waktu nyata. Untuk informasi selengkapnya, lihat Memantau kesalahan di Amazon Aurora MySQL dan Amazon RDS for MySQL menggunakan Amazon dan mengirim notifikasi menggunakan Amazon SNS. CloudWatch

  Menggunakan CloudWatch Log dengan `innodb_print_all_deadlocks` dihidupkan, Anda dapat mengonfigurasi alarm untuk memberi tahu Anda ketika jumlah kebuntuan melebihi ambang batas yang diberikan. Untuk menentukan ambang batas, kami sarankan Anda mengamati tren Anda dan menggunakan nilai berdasarkan beban kerja normal Anda.
- Wawasan Performa – Saat menggunakan Wawasan Performa, Anda dapat memantau metrik `innodb_deadlocks` dan `innodb_lock_wait_timeout`. Untuk informasi selengkapnya tentang metrik ini, lihat Penghitung non-native untuk Aurora MySQL.

Memecahkan masalah kinerja database MySQL Amazon Aurora

Topik ini berfokus pada beberapa masalah kinerja Aurora MySQL DB yang umum, dan cara memecahkan masalah atau mengumpulkan informasi untuk memperbaiki masalah ini dengan cepat. Kami membagi kinerja database menjadi dua kategori:

- Kinerja server — Seluruh server database berjalan lebih lambat.
- Kinerja kueri — Satu atau beberapa kueri membutuhkan waktu lebih lama untuk dijalankan.

AWS opsi pemantauan

Kami menyarankan Anda menggunakan opsi AWS pemantauan berikut untuk membantu pemecahan masalah:

- Amazon CloudWatch — Amazon CloudWatch memantau AWS sumber daya Anda dan aplikasi yang Anda jalankan AWS secara real time. Anda dapat menggunakan CloudWatch untuk mengumpulkan dan melacak metrik, yang merupakan variabel yang dapat Anda ukur untuk sumber daya dan aplikasi Anda. Untuk informasi selengkapnya, lihat Apa itu Amazon CloudWatch? .

  Anda dapat melihat semua metrik sistem dan memproses informasi untuk instans DB Anda di. AWS Management Console Anda dapat mengonfigurasi cluster DB MySQL Aurora Anda untuk mempublikasikan data log umum, lambat, audit, dan kesalahan ke grup log di Amazon Logs. CloudWatch Ini memungkinkan Anda untuk melihat tren, memelihara log jika host terpengaruh, dan membuat garis dasar untuk kinerja “normal” untuk mengidentifikasi anomali atau perubahan dengan mudah. Untuk informasi selengkapnya, lihat Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch .
- Pemantauan yang Disempurnakan — Untuk mengaktifkan CloudWatch metrik Amazon tambahan untuk database MySQL Aurora, aktifkan Pemantauan yang Ditingkatkan. Saat Anda membuat atau memodifikasi klaster Aurora DB, pilih Aktifkan Pemantauan yang Ditingkatkan. Hal ini memungkinkan Aurora untuk mempublikasikan metrik kinerja ke. CloudWatch Beberapa metrik utama yang tersedia termasuk penggunaan CPU, koneksi database, penggunaan penyimpanan, dan latensi kueri. Ini dapat membantu mengidentifikasi kemacetan kinerja.

  Jumlah informasi yang ditransfer untuk instans DB berbanding lurus dengan granularitas yang ditentukan untuk Enhanced Monitoring. Interval pemantauan yang lebih kecil menghasilkan pelaporan metrik OS yang lebih sering dan meningkatkan biaya pemantauan. Untuk mengelola biaya, tetapkan granularitas yang berbeda untuk contoh yang berbeda di Anda. Akun AWS

Granularitas default pada pembuatan instance adalah 60 detik. Untuk informasi selengkapnya, lihat Biaya Pemantauan yang Disempurnakan.

- Performance Insights — Anda dapat melihat semua metrik panggilan database. Ini termasuk kunci DB, menunggu, dan jumlah baris yang diproses, yang semuanya dapat Anda gunakan untuk pemecahan masalah. Saat Anda membuat atau memodifikasi klaster Aurora DB, pilih Aktifkan Performance Insights. Secara default, Performance Insights memiliki periode retensi data 7 hari, tetapi dapat disesuaikan untuk menganalisis tren kinerja jangka panjang. Untuk retensi lebih dari 7 hari, Anda perlu meningkatkan ke tingkat berbayar. Untuk informasi selengkapnya, lihat harga Performance Insights. Anda dapat mengatur periode retensi data untuk setiap instans Aurora DB secara terpisah. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

Alasan paling umum untuk masalah kinerja database

Anda dapat menggunakan langkah-langkah berikut untuk memecahkan masalah kinerja di database Aurora MySQL Anda. Kami mencantumkan langkah-langkah ini dalam urutan investigasi logis, tetapi tidak dimaksudkan untuk linier. Satu penemuan bisa melompat melintasi langkah, yang memungkinkan serangkaian jalur investigasi.

1. Beban kerja— Memahami beban kerja database Anda.
2. Pencatatan log— Tinjau semua log database.
3. Kinerja kueri— Periksa rencana eksekusi kueri Anda untuk melihat apakah mereka telah berubah. Perubahan kode dapat menyebabkan rencana berubah.

Beban kerja

Beban kerja database dapat dilihat sebagai membaca dan menulis. Dengan pemahaman tentang beban kerja database “normal”, Anda dapat menyetel kueri dan server database untuk memenuhi permintaan saat berubah. Ada sejumlah alasan berbeda mengapa kinerja dapat berubah, jadi langkah pertama adalah memahami apa yang telah berubah.

- Apakah ada peningkatan versi mayor atau minor?

  Peningkatan versi utama mencakup perubahan pada kode mesin, terutama di pengoptimal, yang dapat mengubah rencana eksekusi kueri. Saat memutakhirkan versi basis data, terutama versi utama, sangat penting bagi Anda untuk menganalisis beban kerja database dan menyetelnya.

Tuning dapat melibatkan mengoptimalkan dan menulis ulang kueri, atau menambahkan dan memperbarui pengaturan parameter, tergantung pada hasil pengujian. Memahami apa yang menyebabkan dampak akan memungkinkan Anda untuk mulai fokus pada area spesifik itu.

Untuk informasi selengkapnya, lihat Apa yang baru di MySQL 8.0 dan Server dan variabel status serta opsi ditambahkan, tidak digunakan lagi, atau dihapus di MySQL 8.0 dalam dokumentasi MySQL, dan. Perbandingan Aurora MySQL versi 2 dan Aurora MySQL versi 3

- Apakah ada peningkatan data yang sedang diproses (jumlah baris)?
- Apakah ada lebih banyak kueri yang berjalan secara bersamaan?
- Apakah ada perubahan skema atau database?
- Apakah ada cacat atau perbaikan kode?

Daftar Isi

- Metrik host instance
  - CPU
  - Memori
  - Jaringan
- Metrik basis data

Metrik host instance

Pantau metrik host instans seperti CPU, memori, dan aktivitas jaringan untuk membantu memahami apakah telah terjadi perubahan beban kerja. Ada dua konsep utama untuk memahami perubahan beban kerja:

- Pemanfaatan — Penggunaan perangkat, seperti CPU atau disk. Ini bisa berbasis waktu atau berbasis kapasitas.
  - Berbasis waktu — Jumlah waktu sumber daya sibuk selama periode pengamatan tertentu.
  - Berbasis kapasitas — Jumlah throughput yang dapat diberikan oleh sistem atau komponen, sebagai persentase dari kapasitasnya.
- Saturasi — Sejauh mana lebih banyak pekerjaan dibutuhkan dari sumber daya daripada yang dapat diproses. Ketika penggunaan berbasis kapasitas mencapai 100%, pekerjaan ekstra tidak dapat diproses dan harus antri.

CPU

Anda dapat menggunakan alat berikut untuk mengidentifikasi penggunaan dan saturasi CPU:

- CloudWatch menyediakan `CPUUtilization` metrik. Jika ini mencapai 100%, maka instance jenuh. Namun, CloudWatch metrik dirata-ratakan lebih dari 1 menit, dan tidak memiliki perincian.

  Untuk informasi selengkapnya, lihat Metrik tingkat instans untuk Amazon Aurora.
- Enhanced Monitoring menyediakan metrik yang dikembalikan oleh `top` perintah sistem operasi. Ini menunjukkan rata-rata beban dan status CPU berikut, dengan perincian 1 detik:
  - `Idle (%)`= Waktu idle
  - `IRQ (%)`= Perangkat lunak menyela
  - `Nice (%)`= Waktu yang tepat untuk proses dengan prioritas yang baik.
  - `Steal (%)`= Waktu yang dihabiskan untuk melayani penyewa lain (terkait virtualisasi)
  - `System (%)`= Waktu sistem
  - `User (%)`= Waktu pengguna
  - `Wait (%)`= I/O tunggu

  Untuk informasi selengkapnya, lihat Metrik OS untuk Aurora.

Memori

Jika sistem berada di bawah tekanan memori, dan konsumsi sumber daya mencapai saturasi, Anda harus mengamati pemindaian halaman, paging, pertukaran, dan kesalahan tingkat tinggi. out-of-memory

Anda dapat menggunakan alat berikut untuk mengidentifikasi penggunaan dan saturasi memori:

CloudWatch menyediakan `FreeableMemory` metrik, yang menunjukkan berapa banyak memori yang dapat direklamasi dengan membilas beberapa cache OS dan memori bebas saat ini.

Untuk informasi selengkapnya, lihat Metrik tingkat instans untuk Amazon Aurora.

Enhanced Monitoring menyediakan metrik berikut yang dapat membantu Anda mengidentifikasi masalah penggunaan memori:

- `Buffers (KB)`— Jumlah memori yang digunakan untuk buffering permintaan I/O sebelum menulis ke perangkat penyimpanan, dalam kilobyte.

- `Cached (KB)`— Jumlah memori yang digunakan untuk caching file system berbasis I/O.
- `Free (KB)`— Jumlah memori yang tidak ditetapkan, dalam kilobyte.
- `Swap`— Cached, Gratis, dan Total.

Misalnya, jika Anda melihat bahwa instans DB Anda menggunakan `Swap` memori, maka jumlah total memori untuk beban kerja Anda lebih besar daripada instans Anda saat ini tersedia. Kami merekomendasikan untuk meningkatkan ukuran instans DB Anda atau menyetel beban kerja Anda untuk menggunakan lebih sedikit memori.

Untuk informasi selengkapnya, lihat Metrik OS untuk Aurora.

Jaringan

CloudWatch menyediakan metrik berikut untuk total throughput jaringan, semuanya rata-rata lebih dari 1 menit:

- `NetworkReceiveThroughput`— Jumlah throughput jaringan yang diterima dari klien oleh setiap instance di cluster Aurora DB.
- `NetworkTransmitThroughput`— Jumlah throughput jaringan yang dikirim ke klien oleh setiap instance di cluster Aurora DB.
- `NetworkThroughput`— Jumlah throughput jaringan yang diterima dari dan ditransmisikan ke klien oleh setiap instance di cluster Aurora DB.
- `StorageNetworkReceiveThroughput`— Jumlah throughput jaringan yang diterima dari subsistem penyimpanan Aurora oleh setiap instance di cluster DB.
- `StorageNetworkTransmitThroughput`— Jumlah throughput jaringan yang dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB.
- `StorageNetworkThroughput`— Jumlah throughput jaringan yang diterima dari dan dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB.

Untuk informasi selengkapnya, lihat Metrik tingkat instans untuk Amazon Aurora.

Enhanced Monitoring menyediakan grafik yang `network` diterima (RX) dan ditransmisikan (TX), dengan granularitas hingga 1 detik.

Untuk informasi selengkapnya, lihat Metrik OS untuk Aurora.

Metrik basis data

Periksa CloudWatch metrik berikut untuk perubahan beban kerja:

- `BlockedTransactions`— Rata-rata jumlah transaksi dalam database yang diblokir per detik.
- `BufferCacheHitRatio`— Persentase permintaan yang dilayani oleh cache buffer.
- `CommitThroughput`— Jumlah rata-rata operasi komit per detik.
- `DatabaseConnections`— Jumlah koneksi jaringan klien ke instance database.
- `Deadlocks`— Jumlah rata-rata kebuntuan dalam database per detik.
- `DMLThroughput`— Jumlah rata-rata sisipan, pembaruan, dan penghapusan per detik.
- `QueryCacheHitRatio`— Persentase permintaan yang dilayani oleh cache kueri.
- `RollbackSegmentHistoryListLength`— Log batalkan yang mencatat transaksi yang dilakukan dengan catatan yang ditandai hapus.
- `RowLockTime`— Total waktu yang dihabiskan untuk memperoleh kunci baris untuk tabel InnoDB.
- `SelectThroughput`— Jumlah rata-rata kueri pilih per detik.

Untuk informasi selengkapnya, lihat Metrik tingkat instans untuk Amazon Aurora.

Pertimbangkan pertanyaan-pertanyaan berikut saat memeriksa beban kerja:

1. Apakah ada perubahan terbaru di kelas instans DB, misalnya mengurangi ukuran instance dari 8xlarge menjadi 4xlarge, atau mengubah dari db.r5 ke db.r6?
2. Bisakah Anda membuat klon dan mereproduksi masalah, atau apakah itu hanya terjadi pada satu contoh itu?
3. Apakah ada kelelahan sumber daya server, CPU tinggi atau kelelahan memori? Jika ya, ini bisa berarti bahwa perangkat keras tambahan diperlukan.
4. Apakah satu atau lebih pertanyaan membutuhkan waktu lebih lama?
5. Apakah perubahan disebabkan oleh peningkatan, terutama peningkatan versi utama? Jika ya, bandingkan metrik sebelum dan sesudah peningkatan.
6. Apakah ada perubahan dalam jumlah instans DB pembaca?
7. Sudahkah Anda mengaktifkan pencatatan umum, audit, atau biner? Untuk informasi selengkapnya, lihat Pencatatan log.
8. Apakah Anda mengaktifkan, menonaktifkan, atau mengubah penggunaan replikasi log biner (binlog) Anda?

9. Apakah ada transaksi jangka panjang yang memegang sejumlah besar kunci baris? Periksa panjang daftar riwayat InnoDB (HLL) untuk indikasi transaksi yang berjalan lama.

   Untuk informasi lebih lanjut, lihat Panjang daftar riwayat InnoDB meningkat secara signifikan dan posting blog Mengapa kueri SELECT saya berjalan lambat di cluster DB Amazon Aurora MySQL saya? .

   a. Jika HLL besar disebabkan oleh transaksi tulis, itu berarti UNDO log terakumulasi (tidak dibersihkan secara teratur). Dalam transaksi tulis besar, akumulasi ini dapat tumbuh dengan cepat. Di MySQLUNDO, disimpan di tablespace SYSTEM. SYSTEMRuang meja tidak dapat menyusut. UNDOLog dapat menyebabkan SYSTEM tablespace tumbuh menjadi beberapa GB, atau bahkan TB. Setelah pembersihan, lepaskan ruang yang dialokasikan dengan mengambil cadangan logis (dump) data, lalu impor dump ke instance DB baru.

   b. Jika HLL besar disebabkan oleh transaksi baca (kueri yang berjalan lama), itu bisa berarti bahwa kueri menggunakan sejumlah besar ruang sementara. Lepaskan ruang sementara dengan me-reboot. Periksa metrik DB Performance Insights untuk setiap perubahan di Temp bagian, seperti. created_tmp_tables Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

10. Bisakah Anda membagi transaksi yang berjalan lama menjadi yang lebih kecil yang memodifikasi lebih sedikit baris?

11. Apakah ada perubahan dalam transaksi yang diblokir atau peningkatan kebuntuan? Periksa metrik DB Performance Insights untuk setiap perubahan variabel status di Locks bagian, seperti innodb_row_lock_time, innodb_row_lock_waits dan. innodb_dead_locks Gunakan interval 1 menit atau 5 menit.

12. Apakah ada peningkatan acara tunggu? Periksa Performance Insights acara tunggu dan jenis tunggu menggunakan interval 1 menit atau 5 menit. Analisis peristiwa tunggu teratas dan lihat apakah mereka berkorelasi dengan perubahan beban kerja atau pertentangan database. Misalnya, buf_pool mutex menunjukkan pertentangan kumpulan buffer. Untuk informasi selengkapnya, lihat Menyesuaikan Aurora MySQL dengan peristiwa tunggu.

Pencatatan log

Log MySQL Aurora memberikan informasi penting tentang aktivitas dan kesalahan database. Dengan mengaktifkan log ini, Anda dapat mengidentifikasi dan memecahkan masalah, memahami kinerja database, dan mengaudit aktivitas database. Kami menyarankan Anda mengaktifkan log ini untuk semua instans Aurora MySQL DB Anda untuk memastikan kinerja dan ketersediaan database

yang optimal. Jenis logging berikut dapat diaktifkan. Setiap log berisi informasi spesifik yang dapat menyebabkan mengungkap dampak pada pemrosesan basis data.

- Kesalahan — Aurora MySQL menulis ke log kesalahan hanya pada startup, shutdown, dan ketika menemukan kesalahan. Instan DB dapat memakan waktu berjam-jam atau berhari-hari tanpa perlu menulis entri baru ke log kesalahan. Jika Anda melihat tidak ada entri terbaru, berarti server tidak mengalami kesalahan yang akan mengakibatkan entri log. Pencatatan kesalahan diaktifkan secara default. Untuk informasi selengkapnya, lihat Log kesalahan Aurora MySQL.
- Umum — Log umum memberikan informasi rinci tentang aktivitas database, termasuk semua pernyataan SQL yang dijalankan oleh mesin database. Untuk informasi selengkapnya tentang mengaktifkan logging umum dan pengaturan parameter logging, lihatLog umum dan kueri lambat Aurora MySQL, dan Log kueri umum dalam dokumentasi MySQL.

  > **Note**
  >
  > Log umum dapat tumbuh menjadi sangat besar dan menghabiskan penyimpanan Anda. Untuk informasi selengkapnya, lihat Rotasi dan retensi log untuk Aurora MySQL.

- Kueri lambat - Log kueri lambat terdiri dari pernyataan SQL yang membutuhkan waktu lebih dari long_query_time detik untuk dijalankan dan memerlukan setidaknya baris min_examined_row_limit untuk diperiksa. Anda dapat menggunakan log kueri lambat untuk menemukan kueri yang membutuhkan waktu lama untuk dijalankan dan oleh karena itu kandidat untuk pengoptimalan.

  Nilai default untuk `long_query_time` adalah 10 detik. Kami menyarankan Anda memulai dengan nilai tinggi untuk mengidentifikasi kueri paling lambat, lalu turun untuk fine tuning.

  Anda juga dapat menggunakan parameter terkait, seperti `log_slow_admin_statements` dan`log_queries_not_using_indexes`. Bandingkan `rows_examined` dengan`rows_returned`. Jika `rows_examined` jauh lebih besar dari`rows_returned`, maka kueri tersebut berpotensi memblokir.

  Di Aurora MySQL versi 3, Anda dapat mengaktifkan untuk mendapatkan rincian lebih lanjut. `log_slow_extra` Untuk informasi selengkapnya, lihat Konten log kueri lambat dalam dokumentasi MySQL. Anda juga dapat memodifikasi `long_query_time` pada tingkat sesi untuk men-debug eksekusi kueri secara interaktif, yang sangat berguna jika `log_slow_extra` diaktifkan secara global.

Untuk informasi selengkapnya tentang mengaktifkan pencatatan kueri lambat dan pengaturan parameter logging, lihatLog umum dan kueri lambat Aurora MySQL, dan Log kueri lambat dalam dokumentasi MySQL.

- Audit — Log audit memantau dan mencatat aktivitas database. Pencatatan log audit untuk Aurora MySQL disebut Audit Lanjutan. Untuk mengaktifkan Audit Lanjutan, Anda menetapkan parameter cluster DB tertentu. Untuk informasi selengkapnya, lihat Menggunakan Audit Lanjutan dengan klaster DB Amazon Aurora MySQL.
- Biner — Log biner (binlog) berisi peristiwa yang menggambarkan perubahan database, seperti operasi pembuatan tabel dan perubahan data tabel. Ini juga berisi peristiwa untuk pernyataan yang berpotensi membuat perubahan (misalnya, DELETE yang tidak cocok dengan baris), kecuali logging berbasis baris digunakan. Log biner juga berisi informasi tentang berapa lama setiap pernyataan mengambil data yang diperbarui.

  Menjalankan server dengan logging biner diaktifkan membuat kinerja sedikit lebih lambat. Namun, manfaat log biner yang memungkinkan Anda mengatur replikasi dan untuk operasi pemulihan umumnya lebih besar daripada penurunan kinerja kecil ini.

  > **Note**
  >
  > Aurora MySQL tidak memerlukan pencatatan biner untuk operasi pemulihan.

  Untuk informasi lebih lanjut tentang mengaktifkan logging biner dan pengaturan format binlog, lihatMengkonfigurasi pengelogan biner Aurora MySQL, dan Log biner dalam dokumentasi MySQL.

Anda dapat mempublikasikan log kesalahan, umum, lambat, kueri, dan audit ke Amazon CloudWatch Logs. Untuk informasi selengkapnya, lihat Menerbitkan log basis data ke Log Amazon CloudWatch.

Alat lain yang berguna untuk meringkas file log lambat, umum, dan biner adalah pt-query-digest.

Kinerja kueri

MySQL menyediakan kontrol pengoptimal kueri melalui variabel sistem yang memengaruhi cara rencana kueri dievaluasi, pengoptimalan yang dapat dialihkan, petunjuk pengoptimal dan indeks, dan model biaya pengoptimal. Titik data ini dapat membantu tidak hanya saat membandingkan lingkungan MySQL yang berbeda, tetapi juga untuk membandingkan rencana eksekusi kueri

sebelumnya dengan rencana eksekusi saat ini, dan untuk memahami eksekusi keseluruhan kueri MySQL kapan saja.

Kinerja kueri tergantung pada banyak faktor, termasuk rencana eksekusi, skema tabel dan ukuran, statistik, sumber daya, indeks, dan konfigurasi parameter. Penyetelan kueri memerlukan identifikasi kemacetan dan mengoptimalkan jalur eksekusi.

- Temukan rencana eksekusi untuk kueri dan periksa apakah kueri menggunakan indeks yang sesuai. Anda dapat mengoptimalkan kueri Anda dengan menggunakan `EXPLAIN` dan meninjau detail setiap paket.
- Aurora MySQL versi 3 (kompatibel dengan MySQL 8.0 Community Edition) menggunakan pernyataan. `EXPLAIN ANALYZE` `EXPLAIN ANALYZE`Pernyataan ini adalah alat profil yang menunjukkan di mana MySQL menghabiskan waktu pada kueri Anda dan mengapa. Dengan`EXPLAIN ANALYZE`, Aurora MySQL merencanakan, menyiapkan, dan menjalankan kueri sambil menghitung baris dan mengukur waktu yang dihabiskan di berbagai titik rencana eksekusi. Saat kueri selesai, `EXPLAIN ANALYZE` mencetak rencana dan pengukurannya, bukan hasil kueri.
- Perbarui statistik skema Anda dengan menggunakan `ANALYZE` pernyataan. Pengoptimal kueri terkadang dapat memilih rencana eksekusi yang buruk karena statistik yang sudah ketinggalan zaman. Hal ini dapat menyebabkan kinerja kueri yang buruk karena perkiraan kardinalitas yang tidak akurat dari tabel dan indeks. `last_update`Kolom tabel innodb_table_stats menunjukkan terakhir kali statistik skema Anda diperbarui, yang merupakan indikator “kebuntuan” yang baik.
- Masalah lain dapat terjadi, seperti kemiringan distribusi data, yang tidak diperhitungkan untuk kardinalitas tabel. Untuk informasi selengkapnya, lihat Memperkirakan kompleksitas TABEL ANALISIS untuk tabel InnoDB dan statistik Histogram di MySQL dalam dokumentasi MySQL.

Memahami waktu yang dihabiskan oleh pertanyaan

Berikut ini adalah cara untuk menentukan waktu yang dihabiskan oleh kueri:

- Profiling
- Skema Kinerja
- Pengoptimal kueri

Profiling

Secara default, pembuatan profil dinonaktifkan. Aktifkan pembuatan profil, lalu jalankan kueri lambat dan tinjau profilnya.

```
SET profiling = 1;
Run your query.
SHOW PROFILE;
```

1. Identifikasi tahap di mana waktu paling banyak dihabiskan. Menurut status utas Umum dalam dokumentasi MySQL, membaca dan memproses baris untuk pernyataan seringkali merupakan status `SELECT` yang paling lama berjalan selama masa kueri yang diberikan. Anda dapat menggunakan `EXPLAIN` pernyataan untuk memahami bagaimana MySQL menjalankan kueri ini.
2. Tinjau log kueri lambat `rows_sent` untuk mengevaluasi `rows_examined` dan memastikan bahwa beban kerja serupa di setiap lingkungan. Untuk informasi selengkapnya, lihat Pencatatan log.
3. Jalankan perintah berikut untuk tabel yang merupakan bagian dari kueri yang diidentifikasi:

   ```
   SHOW TABLE STATUS\G;
   ```
4. Tangkap output berikut sebelum dan sesudah menjalankan kueri di setiap lingkungan:

   ```
   SHOW GLOBAL STATUS;
   ```
5. Jalankan perintah berikut di setiap lingkungan untuk melihat apakah ada kueri/sesi lain yang memengaruhi kinerja kueri sampel ini.

   ```
   SHOW FULL PROCESSLIST;

   SHOW ENGINE INNODB STATUS\G;
   ```

   Terkadang, ketika sumber daya di server sibuk, itu berdampak pada setiap operasi lain di server, termasuk kueri. Anda juga dapat menangkap informasi secara berkala saat kueri dijalankan atau menyiapkan `cron` pekerjaan untuk menangkap informasi pada interval yang berguna.

Skema Performa

Skema Kinerja memberikan informasi yang berguna tentang kinerja runtime server, sementara memiliki dampak minimal pada kinerja itu. Ini berbeda dari`information_schema`, yang menyediakan informasi skema tentang instans DB. Untuk informasi selengkapnya, lihat Mengaktifkan Skema Performa untuk Wawasan Performa di Aurora MySQL.

Jejak pengoptimal kueri

Untuk memahami mengapa rencana kueri tertentu dipilih untuk dieksekusi, Anda dapat mengatur `optimizer_trace` untuk mengakses pengoptimal kueri MySQL.

Jalankan jejak pengoptimal untuk menampilkan informasi ekstensif tentang semua jalur yang tersedia untuk pengoptimal dan pilihannya.

```
SET SESSION OPTIMIZER_TRACE="enabled=on";
SET optimizer_trace_offset=-5, optimizer_trace_limit=5;

-- Run your query.
SELECT * FROM table WHERE x = 1 AND y = 'A';

-- After the query completes:
SELECT * FROM information_schema.OPTIMIZER_TRACE;
SET SESSION OPTIMIZER_TRACE="enabled=off";
```

Meninjau pengaturan pengoptimal kueri

Aurora MySQL versi 3 (kompatibel dengan MySQL 8.0 Community Edition) memiliki banyak perubahan terkait pengoptimalan dibandingkan dengan Aurora MySQL versi 2 (kompatibel dengan MySQL 5.7 Community Edition). Jika Anda memiliki beberapa nilai kustom untuk`optimizer_switch`, kami sarankan Anda meninjau perbedaan default dan menetapkan `optimizer_switch` nilai yang paling sesuai untuk beban kerja Anda. Kami juga menyarankan Anda menguji opsi yang tersedia untuk Aurora MySQL versi 3 untuk memeriksa kinerja kueri Anda.

> **Note**
>
> Aurora MySQL versi 3 menggunakan nilai default komunitas 20 untuk parameter innodb_stats_persistent_sample_pages.

Anda dapat menggunakan perintah berikut untuk menunjukkan `optimizer_switch` nilai-nilai:

```
SELECT @@optimizer_switch\G;
```

Tabel berikut menunjukkan `optimizer_switch` nilai default untuk Aurora MySQL versi 2 dan 3.

Pengaturan	Aurora MySQL versi 2	Aurora MySQL versi 3
batched_key_access	off	off
block_nested_loop	on	on
condition_fanout_filter	on	on
derived_condition_pushdown	–	on
derived_merge	on	on
duplicateweedout	on	on
engine_condition_pushdown	on	on
firstmatch	on	on
hash_join	off	on
hash_join_cost_based	on	–
hypergraph_optimizer	–	off
index_condition_pushdown	on	on
index_merge	on	on
index_merge_intersection	on	on
index_merge_sort_union	on	on
index_merge_union	on	on
loosescan	on	on
materialization	on	on
mrr	on	on
mrr_cost_based	on	on

Pengaturan	Aurora MySQL versi 2	Aurora MySQL versi 3
prefer_ordering_index	on	on
semijoin	on	on
skip_scan	–	on
subquery_materialization_cost_based	on	on
subquery_to_derived	–	off
use_index_extensions	on	on
use_invisible_indexes	–	off

Untuk informasi selengkapnya, lihat Pengoptimalan yang dapat dialihkan (MySQL 5.7) dan pengoptimalan Switchable (MySQL 8.0) dalam dokumentasi MySQL.

Referensi Amazon Aurora MySQL

Referensi ini mencakup informasi tentang parameter Aurora MySQL, variabel status, dan ekstensi SQL umum atau perbedaan dari mesin basis data MySQL komunitas.

Topik

- Parameter konfigurasi Aurora MySQL
- Peristiwa tunggu Aurora MySQL
- Status thread Aurora MySQL
- Tingkat isolasi Aurora MySQL
- Petunjuk Aurora MySQL
- Prosedur tersimpan Aurora MySQL
- Tabel information_schema khusus Aurora MySQL

Parameter konfigurasi Aurora MySQL

Anda mengelola klaster DB Amazon Aurora MySQL Anda dengan cara yang sama seperti Anda mengelola instans DB Amazon RDS lain, dengan menggunakan parameter dalam grup parameter DB. Amazon Aurora berbeda dari mesin DB lainnya karena Anda memiliki klaster DB yang berisi beberapa instans DB. Akibatnya, beberapa parameter yang Anda gunakan untuk mengelola klaster DB Aurora MySQL berlaku untuk seluruh klaster. Parameter lain hanya berlaku untuk instans DB tertentu dalam klaster DB.

Untuk mengelola parameter tingkat klaster, gunakan grup parameter klaster DB. Untuk mengelola parameter tingkat instans, gunakan grup parameter DB. Setiap instans DB di klaster DB Aurora MySQL kompatibel dengan mesin basis data MySQL. Namun, Anda menerapkan beberapa parameter mesin basis data MySQL di tingkat klaster, dan Anda mengelola parameter ini menggunakan grup parameter klaster DB. Anda tidak dapat menemukan parameter tingkat klaster dalam grup parameter DB untuk instans dalam klaster Aurora DB. Daftar parameter tingkat klaster akan muncul nanti dalam topik ini.

Anda dapat mengelola parameter tingkat cluster dan tingkat instans menggunakan, API, AWS CLI atau Amazon AWS Management Console RDS. Anda menggunakan perintah terpisah untuk mengelola parameter tingkat klaster dan parameter tingkat instans. Misalnya, Anda dapat menggunakan perintah modify-db-cluster-parameter-group CLI untuk mengelola parameter tingkat cluster dalam grup parameter cluster DB. Anda dapat menggunakan perintah modify-db-parameter-

groupCLI untuk mengelola parameter tingkat instance dalam grup parameter DB untuk instance DB di cluster DB.

Anda dapat melihat parameter tingkat klaster dan tingkat instans di konsol, atau dengan menggunakan CLI atau API RDS. Misalnya, Anda dapat menggunakan describe-db-cluster-parameters AWS CLI perintah untuk melihat parameter tingkat cluster dalam grup parameter cluster DB. Anda dapat menggunakan perintah describe-db-parametersCLI untuk melihat parameter tingkat instance dalam grup parameter DB untuk instance DB di cluster DB.

> **Note**
>
> Setiap grup parameter default berisi nilai default untuk semua parameter di grup parameter. Jika parameter memiliki "mesin default" untuk nilai ini, lihat dokumentasi MySQL atau PostgreSQL khusus versi untuk nilai default sebenarnya.
> Kecuali jika dinyatakan lain, parameter yang tercantum dalam tabel berikut berlaku untuk Aurora MySQL versi 2 dan 3.

Untuk informasi selengkapnya tentang grup parameter DB, lihat Bekerja dengan grup parameter. Untuk aturan dan batasan klaster Aurora Serverless v1, lihat Grup parameter untuk Aurora Serverless v1.

Topik

- Parameter tingkat klaster
- Parameter tingkat instans
- Parameter MySQL yang tidak berlaku untuk Aurora MySQL
- Variabel status global Aurora MySQL
- Variabel status MySQL yang tidak berlaku untuk Aurora MySQL

Parameter tingkat klaster

Tabel berikut menunjukkan semua parameter yang berlaku untuk seluruh klaster DB Aurora MySQL.

Nama parameter	Dapat diubah	Catatan
`aurora_binlog_read_buffer_size`	Ya	Hanya memengaruhi klaster yang menggunakan replikasi biner log (binlog). Untuk informasi tentang replikasi binlog, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner). Dihapus dari Aurora MySQL versi 3.
`aurora_binlog_replication_max_yield_seconds`	Ya	Hanya memengaruhi klaster yang menggunakan replikasi biner log (binlog). Untuk informasi tentang replikasi binlog, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner).
`aurora_binlog_replication_sec_index_parallel_workers`	Ya	Menetapkan jumlah thread paralel yang tersedia untuk menerapkan perubahan indeks sekunder saat mereplikasi transaksi untuk tabel besar dengan lebih dari satu indeks sekunder. Parameter diatur ke `0` (dinonaktifkan) secara default. Parameter ini tersedia di Aurora MySQL versi 306 dan lebih tinggi. Untuk informasi selengkapnya, lihat Mengoptimalkan replikasi log biner.
`aurora_binlog_use_large_read_buffer`	Ya	Hanya memengaruhi klaster yang menggunakan replikasi biner log (binlog). Untuk informasi tentang replikasi binlog, lihat Replikasi antara

Nama parameter	Dapat diubah	Catatan
		Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner). Dihapus dari Aurora MySQL versi 3.
aurora_disable_hash_join	Ya	Atur parameter ini ke 0N untuk menonaktifkan pengoptimalan hash join di Aurora MySQL versi 2.09 atau lebih tinggi. Parameter ini tidak didukung untuk versi 3. Untuk informasi selengkapnya, lihat Bekerja dengan kueri paralel untuk Amazon Aurora MySQL.
aurora_enable_replica_log_compression	Ya	Untuk informasi selengkapnya, lihat Pertimbangan performa untuk replikasi Amazon Aurora MySQL. Tidak berlaku untuk klaster yang merupakan bagian dari basis data global Aurora. Dihapus dari Aurora MySQL versi 3.
aurora_enable_repl_bin_log_filtering	Ya	Untuk informasi selengkapnya, lihat Pertimbangan performa untuk replikasi Amazon Aurora MySQL. Tidak berlaku untuk klaster yang merupakan bagian dari basis data global Aurora. Dihapus dari Aurora MySQL versi 3.
aurora_enable_staggered_replica_restart	Ya	Pengaturan ini tersedia di Aurora MySQL versi 3, tetapi tidak digunakan.

Nama parameter	Dapat diubah	Catatan
aurora_enable_zdr	Ya	Pengaturan ini diaktifkan secara default di Aurora MySQL 2.10 dan yang lebih tinggi. Untuk informasi selengkapnya, lihat Zero-downtime restart (ZDR) untuk Amazon Aurora MySQL.
aurora_enhanced_binlog	Ya	Tetapkan nilai parameter ini ke 1 untuk mengaktifkan binlog yang ditingkatkan di Aurora MySQL versi 3.03.1 dan yang lebih tinggi. Untuk informasi selengkap nya, lihat Menyiapkan binlog yang ditingkatkan.
aurora_jemalloc_background_thread	Ya	Gunakan parameter ini untuk mengaktif kan thread latar belakang untuk melakukan operasi pemeliharaan memori. Nilai yang diizinkan adalah 0 (dinonaktifkan) dan 1 (diaktifkan). Nilai default-nya adalah 0. Parameter ini berlaku untuk Aurora MySQL versi 3.05 dan lebih tinggi.

Nama parameter	Dapat diubah	Catatan
`aurora_jemalloc_dirty_decay_ms`	Ya	Gunakan parameter ini untuk mempertahankan memori yang dibebaskan untuk jangka waktu tertentu (dalam milidetik). Mempertahankan memori memungkinkan penggunaan kembali lebih cepat. Nilai yang diizinkan adalah `0`–`18446744073709551615`. Nilai default (`0`) mengembalikan semua memori ke sistem operasi sebagai memori freeable. Parameter ini berlaku untuk Aurora MySQL versi 3.05 dan lebih tinggi.
`aurora_jemalloc_tcache_enabled`	Ya	Gunakan parameter ini untuk melayani permintaan memori kecil (hingga 32 KiB) dalam cache lokal thread, melewati arena memori. Nilai yang diizinkan adalah `0` (dinonaktifkan) dan 1 (diaktifkan). Nilai default-nya adalah 1. Parameter ini berlaku untuk Aurora MySQL versi 3.05 dan lebih tinggi.
`aurora_load_from_s3_role`	Ya	Untuk informasi selengkapnya, lihat Memuat data ke klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3. Saat ini tidak tersedia di Aurora MySQL versi 3. Gunakan `aws_default_s3_role`.

Nama parameter	Dapat diubah	Catatan
aurora_mask_password_hashes_type	Ya	Pengaturan ini diaktifkan secara default di Aurora MySQL 2.11 dan yang lebih tinggi. Gunakan pengaturan ini untuk me-masking hash kata sandi MySQL Aurora di log kueri dan audit yang lambat. Nilai yang diizinkan adalah 0 dan 1 (default) . Saat diatur ke 1, kata sandi dicatat sebagai <secret>. Saat diatur ke 0, kata sandi dicatat sebagai nilai hash (#).
aurora_select_into_s3_role	Ya	Untuk informasi selengkapnya, lihat Menyimpan data dari klaster DB Amazon Aurora MySQL ke dalam file teks di bucket Amazon S3. Saat ini tidak tersedia di Aurora MySQL versi 3. Gunakan aws_default_s3_role .
authentication_kerberos_caseins_cmp	Ya	Mengontrol perbandingan nama pengguna yang tidak peka huruf besar/kecil untuk plugin authentication_kerberos . Atur ke true untuk perbandingan yang tidak peka huruf besar/kecil. Secara default, perbandingan peka huruf besar/kecil digunakan (false). Untuk informasi selengkapnya, lihat Menggunakan autentikasi Kerberos untuk Aurora MySQL. Parameter ini tersedia di Aurora MySQL versi 3.03 dan lebih tinggi.
auto_increment_increment	Ya	

Nama parameter	Dapat diubah	Catatan
`auto_increment_offset`	Ya	
`aws_default_lambda_role`	Ya	Untuk informasi selengkapnya, lihat Menginvokasi fungsi Lambda dari klaster DB Amazon Aurora MySQL.
`aws_default_s3_role`	Ya	Digunakan saat menginvokasi pernyataan `LOAD DATA FROM S3`, `LOAD XML FROM S3`, atau `SELECT INTO OUTFILE S3` dari klaster DB Anda. Di Aurora MySQL versi 2, peran IAM yang ditentukan dalam parameter ini akan digunakan jika peran IAM tidak ditentukan untuk `aurora_load_from_s3_role` atau `aurora_select_into_s3_role` untuk pernyataan yang sesuai. Di Aurora MySQL versi 3, peran IAM yang ditentukan untuk parameter ini selalu digunakan. Untuk informasi selengkapnya, lihat Mengaitkan peran IAM dengan klaster DB Amazon Aurora MySQL.

Nama parameter	Dapat diubah	Catatan
binlog_backup	Ya	Tetapkan nilai parameter ini ke 0 untuk mengaktifkan binlog yang ditingkatkan di Aurora MySQL versi 3.03.1 dan yang lebih tinggi. Anda dapat menonaktifkan parameter ini hanya ketika Anda menggunakan binlog yang ditingkatkan. Untuk informasi selengkapnya, lihat Menyiapkan binlog yang ditingkatkan.
binlog_checksum	Ya	API AWS CLI dan RDS melaporkan nilai None jika parameter ini tidak disetel. Dalam hal ini, Aurora MySQL menggunakan nilai default mesin, yaitu CRC32. Ini berbeda dari pengaturan eksplisit NONE, yang menonaktifkan checksum.
binlog-do-db	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
binlog_format	Ya	Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner).
binlog_group_commit_sync_delay	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
binlog_group_commit_sync_no_delay_count	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
binlog-ignore-db	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.

Nama parameter	Dapat diubah	Catatan
binlog_replication_globaldb	Ya	Tetapkan nilai parameter ini ke 0 untuk mengaktifkan binlog yang ditingkatkan di Aurora MySQL versi 3.03.1 dan yang lebih tinggi. Anda dapat menonaktifkan parameter ini hanya ketika Anda menggunakan binlog yang ditingkatkan. Untuk informasi selengkapnya, lihat Menyiapkan binlog yang ditingkatkan.
binlog_row_image	Tidak	
binlog_row_metadata	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
binlog_row_value_options	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
binlog_rows_query_log_events	Ya	
binlog_transaction_compression	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
binlog_transaction_compression_level_zstd	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.

Nama parameter	Dapat diubah	Catatan
binlog_transaction_dependency_history_size	Ya	Parameter ini menetapkan batas atas jumlah hash baris yang dipertahankan dalam memori dan digunakan untuk mencari transaksi yang terakhir memodifikasi baris tertentu. Setelah jumlah hash ini tercapai, riwayatnya dibersihkan. Parameter ini berlaku untuk Aurora MySQL versi 2.12 dan lebih tinggi, serta versi 3.
binlog_transaction_dependency_tracking	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
character-set-client-handshake	Ya	
character_set_client	Ya	
character_set_connection	Ya	
character_set_database	Ya	
character_set_filesystem	Ya	
character_set_results	Ya	
character_set_server	Ya	
collation_connection	Ya	
collation_server	Ya	
completion_type	Ya	

Nama parameter	Dapat diubah	Catatan
`default_storage_engine`	Tidak	Klaster Aurora MySQL menggunakan mesin penyimpanan InnoDB untuk semua data Anda.
`enforce_gtid_consistency`	Terkadang	Dapat dimodifikasi di Aurora MySQL versi 2 dan lebih tinggi.
`event_scheduler`	Ya	Menunjukkan status Penjadwal Peristiwa. Dapat dimodifikasi hanya pada tingkat klaster di Aurora MySQL versi 3.
`gtid-mode`	Terkadang	Dapat dimodifikasi di Aurora MySQL versi 2 dan lebih tinggi.
`information_schema_stats_expiry`	Ya	Jumlah detik setelah server basis data MySQL mengambil data dari mesin penyimpanan dan mengganti data dalam cache. Nilai yang diizinkan adalah `0`–`31536000`. Parameter ini berlaku untuk Aurora MySQL versi 3.

Nama parameter	Dapat diubah	Catatan
`init_connect`	Ya	Perintah yang akan dijalankan oleh server untuk setiap klien yang terhubung. Gunakan tanda kutip ganda (“) untuk pengaturan agar menghindari kegagalan koneksi, misalnya: `SET optimizer_switch="hash_join =off"` Di Aurora MySQL versi 3, parameter ini tidak berlaku untuk pengguna yang memiliki hak akses `CONNECTIO N_ADMIN` . Ini termasuk pengguna master Aurora. Untuk informasi selengkapnya, lihat Model hak akses berbasis peran.
`innodb_adaptive_hash_index`	Ya	Anda dapat memodifikasi parameter ini pada tingkat klaster DB di Aurora MySQL versi 2 dan 3. Indeks Hash Adaptif tidak didukung pada instans DB pembaca.

Nama parameter	Dapat diubah	Catatan
innodb_aurora_instant_alter_column_allowed	Ya	Mengontrol apakah algoritma INSTANT dapat digunakan untuk operasi ALTER COLUMN di tingkat global. Nilai yang diizinkan adalah sebagai berikut: • 0— INSTANT Algoritma tidak diizinkan untuk ALTER COLUMN operasi (OFF). Kembali ke algoritma lain. • 1— INSTANT Algoritma diperbolehkan untuk ALTER COLUMN operasi (ON). Ini adalah nilai default. Untuk informasi selengkapnya, lihat Column Operations dalam dokumentasi MySQL. Parameter ini berlaku untuk Aurora MySQL versi 3.05 dan lebih tinggi.
innodb_autoinc_lock_mode	Ya	
innodb_checksums	Tidak	Dihapus dari Aurora MySQL versi 3.
innodb_cmp_per_index_enabled	Ya	
innodb_commit_concurrency	Ya	
innodb_data_home_dir	Tidak	Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung.

Nama parameter	Dapat diubah	Catatan
`innodb_deadlock_detect`	Ya	Opsi ini digunakan untuk menonaktifkan deteksi deadlock di Aurora MySQL versi 2.11 dan lebih tinggi serta versi 3. Pada sistem konkurensi tinggi, deteksi deadlock dapat menyebabkan perlambatan ketika banyak thread menunggu kunci yang sama. Baca dokumentasi MySQL untuk informasi selengkapnya tentang parameter ini.
`innodb_default_row_format`	Ya	Parameter ini mendefinisikan format baris default untuk tabel InnoDB (termasuk tabel sementara InnoDB yang dibuat pengguna). Ini berlaku untuk Aurora MySQL versi 2 dan 3. Nilainya bisa `DYNAMIC`, `COMPACT`, atau `REDUNDANT`.
`innodb_file_per_table`	Ya	Parameter ini memengaruhi cara penyimpanan tabel disusun. Untuk informasi selengkapnya, lihat Penskalaan penyimpanan.

Nama parameter	Dapat diubah	Catatan
innodb_flush_log_at_trx_commit	Ya	Kami sangat menyarankan agar Anda menggunakan nilai default 1. Di Aurora MySQL versi 3, sebelum Anda dapat mengatur parameter ini ke nilai selain 1, Anda harus menetapkan nilai innodb_trx_commit_allow_data_loss ke 1. Untuk informasi selengkapnya, lihat Mengonfigurasi seberapa sering buffer log di-flush.
innodb_ft_max_token_size	Ya	
innodb_ft_min_token_size	Ya	
innodb_ft_num_word_optimize	Ya	
innodb_ft_sort_pll_degree	Ya	
innodb_online_alter_log_max_size	Ya	
innodb_optimize_fulltext_only	Ya	
innodb_page_size	Tidak	
innodb_print_all_deadlocks	Ya	Saat diaktifkan, mencatat informasi tentang semua deadlock InnoDB di log kesalahan Aurora MySQL. Untuk informasi selengkapnya, lihat Meminimalkan dan memecahkan masalah deadlock Aurora MySQL.

Nama parameter	Dapat diubah	Catatan
innodb_purge_batch_size	Ya	
innodb_purge_threads	Ya	
innodb_rollback_on_timeout	Ya	
innodb_rollback_segments	Ya	
innodb_spin_wait_delay	Ya	
innodb_strict_mode	Ya	
innodb_support_xa	Ya	Dihapus dari Aurora MySQL versi 3.
innodb_sync_array_size	Ya	
innodb_sync_spin_loops	Ya	
innodb_stats_include_delete_marked	Ya	Ketika parameter ini diaktifkan, InnoDB menyertakan catatan yang ditandai hapus saat menghitung statistik pengoptimisasi persisten. Parameter ini berlaku untuk Aurora MySQL versi 2.12 dan lebih tinggi, serta versi 3.
innodb_table_locks	Ya	

Nama parameter	Dapat diubah	Catatan
innodb_trx_commit_allow_data_loss	Ya	Di Aurora MySQL versi 3, atur nilai parameter ini ke 1 sehingga Anda dapat mengubah nilai innodb_flush_log_at_trx_commit . Nilai default innodb_trx_commit_allow_data_loss adalah 0. Untuk informasi selengkapnya, lihat Mengonfigurasi seberapa sering buffer log di-flush.
innodb_undo_directory	Tidak	Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung.
internal_tmp_disk_storage_engine	Ya	Mengontrol mesin penyimpanan dalam memori mana yang digunakan untuk tabel sementara internal. Nilai yang diizinkan adalah INNODB dan MYISAM. Parameter ini berlaku untuk Aurora MySQL versi 2.
internal_tmp_mem_storage_engine	Ya	Mengontrol mesin penyimpanan dalam memori mana yang digunakan untuk tabel sementara internal. Nilai yang diizinkan adalah MEMORY dan TempTable . Parameter ini berlaku untuk Aurora MySQL versi 3.
key_buffer_size	Ya	Cache kunci untuk tabel MyISAM. Untuk informasi selengkapnya, lihat mutex keycache->cache_lock.

Nama parameter	Dapat diubah	Catatan
`lc_time_names`	Ya	
`low_priority_updates`	Ya	Operasi `INSERT`, `UPDATE`, `DELETE`, dan `LOCK TABLE WRITE` menunggu sampai tidak ada operasi `SELECT` yang tertunda. Parameter ini hanya memengaruhi mesin penyimpanan yang hanya menggunakan penguncian tingkat tabel (MyISAM, MEMORY, MERGE). Parameter ini berlaku untuk Aurora MySQL versi 3.

Nama parameter	Dapat diubah	Catatan
`lower_case_table_names`	Ya (Aurora MySQL versi 2) Hanya pada waktu pembuatan klaster (Aurora MySQL versi 3)	Di Aurora MySQL versi 2.10 dan versi 2.x yang lebih tinggi, pastikan untuk mem-boot ulang semua instans pembaca setelah mengubah pengaturan ini dan mem-boot ulang instans penulis. Untuk detailnya, lihat Mem-boot ulang klaster Aurora dengan ketersediaan baca. Di Aurora MySQL versi 3, nilai parameter ini diatur secara permanen pada saat klaster dibuat. Jika Anda menggunakan nilai nondefault untuk opsi ini, siapkan grup parameter kustom Aurora MySQL versi 3 Anda sebelum meningkatkan, dan tentukan grup parameter selama operasi pemulihan snapshot yang membuat klaster versi 3. Dengan basis data global Aurora berdasarkan Aurora MySQL, Anda tidak dapat melakukan peningkatan di tempat dari Aurora MySQL versi 2 ke versi 3 jika parameter `lower_case_table_names` diaktifkan. Untuk informasi selengkapnya tentang metode yang dapat Anda gunakan, lihat Peningkatan versi utama.
`master-info-repository`	Ya	Dihapus dari Aurora MySQL versi 3.
`master_verify_checksum`	Ya	Aurora MySQL versi 2. Gunakan `source_verify_checksum` di Aurora MySQL versi 3.

Nama parameter	Dapat diubah	Catatan
`max_delayed_threads`	Ya	Mengatur jumlah maksimum thread untuk menangani pernyataan `INSERT DELAYED`. Parameter ini berlaku untuk Aurora MySQL versi 3.
`max_error_count`	Ya	Jumlah maksimum pesan kesalahan , peringatan, dan catatan yang akan disimpan untuk ditampilkan. Parameter ini berlaku untuk Aurora MySQL versi 3.
`max_execution_time`	Ya	Batas waktu untuk menjalankan pernyataan `SELECT`, dalam milidetik. Nilainya bisa berkisar `0`–`18446744073709551615` . Ketika diatur ke`0`, tidak ada batas waktu. Untuk informasi selengkapnya, lihat max_execution_time dalam dokumentasi MySQL.
`min_examined_row_limit`	Ya	Gunakan parameter ini agar kueri yang memeriksa lebih sedikit jumlah baris dari yang ditentukan tidak dicatat. Parameter ini berlaku untuk Aurora MySQL versi 3.
`partial_revokes`	Tidak	Parameter ini berlaku untuk Aurora MySQL versi 3.

Nama parameter	Dapat diubah	Catatan
`preload_buffer_size`	Ya	Ukuran buffer yang dialokasikan saat melakukan pra-pemuatan indeks. Parameter ini berlaku untuk Aurora MySQL versi 3.
`query_cache_type`	Ya	Dihapus dari Aurora MySQL versi 3.

Nama parameter	Dapat diubah	Catatan
`read_only`	Ya	Ketika parameter ini diaktifkan, server tidak mengizinkan pembaruan kecuali dari yang dilakukan oleh thread replika. Untuk Aurora MySQL versi 2, nilai yang valid adalah sebagai berikut: • `0` – `OFF` • `1` – `ON` • `{TrueIfReplica}` — `ON` untuk membaca replika. Ini adalah nilai default. • `{TrueIfClusterReplica}` — `ON` untuk klaster replika seperti replika baca lintas wilayah, cluster sekunder dalam database global Aurora, dan penerapan biru/hijau. Untuk Aurora MySQL versi 3, nilai yang valid adalah sebagai berikut: • `0`—`OFF`. Ini adalah nilai default. • `1` – `ON` • `{TrueIfClusterReplica}` — `ON` untuk klaster replika seperti replika baca lintas wilayah, cluster sekunder dalam database global Aurora, dan penerapan biru/hijau. Di Aurora MySQL versi 3, parameter ini tidak berlaku untuk pengguna yang memiliki hak akses `CONNECTION_ADMIN` . Ini termasuk pengguna

Nama parameter	Dapat diubah	Catatan
		master Aurora. Untuk informasi selengkapnya, lihat Model hak akses berbasis peran.
relay-log-space-limit	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
replica_parallel_type	Ya	Parameter ini memungkinkan eksekusi paralel pada replika dari semua thread yang tidak di-commit yang sudah ada dalam fase persiapan, tanpa melanggar konsistensi. Parameter ini berlaku untuk Aurora MySQL versi 3. Di Aurora MySQL versi 3.03.* dan lebih rendah, nilai default-nya adalah DATABASE. Di Aurora MySQL versi 3.04 dan lebih tinggi, nilai default-nya adalah LOGICAL_CLOCK.
replica_preserve_commit_order	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
replica_transaction_retries	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.

Nama parameter	Dapat diubah	Catatan
`replica_type_conversions`	Ya	Parameter ini menentukan jenis konversi yang digunakan pada replika. Nilai yang diizinkan adalah: `ALL_LOSSY` , `ALL_NON_LOSSY` , `ALL_SIGNED` , dan `ALL_UNSIGNED` . Untuk informasi selengkapnya, lihat Replication with differing table definitions on source and replica dalam dokumentasi MySQL. Parameter ini berlaku untuk Aurora MySQL versi 3.
`replicate-do-db`	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
`replicate-do-table`	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
`replicate-ignore-db`	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
`replicate-ignore-table`	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
`replicate-wild-do-table`	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
`replicate-wild-ignore-table`	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
`require_secure_transport`	Ya	Parameter ini berlaku untuk Aurora MySQL versi 2 dan 3. Untuk informasi selengkapnya, lihat Menggunakan TLS dengan klaster DB Aurora MySQL.

Nama parameter	Dapat diubah	Catatan
`rpl_read_size`	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
`server_audit_events`	Ya	
`server_audit_excl_users`	Ya	
`server_audit_incl_users`	Ya	
`server_audit_logging`	Ya	Untuk petunjuk tentang mengungga h log ke Amazon CloudWatch Logs, lihatMenerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch .
`server_audit_logs_upload`	Ya	Anda dapat memublikasikan log audit ke CloudWatch Log dengan mengaktif kan Audit Lanjutan dan menyetel parameter ini. 1 Default untuk parameter `server_audit_logs_upload` adalah `0`. Untuk informasi selengkapnya, lihat Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch .
`server_id`	Tidak	
`skip-character-set-client-h andshake`	Ya	
`skip_name_resolve`	Tidak	
`slave-skip-errors`	Ya	Hanya berlaku untuk klaster Aurora MySQL versi 2, dengan kompatibilitas MySQL 5.7.

Nama parameter	Dapat diubah	Catatan
source_verify_checksum	Ya	Aurora MySQL versi 3
sync_frm	Ya	Dihapus dari Aurora MySQL versi 3.
thread_cache_size	Ya	Jumlah thread yang akan di-cache. Parameter ini berlaku untuk Aurora MySQL versi 2 dan 3.
time_zone	Ya	
tls_version	Ya	Untuk informasi selengkapnya, lihat Versi TLS untuk Aurora MySQL.

Parameter tingkat instans

Tabel berikut menunjukkan semua parameter yang berlaku untuk instans DB tertentu di klaster DB Aurora MySQL.

Nama parameter	Dapat diubah	Catatan
activate_all_roles_on_login	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
allow-suspicious-udfs	Tidak	
aurora_disable_hash_join	Ya	Atur parameter ini ke 0N untuk menonaktifkan pengoptimalan hash join di Aurora MySQL versi 2.09 atau lebih tinggi. Parameter ini tidak didukung untuk versi 3. Untuk informasi selengkapnya, lihat Bekerja dengan kueri paralel untuk Amazon Aurora MySQL.

Nama parameter	Dapat diubah	Catatan
aurora_lab_mode	Ya	Untuk informasi selengkapnya, lihat Mode lab Amazon Aurora MySQL. Dihapus dari Aurora MySQL versi 3.
aurora_oom_response	Ya	Parameter ini didukung untuk Aurora MySQL versi 2 dan 3. Untuk informasi selengkapnya, lihat Amazon Aurora Masalah MySQL out-of-memory .
aurora_parallel_query	Ya	Atur ke ON untuk mengaktifkan kueri paralel di Aurora MySQL versi 2.09 atau lebih tinggi. Parameter aurora_pq lama tidak digunakan dalam versi ini. Untuk informasi selengkapnya, lihat Bekerja dengan kueri paralel untuk Amazon Aurora MySQL.
aurora_pq	Ya	Atur ke OFF untuk menonaktifkan kueri paralel untuk instans DB tertentu di versi Aurora MySQL sebelum 2.09. Di versi 2.09 atau lebih tinggi, aktifkan dan nonaktifkan kueri paralel dengan aurora_parallel_query . Untuk informasi selengkapnya, lihat Bekerja dengan kueri paralel untuk Amazon Aurora MySQL.

Nama parameter	Dapat diubah	Catatan
aurora_read_replica_read_committed	Ya	Mengaktifkan tingkat isolasi READ COMMITTED untuk Replika Aurora dan mengubah perilaku isolasi untuk mengurangi lag pembersihan selama kueri yang berjalan lama. Aktifkan pengaturan ini hanya jika Anda memahami perubahan perilaku dan pengaruhnya terhadap hasil kueri. Misalnya, pengaturan ini menggunakan isolasi yang kurang ketat daripada default MySQL. Ketika diaktifkan, kueri yang berjalan lama mungkin mengamati lebih dari satu salinan dari baris yang sama karena Aurora menyusun ulang data tabel saat kueri sedang berjalan. Untuk informasi selengkapnya, lihat Tingkat isolasi Aurora MySQL.
aurora_tmptable_enable_per_table_limit	Ya	Menentukan apakah parameter tmp_table_size mengontrol ukuran maksimum tabel sementara dalam memori yang dibuat oleh mesin penyimpanan TempTable di Aurora MySQL versi 3.04 dan lebih tinggi. Untuk informasi selengkapnya, lihat Membatasi ukuran tabel sementara internal dalam memori.

Nama parameter	Dapat diubah	Catatan
aurora_use_vector_instructions	Ya	Ketika parameter ini diaktifkan, Aurora MySQL menggunakan petunjuk pemrosesan vektor yang dioptimalkan yang disediakan oleh CPU modern untuk meningkatkan performa pada beban kerja intensif I/O. Pengaturan ini diaktifkan secara default di Aurora MySQL versi 3.05 dan lebih tinggi.
autocommit	Ya	
automatic_sp_privileges	Ya	
back_log	Ya	
basedir	Tidak	Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung.
binlog_cache_size	Ya	
binlog_max_flush_queue_time	Ya	
binlog_order_commits	Ya	
binlog_stmt_cache_size	Ya	
binlog_transaction_compression	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
binlog_transaction_compression_level_zstd	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
bulk_insert_buffer_size	Ya	

Nama parameter	Dapat diubah	Catatan
concurrent_insert	Ya	
connect_timeout	Ya	
core-file	Tidak	Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung.
datadir	Tidak	Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung.
default_authentication_plugin	Tidak	Parameter ini berlaku untuk Aurora MySQL versi 3.
default_time_zone	Tidak	
default_tmp_storage_engine	Ya	Mesin penyimpanan default untuk tabel sementara.
default_week_format	Ya	
delay_key_write	Ya	
delayed_insert_limit	Ya	
delayed_insert_timeout	Ya	
delayed_queue_size	Ya	
div_precision_increment	Ya	
end_markers_in_json	Ya	
eq_range_index_dive_limit	Ya	

Nama parameter	Dapat diubah	Catatan
event_scheduler	Terkadang	Menunjukkan status Penjadwal Peristiwa. Dapat dimodifikasi hanya pada tingkat klaster di Aurora MySQL versi 3.
explicit_defaults_for_timestamp	Ya	
flush	Tidak	
flush_time	Ya	
ft_boolean_syntax	Tidak	
ft_max_word_len	Ya	
ft_min_word_len	Ya	
ft_query_expansion_limit	Ya	
ft_stopword_file	Ya	
general_log	Ya	Untuk petunjuk tentang mengunggah log ke CloudWatch Log, lihatMenerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch .
general_log_file	Tidak	Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung.
group_concat_max_len	Ya	
host_cache_size	Ya	

Nama parameter	Dapat diubah	Catatan
`init_connect`	Ya	Perintah yang akan dijalankan oleh server untuk setiap klien yang terhubung. Gunakan tanda kutip ganda (“) untuk pengaturan agar menghindari kegagalan koneksi, misalnya: `SET optimizer_switch="hash_join =off"` Di Aurora MySQL versi 3, parameter ini tidak berlaku untuk pengguna yang memiliki hak akses `CONNECTIO N_ADMIN` , termasuk pengguna master Aurora. Untuk informasi selengkapnya, lihat Model hak akses berbasis peran.
`innodb_adaptive_hash_index`	Ya	Anda dapat memodifikasi parameter ini di tingkat instans DB di Aurora MySQL versi 2. Ini hanya dapat dimodifikasi pada tingkat klaster DB di Aurora MySQL versi 3. Indeks Hash Adaptif tidak didukung pada instans DB pembaca.
`innodb_adaptive_max_sleep_d elay`	Ya	Memodifikasi parameter ini tidak berdampak karena `innodb_th read_concurrency` selalu 0 untuk Aurora.

Nama parameter	Dapat diubah	Catatan
innodb_aurora_max_partitions_for_range	Ya	Dalam beberapa kasus saat statistik tetap tidak tersedia, Anda dapat menggunakan parameter ini untuk meningkatkan performa estimasi jumlah baris pada tabel yang dipartisi. Anda dapat mengaturnya ke nilai dalam rentang 0–8192, yang menentukan jumlah partisi yang akan diperiksa selama estimasi jumlah baris. Nilai default adalah 0, yang menentukan estimasi menggunakan semua partisi, sesuai dengan perilaku MySQL default. Parameter ini tersedia untuk Aurora MySQL versi 3.03.1 dan lebih tinggi.
innodb_autoextend_increment	Ya	
innodb_buffer_pool_dump_at_shutdown	Tidak	
innodb_buffer_pool_dump_now	Tidak	
innodb_buffer_pool_filename	Tidak	
innodb_buffer_pool_load_abort	Tidak	
innodb_buffer_pool_load_at_startup	Tidak	

Nama parameter	Dapat diubah	Catatan
innodb_buffer_pool_load_now	Tidak	
innodb_buffer_pool_size	Ya	Nilai default-nya direpresentasikan dengan rumus. Untuk detail tentang cara nilai DBInstanceClassMemory dalam rumus dihitung, lihat Variabel formula parameter DB.
innodb_change_buffer_max_size	Tidak	Aurora MySQL tidak menggunakan buffer perubahan InnoDB sama sekali.
innodb_compression_failure_threshold_pct	Ya	
innodb_compression_level	Ya	
innodb_compression_pad_pct_max	Ya	
innodb_concurrency_tickets	Ya	Memodifikasi parameter ini tidak berdampak karena innodb_thread_concurrency selalu 0 untuk Aurora.
innodb_deadlock_detect	Ya	Opsi ini digunakan untuk menonaktifkan deteksi deadlock di Aurora MySQL versi 2.11 dan lebih tinggi serta versi 3. Pada sistem konkurensi tinggi, deteksi deadlock dapat menyebabkan perlambatan ketika banyak thread menunggu kunci yang sama. Baca dokumentasi MySQL untuk informasi selengkapnya tentang parameter ini.

Nama parameter	Dapat diubah	Catatan
innodb_file_format	Ya	Dihapus dari Aurora MySQL versi 3.
innodb_flushing_avg_loops	Tidak	
innodb_force_load_corrupted	Tidak	
innodb_ft_aux_table	Ya	
innodb_ft_cache_size	Ya	
innodb_ft_enable_stopword	Ya	
innodb_ft_server_stopword_table	Ya	
innodb_ft_user_stopword_table	Ya	
innodb_large_prefix	Ya	Dihapus dari Aurora MySQL versi 3.
innodb_lock_wait_timeout	Ya	
innodb_log_compressed_pages	Tidak	
innodb_lru_scan_depth	Ya	
innodb_max_purge_lag	Ya	
innodb_max_purge_lag_delay	Ya	
innodb_monitor_disable	Ya	
innodb_monitor_enable	Ya	
innodb_monitor_reset	Ya	

Nama parameter	Dapat diubah	Catatan
innodb_monitor_reset_all	Ya	
innodb_old_blocks_pct	Ya	
innodb_old_blocks_time	Ya	
innodb_open_files	Ya	
innodb_print_all_deadlocks	Ya	Saat diaktifkan, mencatat informasi tentang semua deadlock InnoDB di log kesalahan Aurora MySQL. Untuk informasi selengkapnya, lihat Meminimalkan dan memecahkan masalah deadlock Aurora MySQL.
innodb_random_read_ahead	Ya	
innodb_read_ahead_threshold	Ya	
innodb_read_io_threads	Tidak	
innodb_read_only	Tidak	Aurora MySQL mengelola status hanya baca dan baca/tulis instans DB berdasarkan jenis klaster. Misalnya, klaster terprovisi memiliki satu instans DB baca/tulis (instans primer) dan instans lain dalam klaster berstatus hanya baca (Replika Aurora).
innodb_replication_delay	Ya	
innodb_sort_buffer_size	Ya	
innodb_stats_auto_recalc	Ya	
innodb_stats_method	Ya	

Nama parameter	Dapat diubah	Catatan
innodb_stats_on_metadata	Ya	
innodb_stats_persistent	Ya	
innodb_stats_persistent_sample_pages	Ya	
innodb_stats_transient_sample_pages	Ya	
innodb_thread_concurrency	Tidak	
innodb_thread_sleep_delay	Ya	Memodifikasi parameter ini tidak berdampak karena innodb_thread_concurrency selalu 0 untuk Aurora.
interactive_timeout	Ya	Aurora mengevaluasi nilai minimum interactive_timeout dan wait_timeout . Kemudian, layanan ini menggunakan nilai minimum tersebut sebagai batas waktu untuk mengakhiri semua sesi idle, interaktif, dan non-interaktif.
internal_tmp_disk_storage_engine	Ya	Mengontrol mesin penyimpanan dalam memori mana yang digunakan untuk tabel sementara internal. Nilai yang diizinkan adalah INNODB dan MYISAM. Parameter ini berlaku untuk Aurora MySQL versi 2.

Nama parameter	Dapat diubah	Catatan
`internal_tmp_mem_storage_engine`	Ya	Mengontrol mesin penyimpanan dalam memori mana yang digunakan untuk tabel sementara internal. Nilai yang diizinkan adalah `MEMORY` dan `TempTable` . Parameter ini berlaku untuk Aurora MySQL versi 3.
`join_buffer_size`	Ya	
`keep_files_on_create`	Ya	
`key_buffer_size`	Ya	Cache kunci untuk tabel MyISAM. Untuk informasi selengkapnya, lihat mutex keycache->cache_lock.
`key_cache_age_threshold`	Ya	
`key_cache_block_size`	Ya	
`key_cache_division_limit`	Ya	
`local_infile`	Ya	
`lock_wait_timeout`	Ya	

Nama parameter	Dapat diubah	Catatan
log-bin	Tidak	Mengatur binlog_format ke STATEMENT , MIXED, atau ROW akan secara otomatis mengatur log-bin ke ON. Mengatur binlog_format ke OFF akan secara otomatis mengatur log-bin ke OFF. Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner).
log_bin_trust_function_creators	Ya	
log_bin_use_v1_row_events	Ya	Dihapus dari Aurora MySQL versi 3.
log_error	Tidak	
log_output	Ya	
log_queries_not_using_indexes	Ya	
log_slave_updates	Tidak	Aurora MySQL versi 2. Gunakan log_replica_updates di Aurora MySQL versi 3.
log_replica_updates	Tidak	Aurora MySQL versi 3
log_throttle_queries_not_using_indexes	Ya	
log_warnings	Ya	Dihapus dari Aurora MySQL versi 3.
long_query_time	Ya	

Nama parameter	Dapat diubah	Catatan
low_priority_updates	Ya	Operasi INSERT, UPDATE, DELETE, dan LOCK TABLE WRITE menunggu sampai tidak ada operasi SELECT yang tertunda. Parameter ini hanya memengaruhi mesin penyimpanan yang hanya menggunakan penguncian tingkat tabel (MyISAM, MEMORY, MERGE). Parameter ini berlaku untuk Aurora MySQL versi 3.
max_allowed_packet	Ya	
max_binlog_cache_size	Ya	
max_binlog_size	Tidak	
max_binlog_stmt_cache_size	Ya	
max_connect_errors	Ya	
max_connections	Ya	Nilai default-nya direpresentasikan dengan rumus. Untuk detail tentang cara nilai DBInstanceClassMemory dalam rumus dihitung, lihat Variabel formula parameter DB. Untuk nilai default yang tergantung pada kelas instans, lihat Koneksi maksimum ke instans DB Aurora MySQL.

Nama parameter	Dapat diubah	Catatan
max_delayed_threads	Ya	Mengatur jumlah maksimum thread untuk menangani pernyataan INSERT DELAYED. Parameter ini berlaku untuk Aurora MySQL versi 3.
max_error_count	Ya	Jumlah maksimum pesan kesalahan , peringatan, dan catatan yang akan disimpan untuk ditampilkan. Parameter ini berlaku untuk Aurora MySQL versi 3.
max_execution_time	Ya	Batas waktu untuk menjalankan pernyataan SELECT, dalam milidetik. Nilainya bisa berkisar 0–18446744073709551615 . Ketika diatur ke0, tidak ada batas waktu. Untuk informasi selengkapnya, lihat max_execution_time dalam dokumentasi MySQL.
max_heap_table_size	Ya	
max_insert_delayed_threads	Ya	
max_join_size	Ya	
max_length_for_sort_data	Ya	Dihapus dari Aurora MySQL versi 3.
max_prepared_stmt_count	Ya	
max_seeks_for_key	Ya	
max_sort_length	Ya	

Nama parameter	Dapat diubah	Catatan
max_sp_recursion_depth	Ya	
max_tmp_tables	Ya	Dihapus dari Aurora MySQL versi 3.
max_user_connections	Ya	
max_write_lock_count	Ya	
metadata_locks_cache_size	Ya	Dihapus dari Aurora MySQL versi 3.
min_examined_row_limit	Ya	Gunakan parameter ini agar kueri yang memeriksa lebih sedikit jumlah baris dari yang ditentukan tidak dicatat. Parameter ini berlaku untuk Aurora MySQL versi 3.
myisam_data_pointer_size	Ya	
myisam_max_sort_file_size	Ya	
myisam_mmap_size	Ya	
myisam_sort_buffer_size	Ya	
myisam_stats_method	Ya	
myisam_use_mmap	Ya	
net_buffer_length	Ya	
net_read_timeout	Ya	
net_retry_count	Ya	
net_write_timeout	Ya	
old-style-user-limits	Ya	

Nama parameter	Dapat diubah	Catatan
old_passwords	Ya	Dihapus dari Aurora MySQL versi 3.
optimizer_prune_level	Ya	
optimizer_search_depth	Ya	
optimizer_switch	Ya	Untuk informasi tentang fitur Aurora MySQL yang menggunakan switch ini, lihat Praktik terbaik dengan Amazon Aurora MySQL.
optimizer_trace	Ya	
optimizer_trace_features	Ya	
optimizer_trace_limit	Ya	
optimizer_trace_max_mem_size	Ya	
optimizer_trace_offset	Ya	
performance-schema-consumer-events-waits-current	Ya	
performance-schema-instrument	Ya	
performance_schema	Ya	
performance_schema_accounts_size	Ya	
performance_schema_consumer_global_instrumentation	Ya	

Nama parameter	Dapat diubah	Catatan
performance_schema_consumer_thread_instrumentation	Ya	
performance_schema_consumer_events_stages_current	Ya	
performance_schema_consumer_events_stages_history	Ya	
performance_schema_consumer_events_stages_history_long	Ya	
performance_schema_consumer_events_statements_current	Ya	
performance_schema_consumer_events_statements_history	Ya	
performance_schema_consumer_events_statements_history_long	Ya	
performance_schema_consumer_events_waits_history	Ya	
performance_schema_consumer_events_waits_history_long	Ya	
performance_schema_consumer_statements_digest	Ya	
performance_schema_digests_size	Ya	

Nama parameter	Dapat diubah	Catatan
performance_schema_events_stages_history_long_size	Ya	
performance_schema_events_stages_history_size	Ya	
performance_schema_events_statements_history_long_size	Ya	
performance_schema_events_statements_history_size	Ya	
performance_schema_events_transactions_history_long_size	Ya	
performance_schema_events_transactions_history_size	Ya	
performance_schema_events_waits_history_long_size	Ya	
performance_schema_events_waits_history_size	Ya	
performance_schema_hosts_size	Ya	
performance_schema_max_cond_classes	Ya	
performance_schema_max_cond_instances	Ya	

Nama parameter	Dapat diubah	Catatan
performance_schema_max_digest_length	Ya	
performance_schema_max_file_classes	Ya	
performance_schema_max_file_handles	Ya	
performance_schema_max_file_instances	Ya	
performance_schema_max_index_stat	Ya	
performance_schema_max_memory_classes	Ya	
performance_schema_max_metadata_locks	Ya	
performance_schema_max_mutex_classes	Ya	
performance_schema_max_mutex_instances	Ya	
performance_schema_max_prepared_statements_instances	Ya	
performance_schema_max_program_instances	Ya	
performance_schema_max_rwlock_classes	Ya	

Nama parameter	Dapat diubah	Catatan
performance_schema_max_rwlock_instances	Ya	
performance_schema_max_socket_classes	Ya	
performance_schema_max_socket_instances	Ya	
performance_schema_max_sql_text_length	Ya	
performance_schema_max_stage_classes	Ya	
performance_schema_max_statement_classes	Ya	
performance_schema_max_statement_stack	Ya	
performance_schema_max_table_handles	Ya	
performance_schema_max_table_instances	Ya	
performance_schema_max_table_lock_stat	Ya	
performance_schema_max_thread_classes	Ya	
performance_schema_max_thread_instances	Ya	

Nama parameter	Dapat diubah	Catatan
performance_schema_session_connect_attrs_size	Ya	
performance_schema_setup_actors_size	Ya	
performance_schema_setup_objects_size	Ya	
performance_schema_show_processlist	Ya	Parameter ini menentukan implementasi SHOW PROCESSLIST mana yang akan digunakan: • Implementasi default diiterasi ke seluruh thread aktif dari dalam pengelola thread sambil mempertahankan mutex global. Hal ini dapat menyebabkan performa lambat, terutama pada sistem yang sibuk. • Implementasi SHOW PROCESSLIST alternatif didasarkan pada tabel Skema Performa processlist. Implementasi ini mengueri data thread aktif dari Skema Performa, bukan pengelola thread, dan tidak memerlukan mutex. Parameter ini berlaku untuk Aurora MySQL versi 2.12 dan lebih tinggi, serta versi 3.
performance_schema_users_size	Ya	

Nama parameter	Dapat diubah	Catatan
pid_file	Tidak	
plugin_dir	Tidak	Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung.
port	Tidak	Aurora MySQL mengelola properti koneksi dan memberlakukan pengaturan yang konsisten untuk semua instans DB dalam klaster.
preload_buffer_size	Ya	Ukuran buffer yang dialokasikan saat melakukan pra-pemuatan indeks. Parameter ini berlaku untuk Aurora MySQL versi 3.
profiling_history_size	Ya	
query_alloc_block_size	Ya	
query_cache_limit	Ya	Dihapus dari Aurora MySQL versi 3.
query_cache_min_res_unit	Ya	Dihapus dari Aurora MySQL versi 3.
query_cache_size	Ya	Nilai default-nya direpresentasikan dengan rumus. Untuk detail tentang cara nilai DBInstanceClassMemory dalam rumus dihitung, lihat Variabel formula parameter DB. Dihapus dari Aurora MySQL versi 3.
query_cache_type	Ya	Dihapus dari Aurora MySQL versi 3.
query_cache_wlock_invalidate	Ya	Dihapus dari Aurora MySQL versi 3.

Nama parameter	Dapat diubah	Catatan
query_prealloc_size	Ya	
range_alloc_block_size	Ya	
read_buffer_size	Ya	

Nama parameter	Dapat diubah	Catatan
`read_only`	Ya	Ketika parameter ini diaktifkan, server tidak mengizinkan pembaruan kecuali dari yang dilakukan oleh thread replika. Untuk Aurora MySQL versi 2, nilai yang valid adalah sebagai berikut: • `0` – `OFF` • `1` – `ON` • `{TrueIfReplica}` — `ON` untuk membaca replika. Ini adalah nilai default. • `{TrueIfClusterReplica}` — `ON` untuk contoh di cluster replika seperti replika baca lintas wilayah, cluster sekunder dalam database global Aurora, dan penerapan biru/hijau. Kami menyarankan Anda menggunakan grup parameter klaster DB di Aurora MySQL versi 2 untuk memastikan bahwa parameter `read_only` diterapkan ke instans penulis baru pada saat failover. **Note** Instans pembaca selalu berstatus hanya baca karena Aurora MySQL mengatur `innodb_read_only` ke 1 di semua pembaca. Oleh karena

Nama parameter	Dapat diubah	Catatan
		itu, `read_only` menjadi redundan di instans pembaca. Dihapus pada tingkat instans dari Aurora MySQL versi 3.
`read_rnd_buffer_size`	Ya	
`relay-log`	Tidak	
`relay_log_info_repository`	Ya	Dihapus dari Aurora MySQL versi 3.
`relay_log_recovery`	Tidak	
`replica_checkpoint_group`	Ya	Aurora MySQL versi 3
`replica_checkpoint_period`	Ya	Aurora MySQL versi 3
`replica_parallel_workers`	Ya	Aurora MySQL versi 3
`replica_pending_jobs_size_max`	Ya	Aurora MySQL versi 3
`replica_skip_errors`	Ya	Aurora MySQL versi 3
`replica_sql_verify_checksum`	Ya	Aurora MySQL versi 3
`safe-user-create`	Ya	
`secure_auth`	Ya	Parameter ini selalu diaktifkan di Aurora MySQL versi 2. Percobaan untuk menonaktifkannya akan menghasilkan kesalahan. Dihapus dari Aurora MySQL versi 3.

Nama parameter	Dapat diubah	Catatan
secure_file_priv	Tidak	Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung.
show_create_table_verbosity	Ya	Pengaktifan variabel ini menyebabkan SHOW_CREATE_TABLE menampilkan ROW_FORMAT terlepas dari apakah format ini merupakan format default. Parameter ini berlaku untuk Aurora MySQL versi 2.12 dan lebih tinggi, serta versi 3.
skip-slave-start	Tidak	
skip_external_locking	Tidak	
skip_show_database	Ya	
slave_checkpoint_group	Ya	Aurora MySQL versi 2. Gunakan replica_checkpoint_group di Aurora MySQL versi 3.
slave_checkpoint_period	Ya	Aurora MySQL versi 2. Gunakan replica_checkpoint_period di Aurora MySQL versi 3.
slave_parallel_workers	Ya	Aurora MySQL versi 2. Gunakan replica_parallel_workers di Aurora MySQL versi 3.
slave_pending_jobs_size_max	Ya	Aurora MySQL versi 2. Gunakan replica_pending_jobs_size_max di Aurora MySQL versi 3.

Nama parameter	Dapat diubah	Catatan
slave_sql_verify_checksum	Ya	Aurora MySQL versi 2. Gunakan replica_sql_verify_checksum di Aurora MySQL versi 3.
slow_launch_time	Ya	
slow_query_log	Ya	Untuk petunjuk tentang mengunggah log ke CloudWatch Log, lihatMenerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch .
slow_query_log_file	Tidak	Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung.
socket	Tidak	
sort_buffer_size	Ya	
sql_mode	Ya	
sql_select_limit	Ya	
stored_program_cache	Ya	
sync_binlog	Tidak	
sync_master_info	Ya	
sync_source_info	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3.
sync_relay_log	Ya	Dihapus dari Aurora MySQL versi 3.
sync_relay_log_info	Ya	
sysdate-is-now	Ya	

Nama parameter	Dapat diubah	Catatan
`table_cache_element_entry_ttl`	Tidak	
`table_definition_cache`	Ya	Nilai default-nya direpresentasikan dengan rumus. Untuk detail tentang cara nilai `DBInstanceClassMemory` dalam rumus dihitung, lihat Variabel formula parameter DB.
`table_open_cache`	Ya	Nilai default-nya direpresentasikan dengan rumus. Untuk detail tentang cara nilai `DBInstanceClassMemory` dalam rumus dihitung, lihat Variabel formula parameter DB.
`table_open_cache_instances`	Ya	
`temp-pool`	Ya	Dihapus dari Aurora MySQL versi 3.
`temptable_max_mmap`	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3. Untuk detailnya, lihat Perilaku tabel sementara baru di Aurora MySQL versi 3.
`temptable_max_ram`	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3. Untuk detailnya, lihat Perilaku tabel sementara baru di Aurora MySQL versi 3.
`temptable_use_mmap`	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3. Untuk detailnya, lihat Perilaku tabel sementara baru di Aurora MySQL versi 3.

Nama parameter	Dapat diubah	Catatan
thread_cache_size	Ya	Jumlah thread yang akan di-cache. Parameter ini berlaku untuk Aurora MySQL versi 2 dan 3.
thread_handling	Tidak	
thread_stack	Ya	
timed_mutexes	Ya	
tmp_table_size	Ya	Mendefinisikan ukuran maksimum tabel sementara dalam memori internal yang dibuat oleh mesin penyimpanan MEMORY di Aurora MySQL versi 3. Di Aurora MySQL versi 3.04 dan yang lebih tinggi, mendefinisikan ukuran maksimum tabel sementara dalam memori internal yang dibuat oleh mesin penyimpanan TempTable saat aurora_tmptable_enable_per_table_limit adalah ON. Untuk informasi selengkapnya, lihat Membatasi ukuran tabel sementara internal dalam memori.
tmpdir	Tidak	Aurora MySQL menggunakan instans terkelola dengan sistem file yang tidak dapat Anda akses secara langsung.
transaction_alloc_block_size	Ya	

Nama parameter	Dapat diubah	Catatan
transaction_isolation	Ya	Parameter ini berlaku untuk Aurora MySQL versi 3. Parameter ini menggantikan tx_isolation .
transaction_prealloc_size	Ya	
tx_isolation	Ya	Dihapus dari Aurora MySQL versi 3. Parameter ini digantikan oleh transaction_isolation .
updatable_views_with_limit	Ya	
validate-password	Tidak	
validate_password_dictionary_file	Tidak	
validate_password_length	Tidak	
validate_password_mixed_case_count	Tidak	
validate_password_number_count	Tidak	
validate_password_policy	Tidak	
validate_password_special_char_count	Tidak	

Nama parameter	Dapat diubah	Catatan
`wait_timeout`	Ya	Aurora mengevaluasi nilai minimum `interactive_timeout` dan `wait_timeout` . Kemudian, layanan ini menggunakan nilai minimum tersebut sebagai batas waktu untuk mengakhiri semua sesi idle, interaktif, dan non-inter aktif.

Parameter MySQL yang tidak berlaku untuk Aurora MySQL

Karena perbedaan arsitektur antara Aurora MySQL dan MySQL, sebagian parameter MySQL tidak berlaku untuk Aurora MySQL.

Parameter MySQL berikut tidak berlaku untuk Aurora MySQL: Daftar ini tidak lengkap.

- `activate_all_roles_on_login` – Parameter ini tidak berlaku untuk Aurora MySQL versi 2. Parameter ini tersedia di Aurora MySQL versi 3.
- `big_tables`
- `bind_address`
- `character_sets_dir`
- `innodb_adaptive_flushing`
- `innodb_adaptive_flushing_lwm`
- `innodb_buffer_pool_chunk_size`
- `innodb_buffer_pool_instances`
- `innodb_change_buffering`
- `innodb_checksum_algorithm`
- `innodb_data_file_path`
- `innodb_dedicated_server`
- `innodb_doublewrite`
- `innodb_flush_log_at_timeout` – Parameter ini tidak berlaku untuk Aurora MySQL. Untuk informasi selengkapnya, lihat Mengonfigurasi seberapa sering buffer log di-flush.

- innodb_flush_method
- innodb_flush_neighbors
- innodb_io_capacity
- innodb_io_capacity_max
- innodb_log_buffer_size
- innodb_log_file_size
- innodb_log_files_in_group
- innodb_log_spin_cpu_abs_lwm
- innodb_log_spin_cpu_pct_hwm
- innodb_log_writer_threads
- innodb_max_dirty_pages_pct
- innodb_numa_interleave
- innodb_page_size
- innodb_redo_log_capacity
- innodb_redo_log_encrypt
- innodb_undo_log_encrypt
- innodb_undo_log_truncate
- innodb_undo_logs
- innodb_undo_tablespaces
- innodb_use_native_aio
- innodb_write_io_threads

Variabel status global Aurora MySQL

Anda dapat menemukan nilai saat ini untuk variabel status global Aurora MySQL dengan menggunakan pernyataan seperti berikut:

```
show global status like '%aurora%';
```

Tabel berikut menjelaskan variabel status global yang digunakan Aurora MySQL.

Nama	Penjelasan
AuroraDb_commits	Jumlah total commit sejak pengaktifan ulang terakhir.
AuroraDb_commit_latency	Latensi commit agregat sejak pengaktifan ulang terakhir.
AuroraDb_ddl_stmt_duration	Latensi DDL agregat sejak pengaktifan ulang terakhir.
AuroraDb_select_stmt_duration	Latensi pernyataan SELECT agregat sejak pengaktifan ulang terakhir.
AuroraDb_insert_stmt_duration	Latensi pernyataan INSERT agregat sejak pengaktifan ulang terakhir.
AuroraDb_update_stmt_duration	Latensi pernyataan UPDATE agregat sejak pengaktifan ulang terakhir.
AuroraDb_delete_stmt_duration	Latensi pernyataan DELETE agregat sejak pengaktifan ulang terakhir.
Aurora_binlog_io_cache_allocated	Jumlah byte yang dialokasikan ke binlog I/O cache.
Aurora_binlog_io_cache_read_requests	Jumlah permintaan baca yang dibuat ke cache I/O binlog.
Aurora_binlog_io_cache_reads	Jumlah permintaan baca yang dilayani dari cache binlog I/O.
Aurora_enhanced_binlog	Menunjukkan apakah binlog yang ditingkatkan diaktifkan atau dinonaktifkan untuk instans DB ini. Untuk informasi selengkapnya, lihat Menyiapkan binlog yang ditingkatkan.
Aurora_external_connection_count	Jumlah koneksi basis data ke instans DB, tidak termasuk koneksi layanan RDS yang

Nama	Penjelasan
	digunakan untuk pemeriksaan kondisi basis data.
`Aurora_fast_insert_cache_hits`	Penghitung yang bertambah saat kursor yang di-cache berhasil diambil dan diverifikasi. Untuk informasi selengkapnya tentang cache penyisipan cepat, lihat Peningkatan performa Amazon Aurora MySQL.
`Aurora_fast_insert_cache_misses`	Penghitung yang bertambah saat kursor yang di-cache tidak lagi valid dan Aurora melakukan penjelajahan indeks normal. Untuk informasi selengkapnya tentang cache penyisipan cepat, lihat Peningkatan performa Amazon Aurora MySQL.
`Aurora_fwd_master_dml_stmt_count`	Total jumlah laporan DML yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora MySQL versi 2.
`Aurora_fwd_master_dml_stmt_duration`	Total durasi laporan DML yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora MySQL versi 2.
`Aurora_fwd_master_errors_rpc_timeout`	Berapa kali koneksi yang diteruskan gagal dibuat pada penulis.
`Aurora_fwd_master_errors_session_limit`	Jumlah kueri yang diteruskan yang ditolak karena `session full` pada penulis.
`Aurora_fwd_master_errors_session_timeout`	Berapa kali sesi penerusan berakhir karena batas waktu pada penulis.
`Aurora_fwd_master_open_sessions`	Jumlah sesi yang diteruskan pada instans DB penulis. Variabel ini berlaku untuk Aurora MySQL versi 2.

Nama	Penjelasan
`Aurora_fwd_master_select_stmt_count`	Total jumlah pernyataan `SELECT` yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora MySQL versi 2.
`Aurora_fwd_master_select_stmt_duration`	Total durasi pernyataan `SELECT` yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora MySQL versi 2.
`Aurora_fwd_writer_dml_stmt_count`	Total jumlah laporan DML yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora MySQL versi 3.
`Aurora_fwd_writer_dml_stmt_duration`	Total durasi laporan DML yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora MySQL versi 3.
`Aurora_fwd_writer_errors_rpc_timeout`	Berapa kali koneksi yang diteruskan gagal dibuat pada penulis.
`Aurora_fwd_writer_errors_session_limit`	Jumlah kueri yang diteruskan yang ditolak karena `session full` pada penulis.
`Aurora_fwd_writer_errors_session_timeout`	Berapa kali sesi penerusan berakhir karena batas waktu pada penulis.
`Aurora_fwd_writer_open_sessions`	Jumlah sesi yang diteruskan pada instans DB penulis. Variabel ini berlaku untuk Aurora MySQL versi 3.
`Aurora_fwd_writer_select_stmt_count`	Total jumlah pernyataan `SELECT` yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora MySQL versi 3.
`Aurora_fwd_writer_select_stmt_duration`	Total durasi pernyataan `SELECT` yang diteruskan ke DB instans penulis ini. Variabel ini berlaku untuk Aurora MySQL versi 3.

Nama	Penjelasan
`Aurora_lockmgr_buffer_pool_memory_used`	Jumlah memori kolam buffer dalam byte yang digunakan oleh pengelola kunci MySQL Aurora.
`Aurora_lockmgr_memory_used`	Jumlah memori dalam byte yang digunakan oleh pengelola kunci Aurora MySQL.
`Aurora_ml_actual_request_cnt`	Jumlah permintaan agregat yang dibuat Aurora MySQL ke layanan machine learning Aurora di semua kueri yang dijalankan oleh pengguna instans DB. Untuk informasi selengkapnya, lihat Menggunakan machine learning Amazon Aurora dengan Aurora MySQL.
`Aurora_ml_actual_response_cnt`	Jumlah respons agregat yang diterima Aurora MySQL dari layanan machine learning Aurora di semua kueri yang dijalankan oleh pengguna instans DB. Untuk informasi selengkapnya, lihat Menggunakan machine learning Amazon Aurora dengan Aurora MySQL.
`Aurora_ml_cache_hit_cnt`	Jumlah hit cache internal agregat yang diterima Aurora MySQL dari layanan machine learning Aurora di semua kueri yang dijalanka n oleh pengguna instans DB. Untuk informasi selengkapnya, lihat Menggunakan machine learning Amazon Aurora dengan Aurora MySQL.

Nama	Penjelasan
`Aurora_ml_logical_request_cnt`	Jumlah permintaan logis yang telah dievaluasi instans DB untuk dikirim ke layanan machine learning Aurora sejak reset status terakhir. Tergantung pada apakah batching telah digunakan, nilai ini dapat lebih tinggi dari `Aurora_ml_actual_request_cnt` . Untuk informasi selengkapnya, lihat Menggunakan machine learning Amazon Aurora dengan Aurora MySQL.
`Aurora_ml_logical_response_cnt`	Jumlah respons agregat yang diterima Aurora MySQL dari layanan machine learning Aurora di semua kueri yang dijalankan oleh pengguna instans DB. Untuk informasi selengkapnya, lihat Menggunakan machine learning Amazon Aurora dengan Aurora MySQL.
`Aurora_ml_retry_request_cnt`	Jumlah permintaan yang dicoba ulang yang dikirim instans DB ke layanan machine learning Aurora sejak reset status terakhir. Untuk informasi selengkapnya, lihat Menggunakan machine learning Amazon Aurora dengan Aurora MySQL.
`Aurora_ml_single_request_cnt`	Jumlah agregat fungsi machine learning Aurora yang dievaluasi oleh mode non-batch di semua kueri yang dijalankan oleh pengguna instans DB. Untuk informasi selengkapnya, lihat Menggunakan machine learning Amazon Aurora dengan Aurora MySQL.

Nama	Penjelasan
`Aurora_pq_bytes_returned`	Jumlah byte untuk struktur data tuple yang ditransmisikan ke simpul head selama kueri paralel. Bagi dengan 16.384 untuk membandingkan dengan `Aurora_pq_pages_pushed_down` .
`Aurora_pq_max_concurrent_requests`	Jumlah maksimum sesi kueri paralel yang dapat berjalan secara konkuren di instans Aurora DB ini. Ini adalah nomor tetap yang tergantung pada kelas instans AWS DB.
`Aurora_pq_pages_pushed_down`	Jumlah halaman data (masing-masing dengan ukuran tetap 16 KiB) tempat kueri paralel menghindari transmisi jaringan ke simpul head.
`Aurora_pq_request_attempted`	Jumlah sesi kueri paralel yang diminta. Nilai ini dapat menunjukkan lebih dari satu sesi per kueri, bergantung pada konsep SQL seperti subkueri dan penggabungan.
`Aurora_pq_request_executed`	Jumlah sesi kueri paralel yang berhasil berjalan.
`Aurora_pq_request_failed`	Jumlah sesi kueri paralel yang menampilkan kesalahan kepada klien. Dalam beberapa kasus, permintaan untuk kueri paralel mungkin gagal, misalnya karena masalah dalam lapisan penyimpanan. Dalam kasus ini, bagian kueri yang gagal dicoba kembali menggunakan mekanisme kueri nonparalel. Jika kueri yang dicoba kembali juga gagal, kesalahan ditampilkan ke klien dan penghitung ini ditambah.

Nama	Penjelasan
`Aurora_pq_request_in_progress`	Jumlah sesi kueri paralel yang sedang berlangsung. Angka ini berlaku untuk instans Aurora DB tertentu yang Anda hubungkan, bukan seluruh klaster Aurora DB. Untuk melihat apakah instans DB mendekati batas konkurens inya, bandingkan nilai ini dengan `Aurora_pq_max_concurrent_requests` .
`Aurora_pq_request_not_chosen`	Berapa kali kueri paralel tidak dipilih untuk memenuhi suatu kueri. Nilai ini adalah jumlah dari beberapa penghitung granular lainnya. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-ben ar dilakukan.
`Aurora_pq_request_not_chose n_below_min_rows`	Berapa kali kueri paralel tidak dipilih karena jumlah baris dalam tabel. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan.
`Aurora_pq_request_not_chose n_column_bit`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena jenis data yang tidak didukung dalam daftar kolom yang diproyeks ikan.
`Aurora_pq_request_not_chose n_column_geometry`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki kolom dengan jenis data `GEOMETRY`. Untuk informasi tentang versi Aurora MySQL yang menghapus batasan ini, lihat Meningkatkan klaster kueri paralel ke Aurora MySQL versi 3.

Nama	Penjelasan
`Aurora_pq_request_not_chosen_column_lob`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki kolom dengan jenis data `LOB`, atau kolom `VARCHAR` yang disimpan secara eksternal karena panjang yang dinyatakan. Untuk informasi tentang versi Aurora MySQL yang menghapus batasan ini, lihat Meningkatkan klaster kueri paralel ke Aurora MySQL versi 3.
`Aurora_pq_request_not_chosen_column_virtual`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel berisi kolom virtual.
`Aurora_pq_request_not_chosen_custom_charset`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki kolom dengan set karakter kustom.
`Aurora_pq_request_not_chosen_fast_ddl`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel saat ini sedang diubah oleh pernyataan `ALTER` DDL cepat.
`Aurora_pq_request_not_chosen_ few_pages_outside_buffer_pool`	Berapa kali kueri paralel tidak dipilih, meskipun kurang dari 95 persen data tabel berada di dalam pool buffer, karena data tabel yang tidak di-buffer tidak cukup untuk membuat kueri paralel layak dijalankan.
`Aurora_pq_request_not_chosen_full_text_index`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel memiliki indeks teks penuh.

Nama	Penjelasan
`Aurora_pq_request_not_chosen_high_buffer_pool_pct`	Berapa kali kueri paralel tidak dipilih karena persentase tinggi data tabel (saat ini, lebih dari 95 persen) sudah berada di dalam pool buffer. Dalam kasus ini, pengoptimal menentukan bahwa membaca data dari pool buffer akan lebih efisien. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan.
`Aurora_pq_request_not_chosen_index_hint`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri mencakup petunjuk indeks.
`Aurora_pq_request_not_chosen_innodb_table_format`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena tabel menggunakan format baris InnoDB yang tidak didukung. Kueri paralel Aurora hanya berlaku untuk format baris `COMPACT`, `REDUNDANT` , dan `DYNAMIC`.
`Aurora_pq_request_not_chosen_long_trx`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri dimulai di dalam transaksi yang berjalan lama. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan.
`Aurora_pq_request_not_chosen_no_where_clause`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri tidak mencakup klausa `WHERE` apa pun.

Nama	Penjelasan
`Aurora_pq_request_not_chosen_range_scan`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri menggunakan pemindaian rentang pada indeks.
`Aurora_pq_request_not_chosen_row_length_too_long`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena total panjang gabungan semua kolom terlalu panjang.
`Aurora_pq_request_not_chosen_small_table`	Berapa kali kueri paralel tidak dipilih karena ukuran keseluruhan tabel, sebagaimana ditentukan oleh jumlah baris dan rata-rata panjang baris. Pernyataan `EXPLAIN` dapat menambah penghitung ini meskipun kueri tidak benar-benar dilakukan.
`Aurora_pq_request_not_chosen_temporary_table`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri mengacu pada tabel sementara yang menggunakan jenis tabel `MyISAM` atau `memory` yang tidak didukung.
`Aurora_pq_request_not_chosen_tx_isolation`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri menggunakan tingkat isolasi transaksi yang tidak didukung. Pada instans pembaca DB, kueri paralel hanya berlaku untuk tingkat isolasi `REPEATABLE READ` dan `READ COMMITTED` .
`Aurora_pq_request_not_chosen_update_delete_stmts`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena kueri adalah bagian dari pernyataan `UPDATE` atau `DELETE`.

Nama	Penjelasan
`Aurora_pq_request_not_chosen_unsupported_access`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena klausa `WHERE` tidak memenuhi kriteria untuk kueri paralel. Hasil ini dapat muncul jika kueri tidak memerlukan pemindaian sarat data, atau jika kueri adalah pernyataan `DELETE` atau `UPDATE`.
`Aurora_pq_request_not_chosen_unsupported_storage_type`	Jumlah permintaan kueri paralel yang menggunakan jalur pemrosesan kueri nonparalel karena klaster DB Aurora MySQL tidak menggunakan konfigurasi penyimpanan klaster Aurora yang didukung. Untuk informasi selengkapnya, lihat Pembatasan. Parameter ini berlaku untuk Aurora MySQL versi 3.04 dan lebih tinggi.
`Aurora_pq_request_throttled`	Berapa kali kueri paralel tidak dipilih karena jumlah maksimum kueri paralel konkuren sudah berjalan pada instans Aurora DB tertentu.
`Aurora_repl_bytes_received`	Jumlah byte yang direplikasi ke instans basis data pembaca Aurora MySQL sejak pengaktifan ulang terakhir. Untuk informasi selengkapnya, lihat Replikasi dengan Amazon Aurora MySQL.

Nama	Penjelasan
`Aurora_reserved_mem_exceeded_incidents`	Berapa kali sejak pengaktifan ulang terakhir bahwa mesin telah melampaui batas memori yang digunakan sistem. Jika `aurora_oom_response` dikonfigurasi, ambang batas ini menentukan kapan aktivitas penghindaran out-of-memory (OOM) dipicu. Untuk informasi selengkapnya tentang respons OOM Aurora MySQL, lihat Amazon Aurora Masalah MySQL out-of-memory .
`Aurora_thread_pool_thread_count`	Jumlah thread saat ini di pool thread Aurora. Untuk informasi selengkapnya tentang pool thread di Aurora MySQL, lihat Pool thread.
`Aurora_tmz_version`	Menunjukkan versi informasi zona waktu saat ini yang digunakan oleh klaster DB. Nilai ini mengikuti format Internet Assigned Numbers Authority (IANA): YYYY`suffix` , misalnya `2022a` dan `2023c`. Parameter ini berlaku untuk Aurora MySQL versi 2.12 dan lebih tinggi, serta versi 3.04 dan lebih tinggi.
`server_aurora_das_running`	Menunjukkan apakah Stream Aktivitas Basis Data (DAS) diaktifkan atau dinonaktifkan pada instans DB ini. Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.

Variabel status MySQL yang tidak berlaku untuk Aurora MySQL

Karena perbedaan arsitektur antara Aurora MySQL dan MySQL, sebagian variabel status MySQL tidak berlaku untuk Aurora MySQL.

Variabel status MySQL berikut tidak berlaku untuk Aurora MySQL: Daftar ini tidak lengkap.

- innodb_buffer_pool_bytes_dirty
- innodb_buffer_pool_pages_dirty
- innodb_buffer_pool_pages_flushed

Aurora MySQL versi 3 menghapus variabel status berikut yang ada di Aurora MySQL versi 2:

- AuroraDb_lockmgr_bitmaps0_in_use
- AuroraDb_lockmgr_bitmaps1_in_use
- AuroraDb_lockmgr_bitmaps_mem_used
- AuroraDb_thread_deadlocks
- available_alter_table_log_entries
- Aurora_lockmgr_memory_used
- Aurora_missing_history_on_replica_incidents
- Aurora_new_lock_manager_lock_release_cnt
- Aurora_new_lock_manager_lock_release_total_duration_micro
- Aurora_new_lock_manager_lock_timeout_cnt
- Aurora_total_op_memory
- Aurora_total_op_temp_space
- Aurora_used_alter_table_log_entries
- Aurora_using_new_lock_manager
- Aurora_volume_bytes_allocated
- Aurora_volume_bytes_left_extent
- Aurora_volume_bytes_left_total
- Com_alter_db_upgrade
- Compression
- External_threads_connected
- Innodb_available_undo_logs
- Last_query_cost
- Last_query_partial_plans
- Slave_heartbeat_period

- `Slave_last_heartbeat`
- `Slave_received_heartbeats`
- `Slave_retried_transactions`
- `Slave_running`
- `Time_since_zero_connections`

Variabel status MySQL ini tersedia di Aurora MySQL versi 2, tetapi tidak tersedia di Aurora MySQL versi 3:

- `Innodb_redo_log_enabled`
- `Innodb_undo_tablespaces_total`
- `Innodb_undo_tablespaces_implicit`
- `Innodb_undo_tablespaces_explicit`
- `Innodb_undo_tablespaces_active`

Peristiwa tunggu Aurora MySQL

Berikut ini adalah beberapa peristiwa tunggu umum di Aurora MySQL.

> **Note**
>
> Untuk informasi tentang menyesuaikan performa Aurora MySQL menggunakan peristiwa tunggu, lihat Menyesuaikan Aurora MySQL dengan peristiwa tunggu.
> Untuk informasi tentang konvensi penamaan yang digunakan dalam peristiwa tunggu MySQL, lihat Performance Schema instrument naming conventions dalam dokumentasi MySQL.

cpu

Jumlah koneksi aktif yang siap dijalankan secara konsisten lebih tinggi daripada jumlah vCPU. Untuk informasi selengkapnya, lihat cpu.

io/aurora_redo_log_flush

Sesi mempersistensi data ke penyimpanan Aurora. Biasanya, peristiwa tunggu ini ditujukan untuk operasi I/O tulis di Aurora MySQL. Untuk informasi selengkapnya, lihat io/aurora_redo_log_flush.

io/aurora_respond_to_client

Pemrosesan kueri telah selesai dan hasilnya ditampilkan ke klien aplikasi untuk versi MySQL Aurora berikut: 2.10.2 dan versi 2.10 yang lebih tinggi, versi 2.09.3 dan versi 2.09 yang lebih tinggi, serta versi 2.07.7 dan versi 2.07 yang lebih tinggi. Bandingkan bandwidth jaringan kelas instans DB dengan ukuran set hasil yang ditampilkan. Selain itu, periksa waktu respons sisi klien. Jika klien tidak responsif dan tidak dapat memproses paket TCP, penghapusan paket dan transmisi ulang TCP dapat terjadi. Situasi ini berdampak negatif pada bandwidth jaringan. Dalam versi yang lebih rendah dari 2.10.2, 2.09.3, dan 2.07.7, peristiwa tunggu secara keliru menyertakan waktu idle. Untuk mempelajari cara menyesuaikan basis data Anda saat peristiwa waktu tunggu ini sering muncul, lihat io/aurora_respond_to_client.

io/file/csv/data

Thread menulis ke tabel dalam format nilai yang dipisahkan koma (CSV). Periksa penggunaan tabel CSV Anda. Penyebab umum peristiwa ini adalah mengatur `log_output` pada tabel.

io/file/sql/binlog

Thread sedang menunggu file log biner (binlog) yang sedang ditulis ke disk.

io/redo_log_flush

Sesi mempersistensi data ke penyimpanan Aurora. Biasanya, peristiwa tunggu ini ditujukan untuk operasi I/O tulis di Aurora MySQL. Untuk informasi selengkapnya, lihat io/redo_log_flush.

io/socket/sql/client_connection

Program `mysqld` sibuk membuat thread untuk menangani koneksi klien baru yang masuk. Untuk informasi selengkapnya, lihat io/socket/sql/client_connection.

io/table/sql/handler

Mesin sedang menunggu akses ke tabel. Peristiwa ini terjadi terlepas dari apakah data di-cache di pool buffer atau diakses pada disk. Untuk informasi selengkapnya, lihat io/table/sql/handler.

lock/table/sql/handler

Peristiwa tunggu ini adalah handler peristiwa tunggu kunci tabel. Untuk informasi selengkapnya tentang peristiwa atom dan molekul di Skema Performa, lihat Performance Schema atom and molecule events dalam dokumentasi MySQL.

synch/cond/innodb/row_lock_wait

Pernyataan bahasa manipulasi data (DML) mengakses baris basis data yang sama pada saat yang bersamaan. Untuk informasi selengkapnya, lihat synch/cond/innodb/row_lock_wait.

synch/cond/innodb/row_lock_wait_cond

Beberapa pernyataan DML mengakses baris basis data yang sama secara bersamaan. Untuk informasi selengkapnya, lihat synch/cond/innodb/row_lock_wait_cond.

synch/cond/sql/MDL_context::COND_wait_status

Thread sedang menunggu kunci metadata tabel. Mesin menggunakan jenis kunci ini untuk mengelola akses konkuren ke skema basis data dan untuk memastikan konsistensi data. Untuk informasi selengkapnya, lihat Optimizing locking operations dalam dokumentasi MySQL. Untuk mempelajari cara menyesuaikan basis data Anda saat peristiwa ini sering muncul, lihat synch/cond/sql/MDL_context::COND_wait_status.

synch/cond/sql/MYSQL_BIN_LOG::COND_done

Anda telah mengaktifkan logging biner. Mungkin ada throughput commit yang tinggi, transaksi dalam jumlah besar yang di-commit, atau replika yang membaca binlog. Pertimbangkan untuk menggunakan pernyataan multibaris atau menggabungkan pernyataan ke dalam satu transaksi. Di Aurora, gunakan basis data global alih-alih replikasi log biner, atau gunakan parameter `aurora_binlog_*`.

synch/mutex/innodb/aurora_lock_thread_slot_futex

Beberapa pernyataan DML mengakses baris basis data yang sama secara bersamaan. Untuk informasi selengkapnya, lihat synch/mutex/innodb/aurora_lock_thread_slot_futex.

synch/mutex/innodb/buf_pool_mutex

Pool buffer tidak cukup besar untuk menampung set data kerja. Atau, beban kerja mengakses halaman dari tabel tertentu, yang mengakibatkan pertentangan di pool buffer. Untuk informasi selengkapnya, lihat synch/mutex/innodb/buf_pool_mutex.

synch/mutex/innodb/fil_system_mutex

Proses sedang menunggu akses ke cache memori ruang tabel. Untuk informasi selengkapnya, lihat synch/mutex/innodb/fil_system_mutex.

synch/mutex/innodb/trx_sys_mutex

Operasi memeriksa, memperbarui, menghapus, atau menambahkan ID transaksi di InnoDB secara konsisten atau terkontrol. Operasi ini memerlukan panggilan mutex `trx_sys`, yang dilacak oleh instrumentasi Skema Performa. Operasi mencakup manajemen sistem transaksi ketika basis data dimulai atau dinonaktifkan, rollback, pembersihan undo, akses baca baris, dan beban pool

buffer. Beban basis data yang tinggi dengan sejumlah besar transaksi mengakibatkan seringnya kemunculan peristiwa tunggu ini. Untuk informasi selengkapnya, lihat synch/mutex/innodb/trx_sys_mutex.

synch/mutex/mysys/KEY_CACHE::cache_lock

Mutex `keycache->cache_lock` mengontrol akses ke cache kunci untuk tabel MyISAM. Meskipun Aurora MySQL tidak mengizinkan penggunaan tabel MyISAM untuk menyimpan data persisten, tabel tersebut digunakan untuk menyimpan tabel sementara internal. Pertimbangkan untuk memeriksa penghitung status `created_tmp_disk_tables` atau `created_tmp_tables` karena dalam situasi tertentu, tabel sementara ditulis ke disk ketika tidak lagi muat dalam memori.

synch/mutex/sql/FILE_AS_TABLE::LOCK_offsets

Mesin memperoleh mutex ini saat membuka atau membuat file metadata tabel. Ketika peristiwa tunggu ini terjadi dengan frekuensi yang berlebihan, jumlah tabel yang dibuat atau dibuka telah melonjak.

synch/mutex/sql/FILE_AS_TABLE::LOCK_shim_lists

Mesin memperoleh mutex ini saat melakukan operasi seperti `reset_size`, `detach_contents`, atau `add_contents` pada struktur internal yang melacak tabel yang dibuka. Mutex menyinkronkan akses ke konten daftar. Ketika peristiwa tunggu ini terjadi dengan frekuensi tinggi, ini menunjukkan perubahan mendadak dalam kumpulan tabel yang sebelumnya diakses. Mesin perlu mengakses tabel baru atau melepaskan konteks yang terkait dengan tabel yang diakses sebelumnya.

synch/mutex/sql/LOCK_open

Jumlah tabel yang dibuka sesi Anda melebihi ukuran cache definisi tabel atau cache pembukaan tabel. Tingkatkan ukuran cache ini. Untuk informasi selengkapnya, lihat Cara MySQL membuka dan menutup tabel.

synch/mutex/sql/LOCK_table_cache

Jumlah tabel yang dibuka sesi Anda melebihi ukuran cache definisi tabel atau cache pembukaan tabel. Tingkatkan ukuran cache ini. Untuk informasi selengkapnya, lihat Cara MySQL membuka dan menutup tabel.

synch/mutex/sql/LOG

Dalam peristiwa tunggu ini, terdapat thread yang menunggu kunci log. Misalnya, thread mungkin akan menunggu kunci untuk menulis ke file log kueri lambat.

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_commit

Dalam peristiwa tunggu ini, ada thread yang menunggu untuk mendapatkan kunci dengan tujuan melakukan commit ke log biner. Pertentangan log biner dapat terjadi pada basis data dengan tingkat perubahan yang sangat tinggi. Tergantung pada versi MySQL, terdapat beberapa kunci yang digunakan untuk melindungi konsistensi dan durabilitas log biner. Di RDS for MySQL, log biner digunakan untuk replikasi dan proses pencadangan otomatis. Di Aurora MySQL, log biner tidak diperlukan untuk replikasi atau pencadangan native. Opsi ini dinonaktifkan secara default, tetapi dapat diaktifkan dan digunakan untuk replikasi eksternal atau penangkapan data perubahan. Untuk informasi selengkapnya, lihat The binary log dalam dokumentasi MySQL.

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_dump_thread_metrics_collection

Jika logging biner diaktifkan, mesin memperoleh mutex ini saat mencetak metrik thread dump aktif ke log kesalahan mesin dan ke peta operasi internal.

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_inactive_binlogs_map

Jika logging biner diaktifkan, mesin memperoleh mutex ini ketika menambahkan, menghapus dari, atau mencari melalui daftar file binlog di belakang yang terbaru.

sync/mutex/sql/MYSQL_BIN_LOG::LOCK_io_cache

Jika logging biner diaktifkan, mesin memperoleh mutex ini selama operasi cache Aurora IO binlog: mengalokasikan, mengubah ukuran, membebaskan, menulis, membaca, membersihkan, dan mengakses info cache. Jika peristiwa ini sering terjadi, mesin mengakses cache tempat peristiwa binlog disimpan. Untuk mengurangi waktu tunggu, kurangi commit. Cobalah mengelompokkan beberapa pernyataan ke dalam satu transaksi.

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_log

Anda telah mengaktifkan logging biner. Mungkin ada throughput commit yang tinggi, banyak transaksi yang dilakukan, atau replika yang membaca binlog. Pertimbangkan untuk menggunakan pernyataan multibaris atau menggabungkan pernyataan ke dalam satu transaksi. Di Aurora, gunakan basis data global alih-alih replikasi log biner atau gunakan parameter `aurora_binlog_*`.

synch/mutex/sql/SERVER_THREAD::LOCK_sync

Mutex `SERVER_THREAD::LOCK_sync` diperoleh selama penjadwalan, pemrosesan, atau peluncuran thread untuk penulisan file. Terjadinya peristiwa tunggu yang berlebihan ini menunjukkan peningkatan aktivitas penulisan dalam basis data.

synch/mutex/sql/TABLESPACES:lock

Mesin memperoleh mutex `TABLESPACES:lock` selama operasi ruang tabel berikut: membuat, menghapus, memotong, dan memperpanjang. Terjadinya peristiwa tunggu yang berlebihan ini menunjukkan frekuensi operasi ruang tabel yang tinggi. Contohnya adalah memuat sejumlah besar data ke dalam basis data.

synch/rwlock/innodb/dict

Dalam peristiwa tunggu ini, ada thread yang menunggu rwlock yang dipertahankan di kamus data InnoDB.

synch/rwlock/innodb/dict_operation_lock

Dalam peristiwa tunggu ini, ada thread yang menahan kunci di operasi kamus data InnoDB.

synch/rwlock/innodb/dict sys RW lock

Sejumlah besar pernyataan bahasa kontrol data (DCL) konkuren dalam kode bahasa definisi data (DDL) dipicu pada saat yang bersamaan. Kurangi dependensi aplikasi pada DDL selama aktivitas aplikasi reguler.

synch/rwlock/innodb/index_tree_rw_lock

Sejumlah besar pernyataan bahasa manipulasi data (DML) yang serupa mengakses objek basis data yang sama secara bersamaan. Coba gunakan pernyataan multibaris. Selain itu, sebarkan beban kerja ke objek basis data yang berbeda-beda. Misalnya, dengan menerapkan partisi.

synch/sxlock/innodb/dict_operation_lock

Sejumlah besar pernyataan bahasa kontrol data (DCL) konkuren dalam kode bahasa definisi data (DDL) dipicu pada saat yang bersamaan. Kurangi dependensi aplikasi pada DDL selama aktivitas aplikasi reguler.

synch/sxlock/innodb/dict_sys_lock

Sejumlah besar pernyataan bahasa kontrol data (DCL) konkuren dalam kode bahasa definisi data (DDL) dipicu pada saat yang bersamaan. Kurangi dependensi aplikasi pada DDL selama aktivitas aplikasi reguler.

synch/sxlock/innodb/hash_table_locks

Sesi tidak dapat menemukan halaman dalam pool buffer. Mesin perlu membaca file atau memodifikasi daftar least-recently used (LRU) untuk pool buffer. Pertimbangkan untuk meningkatkan ukuran cache buffer dan meningkatkan jalur akses untuk kueri yang relevan.

synch/sxlock/innodb/index_tree_rw_lock

Banyak pernyataan bahasa manipulasi data (DML) yang serupa mengakses objek basis data yang sama secara bersamaan. Coba gunakan pernyataan multibaris. Selain itu, sebarkan beban kerja ke objek basis data yang berbeda-beda. Misalnya, dengan menerapkan partisi.

Untuk informasi selengkapnya tentang pemecahan masalah peristiwa tunggu sinkronisasi, lihat Mengapa instans DB MySQL saya menampilkan sejumlah besar sesi aktif yang menunggu peristiwa tunggu SYNCH di Wawasan Performa?.

Status thread Aurora MySQL

Berikut ini adalah beberapa status thread umum untuk Aurora MySQL.

memeriksa izin

Thread memeriksa apakah server memiliki hak akses yang diperlukan untuk menjalankan pernyataan.

memeriksa cache kueri untuk kueri

Server memeriksa apakah kueri saat ini ada di cache kueri.

dibersihkan

Ini adalah status akhir dari koneksi yang pekerjaannya selesai, tetapi belum ditutup oleh klien. Solusi terbaik adalah secara eksplisit menutup koneksi dalam kode. Atau, Anda dapat menetapkan nilai yang lebih rendah untuk `wait_timeout` di grup parameter Anda.

menutup tabel

Thread melakukan flushing data tabel yang diubah ke disk dan menutup tabel yang digunakan. Jika operasi ini tidak cepat, periksa metrik konsumsi bandwidth jaringan berdasarkan bandwidth jaringan kelas instans. Selain itu, periksa apakah nilai untuk parameter `table_open_cache` dan `table_definition_cache` memungkinkan tabel yang cukup dibuka secara bersamaan sehingga mesin tidak perlu sering membuka dan menutup tabel. Parameter ini memengaruhi konsumsi memori pada instans.

mengonversi HEAP ke MyISAM

Kueri mengonversi tabel sementara dari dalam memori ke di disk. Konversi ini diperlukan karena tabel sementara yang dibuat oleh MySQL dalam langkah-langkah perantara pemrosesan kueri bertambah terlalu banyak untuk memori. Periksa nilai `tmp_table_size` dan

`max_heap_table_size`. Di versi yang lebih baru, nama status thread ini adalah `converting HEAP to ondisk`.

mengonversi HEAP ke di disk

Thread mengonversi tabel sementara internal dari tabel dalam memori ke tabel di disk.

salin ke tabel sementara

Thread sedang memproses pernyataan `ALTER TABLE`. Status ini terjadi setelah tabel dengan struktur baru telah dibuat, tetapi sebelum baris disalin ke dalamnya. Untuk thread dalam status ini, Anda dapat menggunakan Skema Performa untuk mendapatkan informasi tentang progres operasi penyalinan.

membuat indeks pengurutan

Aurora MySQL melakukan pengurutan karena tidak dapat menggunakan indeks yang ada untuk memenuhi klausa `ORDER BY` atau `GROUP BY` pada kueri. Untuk informasi selengkapnya, lihat creating sort index.

membuat tabel

Thread membuat tabel permanen atau sementara.

commit tertunda ok selesai

Commit asinkron di Aurora MySQL telah menerima konfirmasi dan selesai.

commit tertunda ok diinisiasi

Thread Aurora MySQL telah memulai proses commit asinkron, tetapi sedang menunggu konfirmasi. Status ini biasanya menunjukkan waktu commit yang sebenarnya dari suatu transaksi.

pengiriman tertunda ok selesai

Thread pekerja Aurora MySQL yang terikat ke koneksi dapat dibebaskan saat respons dikirim ke klien. Utas dapat memulai pekerjaan lain. Status `delayed send ok` berarti bahwa konfirmasi asinkron ke klien selesai.

pengiriman tertunda ok diinisiasi

Thread pekerja Aurora MySQL telah mengirim respons secara asinkron ke klien dan sekarang bebas melakukan pekerjaan untuk koneksi lain. Transaksi telah memulai proses commit asinkron yang belum dikonfirmasi.

mengeksekusi

Thread telah mulai menjalankan pernyataan.

membebaskan item

Thread telah menjalankan perintah. Beberapa pembebasan item yang dilakukan selama status ini memerlukan cache kueri. Status ini biasanya diikuti dengan pembersihan.

inisialisasi

Status ini terjadi sebelum inisialisasi pernyataan `ALTER TABLE`, `DELETE`, `INSERT`, `SELECT`, atau `UPDATE`. Tindakan dalam status ini termasuk flushing log biner atau log InnoDB, dan beberapa pembersihan cache kueri.

master telah mengirim semua binlog ke slave

Simpul primer telah menyelesaikan bagiannya dalam replikasi. Thread sedang menunggu lebih banyak kueri untuk dijalankan sehingga dapat menulis ke log biner (binlog).

membuka tabel

Thread mencoba membuka tabel. Operasi ini cepat kecuali jika pernyataan `ALTER TABLE` atau `LOCK TABLE` perlu diselesaikan, atau melebihi nilai `table_open_cache`.

mengoptimalkan

Server melakukan optimisasi awal untuk kueri.

mempersiapkan

Status ini terjadi selama optimisasi kueri.

akhir kueri

Status ini terjadi setelah memproses kueri, tetapi sebelum status membebaskan item.

menghapus duplikat

Aurora MySQL tidak dapat mengoptimalkan operasi `DISTINCT` pada tahap awal kueri. Aurora MySQL harus menghapus semua baris duplikat sebelum mengirim hasilnya ke klien.

mencari baris untuk pembaruan

Thread mencari semua baris yang cocok sebelum memperbaruinya. Tahap ini diperlukan jika `UPDATE` mengubah indeks yang digunakan mesin untuk menemukan baris.

mengirim peristiwa binlog ke slave

Thread membaca peristiwa dari log biner dan mengirimkannya ke replika.

mengirim hasil yang di-cache ke klien

Server mengambil hasil kueri dari cache kueri dan mengirimkannya ke klien.

mengirim data

Thread sedang membaca dan memproses baris untuk pernyataan `SELECT`, tetapi belum mulai mengirim data ke klien. Proses ini mengidentifikasi halaman mana yang berisi hasil yang diperlukan untuk memenuhi kueri. Untuk informasi selengkapnya, lihat sending data.

mengirim ke klien

Server menulis paket ke klien. Dalam versi MySQL sebelumnya, peristiwa tunggu ini diberi label `writing to net`.

memulai

Ini adalah tahap pertama di awal eksekusi pernyataan.

statistik

Server menghitung statistik untuk mengembangkan rencana eksekusi kueri. Jika thread dalam status ini dalam waktu yang lama, server mungkin terikat pada disk sambil melakukan pekerjaan lain.

menyimpan hasil dalam cache kueri

Server menyimpan hasil kueri dalam cache kueri.

kunci sistem

Thread telah memanggil `mysql_lock_tables`, tetapi status thread belum diperbarui sejak panggilan ini. Status umum ini terjadi karena berbagai alasan.

perbarui

Thread sedang bersiap untuk mulai memperbarui tabel.

memperbarui

Thread sedang mencari baris dan memperbaruinya.

kunci pengguna

Thread mengeluarkan panggilan `GET_LOCK`. Thread meminta kunci advisory dan sedang menunggunya, atau berencana untuk memintanya.

menunggu pembaruan lainnya

Simpul primer telah menyelesaikan bagiannya dalam replikasi. Thread sedang menunggu lebih banyak kueri untuk dijalankan sehingga dapat menulis ke log biner (binlog).

menunggu kunci metadata skema

Ini adalah peristiwa tunggu untuk kunci metadata.

menunggu kunci metadata fungsi tersimpan

Ini adalah peristiwa tunggu untuk kunci metadata.

menunggu kunci metadata prosedur tersimpan

Ini adalah peristiwa tunggu untuk kunci metadata.

menunggu flushing tabel

Thread sedang menjalankan `FLUSH TABLES` dan sedang menunggu semua thread untuk menutup tabelnya. Atau, thread menerima notifikasi bahwa struktur dasar untuk tabel berubah, sehingga harus membuka kembali tabel untuk mendapatkan struktur baru. Untuk membuka kembali tabel, thread harus menunggu sampai semua thread lainnya menutup tabel. Notifikasi ini terjadi jika thread lain telah menggunakan salah satu pernyataan berikut di tabel: `FLUSH TABLES`, `ALTER TABLE`, `RENAME TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, atau `OPTIMIZE TABLE`.

menunggu kunci tingkat tabel

Satu sesi mempertahankan kunci di tabel sementara sesi lain mencoba mendapatkan kunci yang sama di tabel yang sama.

menunggu kunci metadata tabel

Aurora MySQL menggunakan penguncian metadata untuk mengelola akses konkuren ke objek basis data dan untuk memastikan konsistensi data. Dalam peristiwa tunggu ini, satu sesi mempertahankan kunci metadata di tabel sementara sesi lain mencoba mendapatkan kunci yang sama di tabel yang sama. Saat Skema Performa diaktifkan, status thread ini dilaporkan sebagai peristiwa tunggu `synch/cond/sql/MDL_context::COND_wait_status`.

menulis ke net

Server sedang menulis paket ke jaringan. Dalam versi MySQL yang lebih baru, peristiwa tunggu ini diberi label `Sending to client`.

Tingkat isolasi Aurora MySQL

Pelajari bagaimana instans DB di klaster Aurora MySQL menerapkan properti basis data isolasi. Topik ini menjelaskan bagaimana perilaku default Aurora MySQL menyeimbangkan antara konsistensi yang ketat dan performa yang tinggi. Anda dapat menggunakan informasi ini untuk

membantu memutuskan kapan akan mengubah pengaturan default berdasarkan karakteristik beban kerja Anda.

Tingkat isolasi yang tersedia untuk instans penulis

Anda dapat menggunakan tingkat isolasi `REPEATABLE READ`, `READ COMMITTED`, `READ UNCOMMITTED`, dan `SERIALIZABLE` pada instans primer klaster DB Aurora MySQL. Tingkat isolasi ini berfungsi di Aurora MySQL sama seperti di RDS for MySQL.

Tingkat isolasi REPEATABLE READ untuk instans pembaca

Secara default, instans DB Aurora MySQL yang dikonfigurasi sebagai Replika Aurora hanya baca selalu menggunakan tingkat isolasi `REPEATABLE READ`. Instans DB ini mengabaikan pernyataan `SET TRANSACTION ISOLATION LEVEL` dan terus menggunakan tingkat isolasi `REPEATABLE READ`.

Anda tidak dapat mengatur tingkat isolasi untuk instans DB pembaca menggunakan parameter DB atau parameter klaster DB.

Tingkat isolasi READ COMMITTED untuk instans pembaca

Jika aplikasi Anda mencakup beban kerja sarat penulisan pada instans primer dan kueri yang berjalan lama pada Replika Aurora, Anda mungkin mengalami lag pembuangan yang substansial. Lag pembuangan terjadi jika pengumpulan sampah internal diblokir oleh kueri yang berjalan lama. Gejala yang Anda lihat adalah nilai yang tinggi untuk `history list length` dalam output dari perintah `SHOW ENGINE INNODB STATUS`. Anda dapat memantau nilai ini menggunakan metrik `RollbackSegmentHistoryListLength` di CloudWatch. Lag pembuangan yang substansial dapat mengurangi efektivitas indeks sekunder, menurunkan performa kueri secara keseluruhan, dan menyebabkan pemborosan ruang penyimpanan.

Jika Anda mengalami masalah seperti itu, Anda dapat mengatur pengaturan konfigurasi tingkat sesi Aurora MySQL, `aurora_read_replica_read_committed`, agar menggunakan tingkat isolasi `READ COMMITTED` pada Replika Aurora. Saat menerapkan pengaturan ini, Anda dapat membantu mengurangi pelambatan dan ruang terbuang yang dapat timbul karena melakukan kueri berjalan lama pada saat yang sama seperti transaksi yang mengubah tabel Anda.

Kami menyarankan Anda untuk memahami perilaku tertentu Aurora MySQL karena isolasi `READ COMMITTED` sebelum menggunakan pengaturan ini. Perilaku Replika Aurora `READ COMMITTED` sesuai dengan standar ANSI SQL. Namun, isolasinya tidak seketat perilaku

`READ COMMITTED` MySQL biasa yang mungkin Anda kenal. Dengan demikian, Anda mungkin melihat hasil kueri yang berbeda berdasarkan `READ COMMITTED` di replika baca Aurora MySQL daripada kueri yang sama berdasarkan `READ COMMITTED` di instans primer Aurora MySQL atau di RDS for MySQL. Anda dapat mempertimbangkan untuk menggunakan pengaturan `aurora_read_replica_read_committed` untuk kasus seperti laporan komprehensif yang memindai basis data yang sangat besar. Sebaliknya, Anda dapat menghindarinya untuk kueri pendek dengan set hasil kecil yang mementingkan presisi dan keterulangan.

Tingkat isolasi `READ COMMITTED` tidak tersedia untuk sesi dengan klaster sekunder dalam basis data global Aurora yang menggunakan fitur penerusan tulis. Untuk informasi tentang penerusan tulis, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.

Menggunakan READ COMMITTED untuk pembaca

Untuk menggunakan tingkat isolasi `READ COMMITTED` untuk Replika Aurora, atur pengaturan konfigurasi `aurora_read_replica_read_committed` ke `ON`. Gunakan pengaturan ini di tingkat sesi saat terhubung ke Replika Aurora tertentu. Untuk melakukannya, jalankan perintah SQL berikut.

```
set session aurora_read_replica_read_committed = ON;
set session transaction isolation level read committed;
```

Anda dapat menggunakan pengaturan konfigurasi ini sementara waktu untuk melakukan kueri interaktif satu kali. Anda mungkin juga ingin menjalankan aplikasi pelaporan atau analisis data yang mendapatkan manfaat dari tingkat isolasi `READ COMMITTED`, sementara membiarkan pengaturan default tidak berubah untuk aplikasi lain.

Saat pengaturan `aurora_read_replica_read_committed` diaktifkan, gunakan perintah `SET TRANSACTION ISOLATION LEVEL` untuk menentukan tingkat isolasi untuk transaksi yang sesuai.

```
set transaction isolation level read committed;
```

Perbedaan dalam perilaku READ COMITTED pada replika Aurora

Pengaturan `aurora_read_replica_read_committed` membuat tingkat isolasi `READ COMMITTED` tersedia untuk Replika Aurora, dengan perilaku konsistensi yang dioptimalkan untuk transaksi yang berjalan lama. Tingkat isolasi `READ COMMITTED` pada Replika Aurora memiliki isolasi yang tidak terlalu ketat dibandingkan pada instans primer Aurora. Oleh karena itu, aktifkan pengaturan ini hanya di Replika Aurora, yang memungkinkan Anda mengetahui bahwa kueri Anda dapat menerima kemungkinan jenis hasil tertentu yang tidak konsisten.

Kueri Anda dapat mengalami beberapa jenis anomali baca ketika pengaturan `aurora_read_replica_read_committed` diaktifkan. Dua jenis anomali ini sangat penting untuk dipahami dan ditangani dalam kode aplikasi Anda. Pembacaan yang tidak dapat diulang terjadi saat transaksi lain di-commit saat kueri Anda sedang berjalan. Kueri yang berjalan lama dapat melihat data yang berbeda pada awal kueri dibandingkan dengan yang dilihat pada akhir kueri. Pembacaan phantom terjadi ketika transaksi lain menyebabkan baris yang ada disusun ulang saat kueri Anda berjalan, dan satu atau beberapa baris dibaca dua kali oleh kueri Anda.

Kueri Anda mungkin mengalami jumlah baris yang tidak konsisten sebagai hasil dari pembacaan phantom. Kueri Anda juga dapat memberikan hasil yang tidak lengkap atau tidak konsisten karena pembacaan yang tidak dapat diulang. Misalnya, anggaplah sebuah operasi join merujuk ke tabel yang secara bersamaan dimodifikasi oleh pernyataan SQL seperti `INSERT` atau `DELETE`. Dalam kasus ini, kueri join mungkin membaca baris dari satu tabel, tetapi bukan baris yang sesuai dari tabel lain.

Standar ANSI SQL memungkinkan kedua perilaku ini untuk tingkat isolasi `READ COMMITTED`. Namun, perilaku tersebut berbeda dari implementasi `READ COMMITTED` MySQL biasa. Jadi, sebelum mengaktifkan pengaturan `aurora_read_replica_read_committed`, periksa kode SQL yang ada untuk memverifikasi apakah kode ini beroperasi seperti yang diharapkan berdasarkan model konsistensi yang lebih longgar.

Jumlah baris dan hasil lainnya mungkin tidak terlalu konsisten berdasarkan tingkat isolasi `READ COMMITTED` sementara pengaturan ini diaktifkan. Dengan demikian, Anda biasanya mengaktifkan pengaturan hanya saat menjalankan kueri analitik yang mengumpulkan data dalam jumlah besar dan tidak memerlukan presisi absolut. Jika Anda tidak memiliki jenis kueri yang berjalan lama ini bersama dengan beban kerja rata penulisan, Anda mungkin tidak memerlukan pengaturan `aurora_read_replica_read_committed`. Tanpa kombinasi kueri yang berjalan lama dan beban kerja yang sarat penulisan, Anda cenderung tidak akan menghadapi masalah dengan panjang daftar riwayat.

Example Kueri yang menunjukkan perilaku isolasi untuk READ COMMITTED pada Replika Aurora

Contoh berikut menunjukkan bagaimana kueri `READ COMMITTED` pada Replika Aurora dapat memberikan hasil yang tidak dapat diulang jika transaksi mengubah tabel terkait pada saat yang sama. Tabel `BIG_TABLE` berisi 1 juta baris sebelum kueri dimulai. Pernyataan bahasa manipulasi data (DHTML) lainnya menambahkan, menghapus, atau mengubah baris saat sedang berjalan.

Kueri pada instans primer Aurora berdasarkan tingkat isolasi `READ COMMITTED` memberikan hasil yang dapat diprediksi. Namun demikian, overhead dalam mempertahankan tampilan baca yang

konsisten selama masa pakai setiap kueri yang berjalan lama dapat menyebabkan pengumpulan sampah yang menimbulkan biaya mahal pada lain waktu.

Kueri pada Replika Aurora berdasarkan tingkat isolasi `READ COMMITTED` akan dioptimalkan untuk meminimalkan overhead pengumpulan sampah. Kompromi nya adalah bahwa hasilnya mungkin berbeda-beda bergantung pada apakah kueri memuat baris yang ditambahkan, dihapus, atau disusun ulang oleh transaksi yang di-commit saat kueri berjalan. Kueri diizinkan untuk mempertimbangkan baris-baris ini, tetapi tidak diwajibkan. Untuk tujuan demonstrasi, kueri hanya memeriksa jumlah baris dalam tabel dengan menggunakan fungsi `COUNT(*)`.

Waktu	Pernyataan DML di instans primer Aurora	Kueri di instans primer Aurora dengan READ COMMITTED	Kueri di replika Aurora dengan READ COMMITTED
T1	`INSERT INTO big_table SELECT * FROM other_table LIMIT 1000000; COMMIT;`		
T2		Q1: `SELECT COUNT(*) FROM big_table;`	Q2: `SELECT COUNT(*) FROM big_table;`
T3	`INSERT INTO big_table (c1, c2) VALUES (1, 'one more row'); COMMIT;`		
T4		Jika Q1 selesai sekarang, hasilnya adalah 1.000.000.	Jika Q2 selesai sekarang, hasilnya adalah 1.000.000 atau 1.000.001.
T5	`DELETE FROM big_table LIMIT 2; COMMIT;`		

Waktu	Pernyataan DML di instans primer Aurora	Kueri di instans primer Aurora dengan READ COMMITTED	Kueri di replika Aurora dengan READ COMMITTED
T6		Jika Q1 selesai sekarang, hasilnya adalah 1.000.000.	Jika Q2 selesai sekarang, hasilnya adalah 1.000.000 atau 1.000.001 atau 999.999 atau 999.998.
T7	`UPDATE big_table SET c2 = CONCAT(c2 ,c2,c2); COMMIT;`		
T8		Jika Q1 selesai sekarang, hasilnya adalah 1.000.000.	Jika Q2 selesai sekarang, hasilnya adalah 1.000.000 atau 1.000.001 atau 999.999, atau mungkin angka tertentu yang lebih tinggi.
T9		Q3: `SELECT COUNT(*) FROM big_table;`	Q4: `SELECT COUNT(*) FROM big_table;`
T10		Jika Q3 selesai sekarang, hasilnya adalah 999.999.	Jika Q4 selesai sekarang, hasilnya adalah 999.999.

Waktu	Pernyataan DML di instans primer Aurora	Kueri di instans primer Aurora dengan READ COMMITTED	Kueri di replika Aurora dengan READ COMMITTED
T11		Q5: `SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;`	Q6: `SELECT COUNT(*) FROM parent_table p JOIN child_table c ON (p.id = c.id) WHERE p.id = 1000;`
T12	`INSERT INTO parent_table (id, s) VALUES (1000, 'hello'); INSERT INTO child_table (id, s) VALUES (1000, 'world'); COMMIT;`		
T13		Jika Q5 selesai sekarang, hasilnya adalah 0.	Jika Q6 selesai sekarang, hasilnya adalah 0 atau 1.

Jika kueri selesai dengan cepat, sebelum transaksi lain melakukan pernyataan DML dan di-commit, hasilnya dapat diprediksi dan sama antara instans primer dan Replika Aurora. Mari kita periksa perbedaan perilaku secara mendetail, dimulai dengan kueri pertama.

Hasil untuk Q1 sangat dapat diprediksi, karena `READ COMMITTED` pada instans primer menggunakan model konsistensi kuat yang serupa dengan tingkat isolasi `REPEATABLE READ`.

Hasil untuk Q2 dapat bervariasi bergantung pada transaksi apa yang dilakukan saat kueri berjalan. Misalnya, transaksi lain melakukan pernyataan DML dan di-commit saat kueri sedang berjalan. Dalam hal ini, kueri di Replika Aurora dengan tingkat isolasi `READ COMMITTED` mungkin atau mungkin tidak memperhitungkan perubahan tersebut. Hitungan baris tidak dapat diprediksi dengan

cara yang sama seperti berdasarkan tingkat isolasi `REPEATABLE READ`. Hitungan baris juga tidak dapat diprediksi seperti kueri yang berjalan berdasarkan tingkat isolasi `READ COMMITTED` pada instans primer, atau pada instans RDS for MySQL.

Pernyataan `UPDATE` di T7 tidak benar-benar mengubah jumlah baris dalam tabel. Namun, dengan mengubah panjang kolom panjang variabel, pernyataan ini dapat menyebabkan baris-baris tersebut disusun ulang secara internal. Transaksi `READ COMMITTED` yang berjalan lama dapat melihat baris versi lama, lalu dalam kueri yang sama, melihat versi baru dari baris yang sama. Kueri ini juga dapat melewati versi baris lama dan baru, sehingga jumlah baris mungkin berbeda dari yang diharapkan.

Hasil Q5 dan Q6 mungkin identik atau sedikit berbeda. Kueri Q6 pada Replika Aurora berdasarkan `READ COMMITTED` dapat melihat, tetapi tidak wajib untuk melihat, baris baru yang di-commit saat kueri berjalan. Kueri ini juga dapat melihat baris dari satu tabel, tetapi tidak dari tabel lainnya. Jika kueri join tidak menemukan baris yang cocok di kedua tabel, kueri ini akan menghasilkan angka nol. Jika kueri ini menemukan baris baru di `PARENT_TABLE` dan `CHILD_TABLE`, kueri tersebut akan menampilkan satu angka. Dalam kueri yang berjalan lama, pencarian dari tabel gabungan dapat terjadi pada waktu yang terpisah secara luas.

> **Note**
>
> Perbedaan perilaku ini bergantung pada waktu transaksi di-commit dan kueri memproses baris tabel yang mendasarinya. Oleh karena itu, kemungkinan besar Anda akan melihat perbedaan tersebut dalam kueri laporan yang memakan waktu beberapa menit atau jam dan yang berjalan di klaster Aurora yang memproses transaksi OLTP pada saat yang sama. Ini adalah jenis beban kerja campuran yang mendapatkan manfaat maksimal dari tingkat isolasi `READ COMMITTED` pada Replika Aurora.

Petunjuk Aurora MySQL

Anda dapat menggunakan petunjuk SQL dengan kueri Aurora MySQL untuk menyesuaikan performa. Anda juga dapat menggunakan petunjuk agar rencana eksekusi untuk kueri penting tidak berubah karena kondisi yang tidak dapat diprediksi.

> **Tip**
>
> Untuk memverifikasi efek petunjuk pada kueri, periksa rencana kueri yang dihasilkan oleh pernyataan `EXPLAIN`. Bandingkan rencana kueri dengan atau tanpa petunjuk.

Di Aurora MySQL versi 3, Anda dapat menggunakan semua petunjuk yang tersedia di MySQL 8.0 Community Edition. Untuk informasi selengkapnya tentang petunjuk ini, lihat Optimizer Hints dalam Panduan Referensi MySQL.

Petunjuk berikut tersedia di Aurora MySQL versi 2. Petunjuk ini berlaku untuk kueri yang menggunakan fitur hash join di Aurora MySQL versi 2, khususnya kueri yang menggunakan optimisasi kueri paralel.

PQ, NO_PQ

Menentukan apakah akan memaksa pengoptimisasi untuk menggunakan kueri paralel berdasarkan per tabel atau per kueri.

`PQ` memaksa pengoptimisasi untuk menggunakan kueri paralel untuk tabel tertentu atau seluruh kueri (blok). `NO_PQ` mencegah pengoptimisasi menggunakan kueri paralel untuk tabel tertentu atau seluruh kueri (blok).

Petunjuk ini tersedia di Aurora MySQL versi 2.11 dan lebih tinggi. Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini.

> **Note**
>
> Menentukan nama tabel akan memaksa pengoptimisasi untuk menerapkan petunjuk `PQ`/`NO_PQ` hanya pada tabel pilihan tersebut. Jika nama tabel tidak ditentukan, petunjuk `PQ`/`NO_PQ` akan dipaksa pada semua tabel yang terpengaruh oleh blok kueri.

```
EXPLAIN SELECT /*+ PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ PQ(t1) */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ PQ(t1,t2) */ f1, f2
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;

EXPLAIN SELECT /*+ NO_PQ() */ f1, f2
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1) */ f1, f2
```

```
    FROM num1 t1 WHERE f1 > 10 and f2 < 100;

EXPLAIN SELECT /*+ NO_PQ(t1,t2) */ f1, f2
    FROM num1 t1, num1 t2 WHERE t1.f1 = t2.f21;
```

HASH_JOIN, NO_HASH_JOIN

Mengaktifkan atau menonaktifkan kemampuan pengoptimisasi kueri paralel untuk memilih apakah akan menggunakan metode optimisasi hash join untuk kueri. `HASH_JOIN` memungkinkan pengoptimisasi menggunakan hash join jika mekanismenya lebih efisien. `NO_HASH_JOIN` mencegah pengoptimisasi menggunakan hash join untuk kueri. Petunjuk ini tersedia di Aurora MySQL versi 2.08 dan lebih tinggi. Ini tidak berpengaruh di Aurora MySQL versi 3.

Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini.

```
EXPLAIN SELECT/*+ HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ NO_HASH_JOIN(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_PROBING, NO_HASH_JOIN_PROBING

Di kueri hash join, tentukan apakah akan menggunakan tabel yang ditentukan untuk sisi probe dari join. Kueri ini menguji apakah nilai kolom dari tabel build ada di tabel probe, bukan membaca seluruh konten tabel probe. Anda dapat menggunakan `HASH_JOIN_PROBING` dan `HASH_JOIN_BUILDING` untuk menentukan cara kueri hash join diproses, tanpa mengatur ulang susunan tabel dalam teks kueri. Petunjuk ini tersedia di Aurora MySQL versi 2.08 dan lebih tinggi. Ini tidak berpengaruh di Aurora MySQL versi 3.

Contoh berikut menunjukkan cara menggunakan petunjuk ini. Menentukan petunjuk `HASH_JOIN_PROBING` untuk tabel `T2` memiliki efek yang sama seperti menentukan `NO_HASH_JOIN_PROBING` untuk tabel `T1`.

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_PROBING(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_PROBING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

HASH_JOIN_BUILDING, NO_HASH_JOIN_BUILDING

Di kueri hash join, tentukan apakah akan menggunakan tabel yang ditentukan untuk sisi build dari join. Kueri ini memproses semua baris dari tabel ini untuk membuat daftar nilai kolom untuk melakukan referensi silang dengan tabel lain. Anda dapat menggunakan `HASH_JOIN_PROBING` dan `HASH_JOIN_BUILDING` untuk menentukan cara kueri hash join diproses, tanpa mengatur ulang susunan tabel dalam teks kueri. Petunjuk ini tersedia di Aurora MySQL versi 2.08 dan lebih tinggi. Ini tidak berpengaruh di Aurora MySQL versi 3.

Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini. Menentukan petunjuk `HASH_JOIN_BUILDING` untuk tabel `T2` memiliki efek yang sama seperti menentukan `NO_HASH_JOIN_BUILDING` untuk tabel `T1`.

```
EXPLAIN SELECT /*+ HASH_JOIN(t2) HASH_JOIN_BUILDING(t2) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;

EXPLAIN SELECT /*+ HASH_JOIN(t2) NO_HASH_JOIN_BUILDING(t1) */ f1, f2
  FROM t1, t2 WHERE t1.f1 = t2.f1;
```

JOIN_FIXED_ORDER

Menentukan bahwa tabel dalam kueri digabungkan berdasarkan urutan pencantumannya dalam kueri. Ini berguna dengan kueri yang memerlukan tiga tabel atau lebih. Petunjuk ini ditujukan sebagai pengganti untuk petunjuk `STRAIGHT_JOIN` MySQL dan setara dengan petunjuk JOIN_FIXED_ORDER MySQL. Petunjuk ini tersedia di Aurora MySQL versi 2.08 dan lebih tinggi.

Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini.

```
EXPLAIN SELECT /*+ JOIN_FIXED_ORDER() */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_ORDER

Menentukan urutan join untuk tabel dalam kueri. Ini berguna dengan kueri yang memerlukan tiga tabel atau lebih. Petunjuk ini setara dengan petunjuk JOIN_ORDER MySQL. Petunjuk ini tersedia di Aurora MySQL versi 2.08 dan lebih tinggi.

Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini.

```
EXPLAIN SELECT /*+ JOIN_ORDER (t4, t2, t1, t3) */ f1, f2
```

```
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_PREFIX

Menentukan tabel yang dimasukkan pertama dalam urutan join. Ini berguna dengan kueri yang memerlukan tiga tabel atau lebih. Petunjuk ini setara dengan petunjuk JOIN_PREFIX MySQL. Petunjuk ini tersedia di Aurora MySQL versi 2.08 dan lebih tinggi.

Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini.

```
EXPLAIN SELECT /*+ JOIN_PREFIX (t4, t2) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

JOIN_SUFFIX

Menentukan tabel yang dimasukkan terakhir dalam urutan join. Ini berguna dengan kueri yang memerlukan tiga tabel atau lebih. Petunjuk ini setara dengan petunjuk JOIN_SUFFIX MySQL. Petunjuk ini tersedia di Aurora MySQL versi 2.08 dan lebih tinggi.

Contoh berikut menunjukkan kepada Anda cara menggunakan petunjuk ini.

```
EXPLAIN SELECT /*+ JOIN_SUFFIX (t1) */ f1, f2
  FROM t1 JOIN t2 USING (id) JOIN t3 USING (id) JOIN t4 USING (id);
```

Untuk informasi tentang menggunakan kueri hash join, lihat Mengoptimalkan kueri join MySQL Aurora besar dengan hash join.

Prosedur tersimpan Aurora MySQL

Anda dapat mengelola klaster DB Aurora MySQL Anda dengan memanggil prosedur tersimpan bawaan.

Topik

- Mengonfigurasi
- Mengakhiri sesi atau kueri
- Pencatatan log
- Mengelola Global Status History
- Mereplikasi

Mengonfigurasi

Prosedur tersimpan berikut mengatur dan menampilkan parameter konfigurasi, seperti untuk retensi file log biner.

Topik

- mysql.rds_set_configuration
- mysql.rds_show_configuration

mysql.rds_set_configuration

Menentukan jumlah jam untuk mempertahankan log biner atau jumlah detik untuk menunda replikasi.

Sintaksis

```
CALL mysql.rds_set_configuration(name,value);
```

Parameter

name

Nama parameter konfigurasi yang akan diatur.

value

Nilai parameter konfigurasi.

Catatan penggunaan

Prosedur `mysql.rds_set_configuration` mendukung parameter konfigurasi berikut:

- binlog retention hours

Parameter konfigurasi disimpan secara permanen dan bertahan dari reboot atau failover instans DB apa pun.

binlog retention hours

Parameter `binlog retention hours` digunakan untuk menentukan jumlah jam untuk mempertahankan file log biner. Amazon Aurora biasanya membersihkan log biner sesegera mungkin, tetapi log biner mungkin masih diperlukan untuk replikasi dengan basis data MySQL di luar Aurora.

Nilai default `binlog retention hours` adalah `NULL`. Untuk Aurora MySQL, `NULL` berarti log biner dibersihkan dengan lambat. Log biner Aurora MySQL mungkin masih berada di dalam sistem untuk jangka waktu tertentu, yang biasanya tidak lebih dari satu hari.

Untuk menentukan jumlah jam untuk mempertahankan log biner pada klaster DB, gunakan prosedur tersimpan `mysql.rds_set_configuration` dan tentukan periode dengan waktu yang cukup untuk terjadinya proses replikasi, seperti yang diperlihatkan dalam contoh berikut.

```
call mysql.rds_set_configuration('binlog retention hours', 24);
```

> **Note**
>
> Anda tidak dapat menggunakan nilai `0` untuk `binlog retention hours`.

Untuk Aurora MySQL versi 2.11.0 dan yang lebih tinggi dan kluster DB versi 3, nilai `binlog retention hours` maksimumnya adalah 2160 (90 hari).

Setelah Anda mengatur periode retensi, pantau penggunaan penyimpanan untuk instans DB guna memastikan bahwa log biner yang dipertahankan tidak memakan terlalu banyak ruang penyimpanan.

mysql.rds_show_configuration

Jumlah jam untuk mempertahankan log biner.

Sintaksis

```
CALL mysql.rds_show_configuration;
```

Catatan penggunaan

Untuk memverifikasi jumlah jam Amazon RDS mempertahankan log biner, gunakan prosedur tersimpan `mysql.rds_show_configuration`.

Contoh

Contoh berikut menampilkan periode retensi:

```
call mysql.rds_show_configuration;
                name                          value     description
                binlog retention hours        24        binlog retention hours specifies
 the duration in hours before binary logs are automatically deleted.
```

Mengakhiri sesi atau kueri

Prosedur tersimpan berikut mengakhiri sesi atau kueri.

Topik

- mysql.rds_kill
- mysql.rds_kill_query

mysql.rds_kill

Mengakhiri koneksi ke server MySQL.

Sintaksis

```
CALL mysql.rds_kill(processID);
```

Parameter

processID

Identitas utas koneksi akan diakhiri.

Catatan penggunaan

Setiap koneksi ke server MySQL berjalan di utas terpisah. Untuk mengakhiri koneksi, gunakan prosedur `mysql.rds_kill` dan teruskan ID utas koneksi tersebut. Untuk mendapatkan ID utas, gunakan perintah SHOW PROCESSLIST MySQL.

Contoh

Contoh berikut mengakhiri koneksi dengan ID utas 4243:

```
CALL mysql.rds_kill(4243);
```

mysql.rds_kill_query

Mengakhiri kueri yang berjalan pada server MySQL.

Sintaksis

```
CALL mysql.rds_kill_query(processID);
```

Parameter

processID

Identitas proses atau utas yang menjalankan kueri yang akan diakhiri.

Catatan penggunaan

Untuk mengakhiri kueri yang berjalan pada server MySQL, gunakan prosedur `mysql_rds_kill_query` dan teruskan ID koneksi utas yang menjalankan kueri. Lalu, prosedur akan mengakhiri koneksi.

Untuk mendapatkan ID, lakukan kueri pada tabel INFORMATION_SCHEMA PROCESSLIST MySQL atau gunakan perintah SHOW PROCESSLIST MySQL. Nilai dalam kolom ID dari `SHOW PROCESSLIST` atau `SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST` adalah *processID*.

Contoh

Contoh berikut mengakhiri kueri dengan ID utas kueri 230040:

```
CALL mysql.rds_kill_query(230040);
```

Pencatatan log

Prosedur tersimpan berikut merotasi log MySQL ke tabel cadangan. Untuk informasi selengkapnya, lihat File log basis data Aurora MySQL.

Topik

- mysql.rds_rotate_general_log
- mysql.rds_rotate_slow_log

mysql.rds_rotate_general_log

Merotasi tabel `mysql.general_log` ke tabel cadangan.

Sintaksis

```
CALL mysql.rds_rotate_general_log;
```

Catatan penggunaan

Anda bisa merotasi tabel `mysql.general_log` ke tabel cadangan dengan memanggil prosedur `mysql.rds_rotate_general_log`. Saat tabel log dirotasi, tabel log saat ini disalin ke tabel log cadangan dan entri di tabel log saat ini dihapus. Jika sudah ada, tabel log cadangan akan dihapus sebelum tabel log saat ini disalin ke cadangan. Anda dapat meminta tabel log cadangan jika diperlukan. Tabel log cadangan untuk tabel `mysql.general_log` bernama `mysql.general_log_backup`.

Anda dapat menjalankan prosedur ini hanya ketika parameter `log_output` diatur ke `TABLE`.

mysql.rds_rotate_slow_log

Merotasi tabel `mysql.slow_log` ke tabel cadangan.

Sintaksis

```
CALL mysql.rds_rotate_slow_log;
```

Catatan penggunaan

Anda bisa merotasi tabel `mysql.slow_log` ke tabel cadangan dengan memanggil prosedur `mysql.rds_rotate_slow_log`. Saat tabel log dirotasi, tabel log saat ini disalin ke tabel log

cadangan dan entri di tabel log saat ini dihapus. Jika sudah ada, tabel log cadangan akan dihapus sebelum tabel log saat ini disalin ke cadangan.

Anda dapat meminta tabel log cadangan jika diperlukan. Tabel log cadangan untuk tabel `mysql.slow_log` bernama `mysql.slow_log_backup`.

Mengelola Global Status History

Amazon RDS menyediakan serangkaian prosedur yang mengambil snapshot nilai-nilai variabel status dari waktu ke waktu dan menuliskannya ke tabel, beserta perubahan apa pun yang dibuat sejak snapshot terakhir. Infrastruktur ini disebut Global Status History. Untuk informasi selengkapnya, lihat Mengelola Global Status History.

Prosedur tersimpan berikut mengelola bagaimana Global Status History dikumpulkan dan dipelihara.

Topik

- mysql.rds_collect_global_status_history
- mysql.rds_disable_gsh_collector
- mysql.rds_disable_gsh_rotation
- mysql.rds_enable_gsh_collector
- mysql.rds_enable_gsh_rotation
- mysql.rds_rotate_global_status_history
- mysql.rds_set_gsh_collector
- mysql.rds_set_gsh_rotation

mysql.rds_collect_global_status_history

Mengambil snapshot sesuai permintaan untuk Global Status History.

Sintaksis

```
CALL mysql.rds_collect_global_status_history;
```

mysql.rds_disable_gsh_collector

Menonaktifkan snapshot yang diambil oleh Global Status History.

Sintaksis

```
CALL mysql.rds_disable_gsh_collector;
```

mysql.rds_disable_gsh_rotation

Menonaktifkan rotasi tabel `mysql.global_status_history`.

Sintaksis

```
CALL mysql.rds_disable_gsh_rotation;
```

mysql.rds_enable_gsh_collector

Mengaktifkan Global Status History untuk mengambil snapshot default pada interval yang ditentukan oleh `rds_set_gsh_collector`.

Sintaksis

```
CALL mysql.rds_enable_gsh_collector;
```

mysql.rds_enable_gsh_rotation

Mengaktifkan rotasi isi tabel `mysql.global_status_history` ke `mysql.global_status_history_old` pada interval yang ditentukan oleh `rds_set_gsh_rotation`.

Sintaksis

```
CALL mysql.rds_enable_gsh_rotation;
```

mysql.rds_rotate_global_status_history

Merotasi isi tabel `mysql.global_status_history` ke `mysql.global_status_history_old` sesuai permintaan.

Sintaksis

```
CALL mysql.rds_rotate_global_status_history;
```

mysql.rds_set_gsh_collector

Menentukan interval, dalam menit, di antara snapshot yang diambil oleh Global Status History.

Sintaksis

```
CALL mysql.rds_set_gsh_collector(intervalPeriod);
```

Parameter

intervalPeriod

Interval, dalam menit, di antara snapshot. Nilai default-nya adalah 5.

mysql.rds_set_gsh_rotation

Menentukan interval, dalam hari, antara rotasi tabel `mysql.global_status_history`.

Sintaksis

```
CALL mysql.rds_set_gsh_rotation(intervalPeriod);
```

Parameter

intervalPeriod

Interval, dalam hari, di antara rotasi tabel. Nilai default-nya adalah 7.

Mereplikasi

Anda dapat memanggil prosedur tersimpan berikut saat terhubung dengan instans primer di klaster Aurora MySQL. Prosedur ini mengontrol bagaimana transaksi direplikasi dari basis data eksternal ke Aurora MySQL, atau dari Aurora MySQL ke basis data eksternal. Untuk mempelajari cara menggunakan replikasi berdasarkan pengidentifikasi transaksi global (GTID) dengan Aurora MySQL, lihat Menggunakan replikasi berbasis GTID untuk Amazon Aurora MySQL.

Topik

- mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL versi 3)
- mysql.rds_disable_session_binlog (Aurora MySQL versi 2)
- mysql.rds_enable_session_binlog (Aurora MySQL versi 2)
- mysql.rds_gtid_purged (Aurora MySQL versi 3)
- mysql.rds_import_binlog_ssl_material
- mysql.rds_next_master_log (Aurora MySQL versi 2)
- mysql.rds_next_source_log (Aurora MySQL versi 3)
- mysql.rds_remove_binlog_ssl_material
- mysql.rds_reset_external_master (Aurora MySQL versi 2)
- mysql.rds_reset_external_source (Aurora MySQL versi 3)
- mysql.rds_set_binlog_source_ssl (Aurora MySQL versi 3)
- mysql.rds_set_external_master (Aurora MySQL versi 2)
- mysql.rds_set_external_master_with_auto_position (Aurora MySQL versi 2)
- mysql.rds_set_external_source (Aurora MySQL versi 3)
- mysql.rds_set_external_source_with_auto_position (Aurora MySQL versi 3)
- mysql.rds_set_master_auto_position (Aurora MySQL versi 2)
- mysql.rds_set_read_only (Aurora MySQL versi 3)
- mysql.rds_set_session_binlog_format (Aurora MySQL versi 2)
- mysql.rds_set_source_auto_position (Aurora MySQL versi 3)
- mysql.rds_skip_transaction_with_gtid (Aurora MySQL versi 2 dan 3)
- mysql.rds_skip_repl_error
- mysql.rds_start_replication
- mysql.rds_start_replication_until (Aurora MySQL versi 3)

- mysql.rds_start_replication_until_gtid (Aurora MySQL versi 3)
- mysql.rds_stop_replication

mysql.rds_assign_gtids_to_anonymous_transactions (Aurora MySQL versi 3)

Mengonfigurasi opsi `ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS` dari pernyataan `CHANGE REPLICATION SOURCE TO`. Hal ini akan membuat saluran replikasi menetapkan GTID ke transaksi replikasi yang tidak memilikinya. Dengan begitu, Anda dapat melakukan replikasi log biner dari sumber yang tidak menggunakan replikasi berbasis GTID ke replika yang menggunakan replikasi tersebut. Untuk informasi selengkapnya, lihat CHANGE REPLICATION SOURCE TO Statement dan Replication From a Source Without GTIDs to a Replica With GTIDs di Panduan Referensi MySQL.

Sintaks

```
CALL mysql.rds_assign_gtids_to_anonymous_transactions(gtid_option);
```

Parameter-parameter

`gtid_option`

Nilai string. Nilai yang diizinkan adalah `OFF`, `LOCAL`, atau UUID yang ditentukan.

Catatan penggunaan

Prosedur ini memiliki efek yang sama seperti mengeluarkan pernyataan `CHANGE REPLICATION SOURCE TO ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS =` *`gtid_option`* di komunitas MySQL.

GTID harus diubah menjadi `ON` agar *`gtid_option`* dapat diatur ke `LOCAL` atau UUID spesifik.

Defaultnya adalah `OFF`, yang artinya fitur tersebut tidak digunakan.

`LOCAL` menetapkan GTID termasuk UUID replika itu sendiri (pengaturan `server_uuid`).

Meneruskan parameter yang merupakan UUID akan menetapkan GTID yang menyertakan UUID tertentu, seperti pengaturan `server_uuid` untuk server sumber replikasi.

Contoh-contoh

Untuk menonaktifkan fitur ini:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('OFF');
+-------------------------------------------------------------+
| Message  |
+-------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: OFF |
+-------------------------------------------------------------+
1 row in set (0.07 sec)
```

Untuk menggunakan UUID replika itu sendiri:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('LOCAL');
+------------------------------------------------------------+
| Message  |
+------------------------------------------------------------+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: LOCAL |
+------------------------------------------------------------+
1 row in set (0.07 sec)
```

Untuk menggunakan UUID yang ditentukan:

```
mysql> call mysql.rds_assign_gtids_to_anonymous_transactions('317a4760-
f3dd-3b74-8e45-0615ed29de0e');
+-----------------------------------------------------------------------------------------------------------
+
| Message |
+-----------------------------------------------------------------------------------------------------------
+
| ASSIGN_GTIDS_TO_ANONYMOUS_TRANSACTIONS has been set to: 317a4760-
f3dd-3b74-8e45-0615ed29de0e |
+-----------------------------------------------------------------------------------------------------------
+
1 row in set (0.07 sec)
```

mysql.rds_disable_session_binlog (Aurora MySQL versi 2)

Menonaktifkan pencatatan log biner untuk sesi saat ini dengan mengatur variabel `sql_log_bin` ke `OFF`.

Sintaksis

```
CALL mysql.rds_disable_session_binlog;
```

Parameter-parameter

Tidak ada

Catatan penggunaan

Untuk klaster DB Aurora MySQL, Anda memanggil prosedur tersimpan ini saat terhubung ke instans primer.

Untuk Aurora, prosedur ini didukung untuk Aurora MySQL versi 2.12 dan versi yang lebih tinggi yang kompatibel dengan MySQL 5.7.

> **Note**
>
> Di Aurora MySQL versi 3, Anda dapat menggunakan perintah berikut untuk menonaktifkan pencatatan log biner untuk sesi saat ini jika Anda memiliki hak istimewa `SESSION_VARIABLES_ADMIN`:
>
> ```
> SET SESSION sql_log_bin = OFF;
> ```

mysql.rds_enable_session_binlog (Aurora MySQL versi 2)

Mengaktifkan pencatatan log biner untuk sesi saat ini dengan mengatur variabel `sql_log_bin` ke `ON`.

Sintaksis

```
CALL mysql.rds_enable_session_binlog;
```

Parameter-parameter

Tidak ada

Catatan penggunaan

Untuk klaster DB Aurora MySQL, Anda memanggil prosedur tersimpan ini saat terhubung ke instans primer.

Untuk Aurora, prosedur ini didukung untuk Aurora MySQL versi 2.12 dan versi yang lebih tinggi yang kompatibel dengan MySQL 5.7.

> **Note**
>
> Di Aurora MySQL versi 3, Anda dapat menggunakan perintah berikut untuk mengaktifkan pencatatan log biner untuk sesi saat ini jika Anda memiliki hak istimewa `SESSION_VARIABLES_ADMIN`:
>
> ```
> SET SESSION sql_log_bin = ON;
> ```

mysql.rds_gtid_purged (Aurora MySQL versi 3)

Menetapkan nilai global variabel sistem `gtid_purged` ke set pengidentifikasi transaksi global (GTID) yang diberikan. Variabel sistem `gtid_purged` adalah set GTID yang terdiri dari GTID dari semua transaksi yang telah dilakukan di server, tetapi tidak ada dalam file log biner apa pun di server.

Untuk memungkinkan kompatibilitas dengan MySQL 8.0, ada dua cara untuk mengatur nilai `gtid_purged`:

- Ganti nilai `gtid_purged` dengan set GTID yang Anda tentukan.
- Tambahkan set GTID yang Anda tentukan ke set GTID yang sudah ada di `gtid_purged`.

Sintaksis

Untuk mengganti nilai `gtid_purged` dengan set GTID yang Anda tentukan:

```
CALL mysql.rds_gtid_purged (gtid_set);
```

Untuk menambahkan nilai `gtid_purged` ke set GTID yang Anda tentukan:

```
CALL mysql.rds_gtid_purged (+gtid_set);
```

Parameter-parameter

`gtid_set`

Nilai *`gtid_set`* harus merupakan superset dari nilai `gtid_purged` saat ini, dan tidak boleh berpotongan dengan `gtid_subtract(gtid_executed,gtid_purged)`. Artinya, set GTID baru harus menyertakan semua GTID yang sudah ada di `gtid_purged`, dan tidak boleh

menyertakan GTID apa pun di `gtid_executed` yang belum dihapus. Parameter *`gtid_set`* juga tidak dapat menyertakan GTID apa pun yang ada di set `gtid_owned` global, GTID untuk transaksi yang saat ini sedang diproses di server.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_gtid_purged`.

Prosedur ini didukung untuk Aurora MySQL versi 3.04 dan yang lebih tinggi.

Contoh-contoh

Contoh berikut menetapkan GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23` ke variabel global `gtid_purged`.

```
CALL mysql.rds_gtid_purged('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_import_binlog_ssl_material

Mengimpor sertifikat otoritas sertifikat, sertifikat klien, dan kunci klien ke klaster DB Aurora MySQL. Informasi tersebut diperlukan untuk komunikasi SSL dan replikasi terenkripsi.

> **Note**
>
> Saat ini, prosedur ini didukung untuk Aurora MySQL versi 2:2.09.2, 2.10.0, 2.10.1, dan 2.11.0; serta versi 3:3.01.1 dan yang lebih tinggi.

Sintaksis

```
CALL mysql.rds_import_binlog_ssl_material (
  ssl_material
);
```

Parameter

`ssl_material`

Payload JSON yang berisi konten file berformat .pem berikut untuk klien MySQL:

- "ssl_ca":"*Sertifikat otoritas sertifikat*"
- "ssl_cert":"*Sertifikat klien*"
- "ssl_key":"*Kunci klien*"

Catatan penggunaan

Persiapkan untuk replikasi terenkripsi sebelum Anda menjalankan prosedur ini:

- Jika Anda tidak memiliki SSL yang diaktifkan pada instans basis data sumber MySQL eksternal dan tidak menyiapkan kunci klien dan sertifikat klien, aktifkan SSL pada server basis data MySQL serta buat kunci klien dan sertifikat klien yang diperlukan.
- Jika SSL diaktifkan pada instans basis data sumber eksternal, sediakan kunci dan sertifikat klien untuk klaster DB Aurora MySQL. Jika Anda tidak memilikinya, buat kunci dan sertifikat baru untuk klaster DB Aurora MySQL. Untuk menandatangani sertifikat klien, Anda harus memiliki kunci otoritas sertifikat yang Anda gunakan untuk mengonfigurasi SSL pada instans basis data sumber MySQL eksternal.

Untuk informasi selengkapnya, lihat Creating SSL certificates and keys using openssl dalam dokumentasi MySQL.

> **Important**
>
> Setelah Anda mempersiapkan replikasi terenkripsi, gunakan koneksi SSL untuk menjalankan prosedur ini. Kunci klien tidak boleh ditransfer melalui koneksi yang tidak aman.

Prosedur ini mengimpor informasi SSL dari basis data MySQL eksternal ke klaster DB Aurora MySQL. Informasi SSL dalam file berformat .pem yang berisi informasi SSL untuk klaster DB Aurora MySQL. Selama replikasi terenkripsi, klaster DB Aurora MySQL berperan sebagai klien untuk server basis data MySQL. Sertifikat dan kunci untuk klien Aurora MySQL ada di dalam file berformat .pem.

Anda dapat menyalin informasi dari file ini ke dalam parameter `ssl_material` dalam payload JSON yang benar. Untuk mendukung replikasi terenkripsi, impor informasi SSL ini ke klaster DB Aurora MySQL.

Payload JSON harus dalam format berikut.

```
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
ssl_ca_pem_body_code
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
ssl_cert_pem_body_code
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
ssl_key_pem_body_code
-----END RSA PRIVATE KEY-----\n"}'
```

Contoh-contoh

Contoh berikut mengimpor informasi SSL ke Aurora MySQL. Dalam file berformat .pem, kode konten biasanya lebih panjang dari kode konten yang ditunjukkan dalam contoh.

```
call mysql.rds_import_binlog_ssl_material(
'{"ssl_ca":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_cert":"-----BEGIN CERTIFICATE-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----
AAAAB3NzaC1yc2EAAAADAQABAAABAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS7O6V
hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr
lsLnBItntckiJ7FbtxJMXLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0FzZ
qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb
BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE
-----END RSA PRIVATE KEY-----\n"}');
```

mysql.rds_next_master_log (Aurora MySQL versi 2)

Mengubah posisi log instans basis data sumber menjadi awal log biner berikutnya pada instans basis data sumber. Gunakan prosedur ini hanya jika Anda menerima kesalahan I/O 1236 replikasi pada replika baca.

Sintaksis

```
CALL mysql.rds_next_master_log(
curr_master_log
);
```

Parameter

curr_master_log

Indeks file log master saat ini. Misalnya, jika file saat ini bernama `mysql-bin-changelog.012345`, maka indeksnya adalah 12345. Untuk menentukan nama file log master saat ini, jalankan perintah `SHOW REPLICA STATUS` dan lihat kolom `Master_Log_File`.

> **Note**
>
> Versi MySQL sebelumnya menggunakan `SHOW SLAVE STATUS`, bukan `SHOW REPLICA STATUS`. Jika Anda menggunakan versi MySQL sebelum 8.0.23, gunakan `SHOW SLAVE STATUS`.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_next_master_log`.

> **Warning**
>
> Panggil `mysql.rds_next_master_log` hanya jika replikasi gagal setelah failover dari instans DB Multi-AZ yang merupakan sumber replikasi, dan kolom `Last_IO_Errno` dari laporan kesalahan I/O 1236 `SHOW REPLICA STATUS`.
> Memanggil `mysql.rds_next_master_log` dapat mengakibatkan hilangnya data di replika baca jika transaksi dalam instans sumber tidak ditulis ke log biner di disk sebelum peristiwa failover terjadi.

Contoh-contoh

Asumsikan replikasi gagal pada replika baca Aurora MySQL. Menjalankan `SHOW REPLICA STATUS \G` pada replika baca akan menampilkan hasil berikut:

```
*************************** 1. row ***************************
             Replica_IO_State:
```

```
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Source_SSL_Allowed: No
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
        Seconds_Behind_Master: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from master when reading data from
 binary log: 'Client requested master to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

Kolom `Last_IO_Errno` menunjukkan bahwa instans menerima kesalahan I/O 1236. Kolom `Master_Log_File` menunjukkan bahwa nama file adalah `mysql-bin-changelog.012345`, yang berarti indeks file log adalah `12345`. Untuk mengatasi kesalahan, Anda dapat memanggil `mysql.rds_next_master_log` dengan parameter berikut:

```
CALL mysql.rds_next_master_log(12345);
```

> **Note**
>
> Versi MySQL sebelumnya menggunakan `SHOW SLAVE STATUS`, bukan `SHOW REPLICA STATUS`. Jika Anda menggunakan versi MySQL sebelum 8.0.23, gunakan `SHOW SLAVE STATUS`.

mysql.rds_next_source_log (Aurora MySQL versi 3)

Mengubah posisi log instans basis data sumber menjadi awal log biner berikutnya pada instans basis data sumber. Gunakan prosedur ini hanya jika Anda menerima kesalahan I/O 1236 replikasi pada replika baca.

Sintaksis

```
CALL mysql.rds_next_source_log(
curr_source_log
);
```

Parameter

curr_source_log

Indeks file log sumber saat ini. Misalnya, jika file saat ini bernama `mysql-bin-changelog.012345`, maka indeksnya adalah 12345. Untuk menentukan nama file log sumber saat ini, jalankan perintah `SHOW REPLICA STATUS` dan lihat kolom `Source_Log_File`.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_next_source_log`.

> **Warning**
>
> Panggil `mysql.rds_next_source_log` hanya jika replikasi gagal setelah failover dari instans DB Multi-AZ yang merupakan sumber replikasi, dan kolom `Last_IO_Errno` dari laporan kesalahan I/O 1236 `SHOW REPLICA STATUS`.
> Memanggil `mysql.rds_next_source_log` dapat mengakibatkan hilangnya data di replika baca jika transaksi dalam instans sumber tidak ditulis ke log biner di disk sebelum peristiwa failover terjadi.

Contoh-contoh

Asumsikan replikasi gagal pada replika baca Aurora MySQL. Menjalankan `SHOW REPLICA STATUS \G` pada replika baca akan menampilkan hasil berikut:

```
*************************** 1. row ***************************
             Replica_IO_State:
                  Source_Host: myhost.XXXXXXXXXXXXXX.rr-rrrr-1.rds.amazonaws.com
                  Source_User: MasterUser
                  Source_Port: 3306
                Connect_Retry: 10
              Source_Log_File: mysql-bin-changelog.012345
          Read_Source_Log_Pos: 1219393
               Relay_Log_File: relaylog.012340
                Relay_Log_Pos: 30223388
        Relay_Source_Log_File: mysql-bin-changelog.012345
           Replica_IO_Running: No
          Replica_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Source_Log_Pos: 30223232
              Relay_Log_Space: 5248928866
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
```

```
           Source_SSL_Allowed: No
           Source_SSL_CA_File:
           Source_SSL_CA_Path:
              Source_SSL_Cert:
            Source_SSL_Cipher:
               Source_SSL_Key:
        Seconds_Behind_Source: NULL
Source_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 1236
                Last_IO_Error: Got fatal error 1236 from source when reading data from
 binary log: 'Client requested source to start replication from impossible position;
 the first event 'mysql-bin-changelog.013406' at 1219393, the last event read from
 '/rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4, the last byte read from '/
rdsdbdata/log/binlog/mysql-bin-changelog.012345' at 4.'
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Source_Server_Id: 67285976
```

Kolom `Last_IO_Errno` menunjukkan bahwa instans menerima kesalahan I/O 1236. Kolom `Source_Log_File` menunjukkan bahwa nama file adalah `mysql-bin-changelog.012345`, yang berarti indeks file log adalah `12345`. Untuk mengatasi kesalahan, Anda dapat memanggil `mysql.rds_next_source_log` dengan parameter berikut:

```
CALL mysql.rds_next_source_log(12345);
```

mysql.rds_remove_binlog_ssl_material

Menghapus sertifikat otoritas sertifikat, sertifikat klien, dan kunci klien untuk komunikasi SSL dan replikasi terenkripsi. Informasi ini diimpor menggunakan mysql.rds_import_binlog_ssl_material.

Sintaksis

```
CALL mysql.rds_remove_binlog_ssl_material;
```

mysql.rds_reset_external_master (Aurora MySQL versi 2)

Mengonfigurasi ulang instans DB Aurora MySQL agar tidak lagi menjadi replika baca dari instans MySQL yang berjalan di luar Amazon RDS.

> **Important**
>
> Untuk menjalankan prosedur ini, `autocommit` harus diaktifkan. Untuk mengaktifkannya, atur parameter `autocommit` ke 1. Lihat informasi tentang cara mengubah parameter di Memodifikasi parameter dalam grup parameter DB.

Sintaksis

```
CALL mysql.rds_reset_external_master;
```

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_reset_external_master`. Prosedur ini harus dijalankan pada instans DB MySQL agar dihapus sebagai replika baca dari instans MySQL yang berjalan di luar Amazon RDS.

> **Note**
>
> Kami menawarkan prosedur tersimpan ini terutama untuk mengaktifkan replikasi dengan instans MySQL yang berjalan di luar Amazon RDS. Kami menyarankan Anda menggunakan Aurora Replicas untuk mengelola replikasi dalam klaster DB Aurora MySQL jika memungkinkan. Untuk informasi tentang cara mengelola replikasi di klaster DB Aurora MySQL, lihat Menggunakan Replika Aurora.

Untuk informasi selengkapnya tentang cara menggunakan replikasi untuk mengimpor data dari instans MySQL yang berjalan di luar Aurora MySQL, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner).

mysql.rds_reset_external_source (Aurora MySQL versi 3)

Mengonfigurasi ulang instans DB Aurora MySQL agar tidak lagi menjadi replika baca dari instans MySQL yang berjalan di luar Amazon RDS.

> **Important**
>
> Untuk menjalankan prosedur ini, `autocommit` harus diaktifkan. Untuk mengaktifkannya, atur parameter `autocommit` ke 1. Lihat informasi tentang cara mengubah parameter di Memodifikasi parameter dalam grup parameter DB.

Sintaksis

```
CALL mysql.rds_reset_external_source;
```

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_reset_external_source`. Prosedur ini harus dijalankan pada instans DB MySQL agar dihapus sebagai replika baca dari instans MySQL yang berjalan di luar Amazon RDS.

> **Note**
>
> Kami menawarkan prosedur tersimpan ini terutama untuk mengaktifkan replikasi dengan instans MySQL yang berjalan di luar Amazon RDS. Kami menyarankan Anda menggunakan Aurora Replicas untuk mengelola replikasi dalam klaster DB Aurora MySQL jika memungkinkan. Untuk informasi tentang cara mengelola replikasi di klaster DB Aurora MySQL, lihat Menggunakan Replika Aurora.

mysql.rds_set_binlog_source_ssl (Aurora MySQL versi 3)

Mengaktifkan `SOURCE_SSL` enkripsi untuk replikasi binlog. Untuk informasi selengkapnya, lihat MENGUBAH SUMBER REPLIKASI KE pernyataan dalam dokumentasi MySQL.

Sintaks

```
CALL mysql.rds_set_binlog_source_ssl(mode);
```

Parameter-parameter

modus

Nilai yang menunjukkan apakah SOURCE_SSL enkripsi diaktifkan:

- 0— SOURCE_SSL enkripsi dinonaktifkan. Nilai default-nya 0.
- 1— SOURCE_SSL enkripsi diaktifkan. Anda dapat mengonfigurasi enkripsi menggunakan SSL atau TLS.

Catatan penggunaan

Prosedur ini didukung untuk Aurora MySQL versi 3.06 dan lebih tinggi.

mysql.rds_set_external_master (Aurora MySQL versi 2)

Mengonfigurasi instans DB Aurora MySQL agar tidak lagi menjadi replika baca dari instans MySQL yang berjalan di luar Amazon RDS.

Prosedur `mysql.rds_set_external_master` tidak digunakan lagi dan akan dihapus dalam rilis mendatang. Gunakan `mysql.rds_set_external_source` sebagai gantinya.

> **Important**
>
> Untuk menjalankan prosedur ini, `autocommit` harus diaktifkan. Untuk mengaktifkannya, atur parameter `autocommit` ke 1. Lihat informasi tentang cara mengubah parameter di Memodifikasi parameter dalam grup parameter DB.

Sintaksis

```
CALL mysql.rds_set_external_master (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

Parameter

host_name

Nama host atau alamat IP instans MySQL yang berjalan di luar Amazon RDS untuk menjadi instans basis data sumber.

host_port

Port yang digunakan oleh instans MySQL yang berjalan di luar Amazon RDS untuk dikonfigurasi sebagai instans basis data sumber. Jika konfigurasi jaringan Anda mencakup replikasi port Secure Shell (SSH) yang mengubah nomor port, tentukan nomor port yang diekspos oleh SSH.

replication_user_name

ID pengguna dengan izin `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada instans MySQL yang berjalan di luar Amazon RDS. Kami menyarankan Anda memberikan akun yang digunakan sepenuhnya untuk replikasi dengan instans eksternal.

replication_user_password

Kata sandi ID pengguna yang ditentukan di `replication_user_name`.

mysql_binary_log_file_name

Nama log biner pada instans basis data sumber yang berisi informasi replikasi.

mysql_binary_log_file_location

Lokasi di log biner `mysql_binary_log_file_name` tempat replikasi mulai membaca informasi replikasi.

Anda dapat menentukan nama dan lokasi file binlog dengan menjalankan `SHOW MASTER STATUS` pada instans basis data sumber.

ssl_encryption

Nilai yang menentukan apakah enkripsi Lapisan Soket Aman (SSL) digunakan pada sambungan replikasi. 1 menentukan untuk menggunakan enkripsi SSL, 0 menentukan untuk tidak menggunakan enkripsi. Default-nya adalah 0.

> **Note**
>
> Opsi `MASTER_SSL_VERIFY_SERVER_CERT` tidak didukung. Opsi ini diatur ke 0, yang berarti koneksi dienkripsi, tetapi sertifikat tidak diverifikasi.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_set_external_master`. Prosedur ini harus dijalankan pada instans DB MySQL untuk dikonfigurasi sebagai replika baca dari instans MySQL yang berjalan di luar Amazon RDS.

Sebelum menjalankan `mysql.rds_set_external_master`, Anda harus mengonfigurasi instans MySQL yang berjalan di luar Amazon RDS menjadi instans basis data sumber. Untuk terhubung ke instans MySQL yang berjalan di luar Amazon RDS, Anda harus menentukan nilai `replication_user_name` dan `replication_user_password` yang menunjukkan pengguna replikasi yang memiliki izin `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada instans eksternal MySQL.

Untuk mengonfigurasi instans eksternal MySQL sebagai instans basis data sumber

1. Dengan menggunakan klien MySQL pilihan Anda, hubungkan ke instans eksternal MySQL dan buat akun pengguna yang akan digunakan untuk replikasi. Berikut adalah contohnya.

   MySQL 5.7

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
   ```

   MySQL 8.0

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED WITH mysql_native_password BY
    'password';
   ```

   > **Note**
   >
   > Tetapkan kata sandi selain penggugah (prompt) yang ditampilkan di sini sebagai praktik terbaik keamanan.

2. Pada instans eksternal MySQL, berikan hak istimewa `REPLICATION CLIENT` dan `REPLICATION SLAVE` kepada pengguna replikasi Anda. Contoh berikut memberikan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada semua basis data untuk pengguna ‘repl_user’ domain Anda.

   MySQL 5.7

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
 IDENTIFIED BY 'password';
```

MySQL 8.0

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
```

Untuk menggunakan replikasi terenkripsi, konfigurasikan instans basis data sumber untuk menggunakan koneksi SSL. Selain itu, impor sertifikat otoritas sertifikat, sertifikat klien, dan kunci klien ke dalam instans DB atau klaster DB menggunakan prosedur mysql.rds_import_binlog_ssl_material.

> **Note**
>
> Kami menawarkan prosedur tersimpan ini terutama untuk mengaktifkan replikasi dengan instans MySQL yang berjalan di luar Amazon RDS. Kami menyarankan Anda menggunakan Aurora Replicas untuk mengelola replikasi dalam klaster DB Aurora MySQL jika memungkinkan. Untuk informasi tentang cara mengelola replikasi di klaster DB Aurora MySQL, lihat Menggunakan Replika Aurora.

Setelah memanggil `mysql.rds_set_external_master` untuk mengonfigurasi instans DB Amazon RDS sebagai replika baca, Anda dapat memanggil mysql.rds_start_replication pada replika baca untuk memulai proses replikasi. Anda dapat memanggil mysql.rds_reset_external_master (Aurora MySQL versi 2) untuk menghapus konfigurasi replika baca.

Saat `mysql.rds_set_external_master` dipanggil, Amazon RDS mencatat waktu, pengguna, dan tindakan `set master` di tabel `mysql.rds_history` dan `mysql.rds_replication_status`.

Contoh-contoh

Ketika dijalankan pada instans DB MySQL, contoh berikut mengonfigurasi instans DB menjadi replika baca dari instans MySQL yang berjalan di luar Amazon RDS.

```
call mysql.rds_set_external_master(
  'Externaldb.some.com',
  3306,
```

```
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  0);
```

mysql.rds_set_external_master_with_auto_position (Aurora MySQL versi 2)

Mengonfigurasi instans primer Aurora MySQL untuk menerima replikasi masuk dari instans MySQL eksternal. Prosedur ini juga mengonfigurasi replikasi berdasarkan pengidentifikasi transaksi global (GTID).

Prosedur ini tidak mengonfigurasi replikasi tertunda, karena Aurora MySQL tidak mendukung replikasi tertunda.

Sintaksis

```
CALL mysql.rds_set_external_master_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

Parameter

host_name

Nama host atau alamat IP instans MySQL yang berjalan di luar Aurora untuk menjadi master replikasi.

host_port

Port yang digunakan oleh instans MySQL yang berjalan di luar Aurora untuk dikonfigurasi sebagai master replikasi. Jika konfigurasi jaringan Anda mencakup replikasi port Secure Shell (SSH) yang mengubah nomor port, tentukan nomor port yang diekspos oleh SSH.

replication_user_name

ID pengguna dengan izin `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada instans MySQL yang berjalan di luar Aurora. Kami menyarankan Anda memberikan akun yang digunakan sepenuhnya untuk replikasi dengan instans eksternal.

replication_user_password

Kata sandi ID pengguna yang ditentukan dalam `replication_user_name`.

ssl_encryption

Opsi ini belum diterapkan. Default-nya adalah 0.

Catatan penggunaan

Untuk klaster DB Aurora MySQL, Anda memanggil prosedur tersimpan ini saat terhubung ke instans primer.

Pengguna utama harus menjalankan prosedur `mysql.rds_set_external_master_with_auto_position`. Pengguna utama menjalankan prosedur ini pada instans primer dari klaster DB Aurora MySQL yang bertindak sebagai target replikasi. Ini dapat menjadi target replikasi dari instans DB MySQL eksternal atau klaster DB Aurora MySQL.

Prosedur ini didukung untuk Aurora MySQL versi 2. Untuk Aurora MySQL versi 3, gunakan prosedur mysql.rds_set_external_source_with_auto_position (Aurora MySQL versi 3) sebagai gantinya.

Sebelum Anda menjalankan `mysql.rds_set_external_master_with_auto_position`, konfigurasikan instans DB MySQL eksternal untuk menjadi master replikasi. Agar terhubung ke instans MySQL eksternal, tentukan nilai untuk `replication_user_name` dan `replication_user_password`. Nilai-nilai ini harus menunjukkan pengguna replikasi yang memiliki `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada instans MySQL eksternal.

Untuk mengonfigurasi instans MySQL eksternal sebagai master replikasi

1. Dengan menggunakan klien MySQL pilihan Anda, hubungkan ke instans MySQL eksternal dan buat akun pengguna yang akan digunakan untuk replikasi. Berikut adalah contohnya.

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
   ```

2. Pada instans MySQL eksternal, berikan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE` kepada pengguna replikasi Anda. Contoh berikut memberikan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada semua basis data untuk pengguna `'repl_user'` domain Anda.

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
   ```

```
IDENTIFIED BY 'SomePassW0rd'
```

Saat Anda memanggil `mysql.rds_set_external_master_with_auto_position`, Amazon RDS mencatat informasi tertentu. Informasi ini adalah waktu, pengguna, dan tindakan "`set master`" di tabel `mysql.rds_history` dan `mysql.rds_replication_status`.

Untuk melewati transaksi berbasis GTID tertentu yang diketahui menyebabkan masalah, Anda dapat menggunakan prosedur tersimpan mysql.rds_skip_transaction_with_gtid. Untuk informasi selengkapnya tentang cara menggunakan replikasi berbasis GTID, lihat Menggunakan replikasi berbasis GTID untuk Amazon Aurora MySQL.

Contoh-contoh

Saat menjalankan di instans primer Aurora, contoh berikut mengonfigurasi klaster Aurora untuk bertindak sebagai replika baca dari instans MySQL yang berjalan di luar Aurora.

```
call mysql.rds_set_external_master_with_auto_position(
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassW0rd');
```

mysql.rds_set_external_source (Aurora MySQL versi 3)

Mengonfigurasi instans DB Aurora MySQL menjadi replika baca dari instans MySQL yang berjalan di luar Amazon RDS.

> **Important**
>
> Untuk menjalankan prosedur ini, `autocommit` harus diaktifkan. Untuk mengaktifkannya, atur parameter `autocommit` ke 1. Lihat informasi tentang cara mengubah parameter di Memodifikasi parameter dalam grup parameter DB.

Sintaksis

```
CALL mysql.rds_set_external_source (
```

```
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , mysql_binary_log_file_name
  , mysql_binary_log_file_location
  , ssl_encryption
);
```

Parameter

host_name

Nama host atau alamat IP instans MySQL yang berjalan di luar Amazon RDS untuk menjadi instans basis data sumber.

host_port

Port yang digunakan oleh instans MySQL yang berjalan di luar Amazon RDS untuk dikonfigurasi sebagai instans basis data sumber. Jika konfigurasi jaringan Anda mencakup replikasi port Secure Shell (SSH) yang mengubah nomor port, tentukan nomor port yang diekspos oleh SSH.

replication_user_name

ID pengguna dengan izin `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada instans MySQL yang berjalan di luar Amazon RDS. Kami menyarankan Anda memberikan akun yang digunakan sepenuhnya untuk replikasi dengan instans eksternal.

replication_user_password

Kata sandi ID pengguna yang ditentukan di `replication_user_name`.

mysql_binary_log_file_name

Nama log biner pada instans basis data sumber yang berisi informasi replikasi.

mysql_binary_log_file_location

Lokasi di log biner `mysql_binary_log_file_name` tempat replikasi mulai membaca informasi replikasi.

Anda dapat menentukan nama dan lokasi file binlog dengan menjalankan `SHOW MASTER STATUS` pada instans basis data sumber.

ssl_encryption

Nilai yang menentukan apakah enkripsi Lapisan Soket Aman (SSL) digunakan pada sambungan replikasi. 1 menentukan untuk menggunakan enkripsi SSL, 0 menentukan untuk tidak menggunakan enkripsi. Standarnya adalah 0.

> **Note**
>
> Anda harus mengimpor sertifikat SSL khusus mysql.rds_import_binlog_ssl_material untuk mengaktifkan opsi ini. Jika Anda belum mengimpor sertifikat SSL kustom, maka atur parameter ini ke 0 dan gunakan untuk mengaktifkan SSL mysql.rds_set_binlog_source_ssl (Aurora MySQL versi 3) untuk replikasi log biner.
> Opsi `MASTER_SSL_VERIFY_SERVER_CERT` tidak didukung. Opsi ini diatur ke 0, yang berarti koneksi dienkripsi, tetapi sertifikat tidak diverifikasi.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_set_external_source`. Prosedur ini harus dijalankan pada instans DB Aurora MySQL untuk dikonfigurasi sebagai replika baca dari instans MySQL yang berjalan di luar Amazon RDS.

Sebelum menjalankan `mysql.rds_set_external_source`, Anda harus mengonfigurasi instans MySQL yang berjalan di luar Amazon RDS menjadi instans basis data sumber. Untuk terhubung ke instans MySQL yang berjalan di luar Amazon RDS, Anda harus menentukan nilai `replication_user_name` dan `replication_user_password` yang menunjukkan pengguna replikasi yang memiliki izin `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada instans eksternal MySQL.

Untuk mengonfigurasi instans eksternal MySQL sebagai instans basis data sumber

1. Dengan menggunakan klien MySQL pilihan Anda, hubungkan ke instans eksternal MySQL dan buat akun pengguna yang akan digunakan untuk replikasi. Berikut adalah contohnya.

   MySQL 5.7

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'password';
   ```

   MySQL 8.0

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED WITH mysql_native_password BY
 'password';
```

> **Note**
>
> Tetapkan kata sandi selain penggugah (prompt) yang ditampilkan di sini sebagai praktik terbaik keamanan.

2. Pada instans eksternal MySQL, berikan hak istimewa `REPLICATION CLIENT` dan `REPLICATION SLAVE` kepada pengguna replikasi Anda. Contoh berikut memberikan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada semua basis data untuk pengguna 'repl_user' domain Anda.

   MySQL 5.7

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
    IDENTIFIED BY 'password';
   ```

   MySQL 8.0

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com';
   ```

Untuk menggunakan replikasi terenkripsi, konfigurasikan instans basis data sumber untuk menggunakan koneksi SSL. Selain itu, impor sertifikat otoritas sertifikat, sertifikat klien, dan kunci klien ke dalam instans DB atau klaster DB menggunakan prosedur mysql.rds_import_binlog_ssl_material.

> **Note**
>
> Kami menawarkan prosedur tersimpan ini terutama untuk mengaktifkan replikasi dengan instans MySQL yang berjalan di luar Amazon RDS. Kami menyarankan Anda menggunakan Aurora Replicas untuk mengelola replikasi dalam klaster DB Aurora MySQL jika memungkinkan. Untuk informasi tentang cara mengelola replikasi di klaster DB Aurora MySQL, lihat Menggunakan Replika Aurora.

Setelah memanggil `mysql.rds_set_external_source` untuk mengonfigurasi instans DB Aurora MySQL sebagai replika baca, Anda dapat memanggil mysql.rds_start_replication pada replika baca untuk memulai proses replikasi. Anda dapat memanggil mysql.rds_external_source untuk menghapus konfigurasi replika baca.

Saat `mysql.rds_set_external_source` dipanggil, Amazon RDS mencatat waktu, pengguna, dan tindakan `set master` di tabel `mysql.rds_history` dan `mysql.rds_replication_status`.

Contoh-contoh

Saat menjalankan pada instans DB Aurora MySQL, contoh berikut mengonfigurasi instans DB menjadi replika baca dari instans MySQL yang berjalan di luar Amazon RDS.

```
call mysql.rds_set_external_source(
  'Externaldb.some.com',
  3306,
  'repl_user',
  'password',
  'mysql-bin-changelog.0777',
  120,
  0);
```

mysql.rds_set_external_source_with_auto_position (Aurora MySQL versi 3)

Mengonfigurasi instans primer Aurora MySQL untuk menerima replikasi masuk dari instans MySQL eksternal. Prosedur ini juga mengonfigurasi replikasi berdasarkan pengidentifikasi transaksi global (GTID).

Sintaksis

```
CALL mysql.rds_set_external_source_with_auto_position (
  host_name
  , host_port
  , replication_user_name
  , replication_user_password
  , ssl_encryption
);
```

Parameter

host_name

Nama host atau alamat IP instans MySQL yang berjalan di luar Aurora untuk menjadi sumber replikasi.

host_port

Port yang digunakan oleh instans MySQL yang berjalan di luar Aurora untuk dikonfigurasi sebagai sumber replikasi. Jika konfigurasi jaringan Anda mencakup replikasi port Secure Shell (SSH) yang mengubah nomor port, tentukan nomor port yang diekspos oleh SSH.

replication_user_name

ID pengguna dengan izin `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada instans MySQL yang berjalan di luar Aurora. Kami menyarankan Anda memberikan akun yang digunakan sepenuhnya untuk replikasi dengan instans eksternal.

replication_user_password

Kata sandi ID pengguna yang ditentukan dalam `replication_user_name`.

ssl_encryption

Opsi ini belum diterapkan. Default-nya adalah 0.

> **Note**
>
> Gunakan mysql.rds_set_binlog_source_ssl (Aurora MySQL versi 3) untuk mengaktifkan SSL untuk replikasi log biner.

Catatan penggunaan

Untuk klaster DB Aurora MySQL, Anda memanggil prosedur tersimpan ini saat terhubung ke instans primer.

Pengguna administratif harus menjalankan prosedur `mysql.rds_set_external_source_with_auto_position`. Pengguna administratif menjalankan prosedur ini pada instans primer dari klaster DB Aurora MySQL yang bertindak sebagai target replikasi. Ini dapat menjadi target replikasi dari instans DB MySQL eksternal atau klaster DB Aurora MySQL.

Prosedur ini didukung untuk Aurora MySQL versi 3. Prosedur ini tidak mengonfigurasi replikasi tertunda, karena Aurora MySQL tidak mendukung replikasi tertunda.

Sebelum Anda menjalankan `mysql.rds_set_external_source_with_auto_position`, konfigurasikan instans DB MySQL eksternal menjadi sumber replikasi. Agar terhubung ke instans MySQL eksternal, tentukan nilai untuk `replication_user_name` dan `replication_user_password`. Nilai-nilai ini harus menunjukkan pengguna replikasi yang memiliki izin `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada instans MySQL eksternal.

Untuk mengonfigurasi instans MySQL eksternal sebagai sumber replikasi

1. Dengan menggunakan klien MySQL pilihan Anda, hubungkan ke instans MySQL eksternal dan buat akun pengguna yang akan digunakan untuk replikasi. Berikut adalah contohnya.

   ```
   CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
   ```

2. Pada instans MySQL eksternal, berikan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE` kepada pengguna replikasi Anda. Contoh berikut memberikan hak akses `REPLICATION CLIENT` dan `REPLICATION SLAVE` pada semua basis data untuk pengguna `'repl_user'` domain Anda.

   ```
   GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'
   IDENTIFIED BY 'SomePassW0rd'
   ```

Saat Anda memanggil `mysql.rds_set_external_source_with_auto_position`, Amazon RDS mencatat informasi tertentu. Informasi ini adalah waktu, pengguna, dan tindakan `"set master"` di tabel `mysql.rds_history` dan `mysql.rds_replication_status`.

Untuk melewati transaksi berbasis GTID tertentu yang diketahui menyebabkan masalah, Anda dapat menggunakan prosedur tersimpan mysql.rds_skip_transaction_with_gtid />. Untuk informasi selengkapnya tentang cara menggunakan replikasi berbasis GTID, lihat Menggunakan replikasi berbasis GTID untuk Amazon Aurora MySQL.

Contoh-contoh

Saat menjalankan di instans primer Aurora, contoh berikut mengonfigurasi klaster Aurora untuk bertindak sebagai replika baca dari instans MySQL yang berjalan di luar Aurora.

```
call mysql.rds_set_external_source_with_auto_position(
```

```
  'Externaldb.some.com',
  3306,
  'repl_user'@'mydomain.com',
  'SomePassW0rd');
```

mysql.rds_set_master_auto_position (Aurora MySQL versi 2)

Mengatur mode replikasi agar didasarkan pada posisi file log biner atau pengidentifikasi transaksi global (GTID).

Sintaksis

```
CALL mysql.rds_set_master_auto_position (
auto_position_mode
);
```

Parameter

auto_position_mode

Nilai yang menunjukkan apakah akan menggunakan replikasi posisi file log atau replikasi berbasis GTID:

- `0` – Gunakan metode replikasi berdasarkan posisi file log biner. Default-nya adalah `0`.
- `1` – Gunakan metode replikasi berbasis GTID.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_set_master_auto_position`.

Prosedur ini didukung untuk Aurora MySQL versi 2.

mysql.rds_set_read_only (Aurora MySQL versi 3)

Menghidupkan atau menonaktifkan `read_only` mode secara global untuk instans DB.

Sintaks

```
CALL mysql.rds_set_read_only(mode);
```

Parameter-parameter

modus

Nilai yang menunjukkan apakah `read_only` mode aktif atau tidak aktif secara global untuk instans DB:

- `0`—`OFF`. Defaultnya adalah`0`.
- `1` – `ON`

Catatan penggunaan

Prosedur yang `mysql.rds_set_read_only` disimpan hanya memodifikasi `read_only` parameter. `innodb_read_only`Parameter tidak dapat diubah pada instans DB pembaca.

Perubahan `read_only` parameter tidak bertahan saat reboot. Untuk membuat perubahan permanen`read_only`, Anda harus menggunakan parameter cluster `read_only` DB.

Prosedur ini didukung untuk Aurora MySQL versi 3.06 dan lebih tinggi.

mysql.rds_set_session_binlog_format (Aurora MySQL versi 2)

Mengatur format log biner untuk sesi saat ini.

Sintaksis

```
CALL mysql.rds_set_session_binlog_format(format);
```

Parameter

format

Nilai yang menunjukkan format log biner untuk sesi saat ini:

- `STATEMENT` – Sumber replikasi menulis peristiwa ke log biner berdasarkan pernyataan SQL.
- `ROW` – Sumber replikasi menulis peristiwa ke log biner yang menunjukkan perubahan pada baris tabel individual.
- `MIXED` – Pencatatan log umumnya didasarkan pada pernyataan SQL, tetapi beralih ke baris dalam kondisi tertentu. Untuk informasi selengkapnya, lihat Mixed Binary Logging Format dalam dokumentasi MySQL.

Catatan penggunaan

Untuk klaster DB Aurora MySQL, Anda memanggil prosedur tersimpan ini saat terhubung ke instans primer.

Untuk menggunakan prosedur tersimpan ini, Anda harus memiliki pencatatan log biner yang dikonfigurasi untuk sesi saat ini.

Untuk Aurora, prosedur ini didukung untuk Aurora MySQL versi 2.12 dan versi yang lebih tinggi yang kompatibel dengan MySQL 5.7.

mysql.rds_set_source_auto_position (Aurora MySQL versi 3)

Mengatur mode replikasi agar didasarkan pada posisi file log biner atau pengidentifikasi transaksi global (GTID).

Sintaksis

```
CALL mysql.rds_set_source_auto_position (auto_position_mode);
```

Parameter

auto_position_mode

Nilai yang menunjukkan apakah akan menggunakan replikasi posisi file log atau replikasi berbasis GTID:

- `0` – Gunakan metode replikasi berdasarkan posisi file log biner. Default-nya adalah `0`.
- `1` – Gunakan metode replikasi berbasis GTID.

Catatan penggunaan

Untuk klaster DB Aurora MySQL, Anda memanggil prosedur tersimpan ini saat terhubung ke instans primer.

Pengguna administratif harus menjalankan prosedur `mysql.rds_set_source_auto_position`.

mysql.rds_skip_transaction_with_gtid (Aurora MySQL versi 2 dan 3)

Melewati replikasi transaksi dengan pengenal transaksi global (GTID) yang ditentukan pada instans primer Aurora.

Anda dapat menggunakan prosedur ini untuk pemulihan bencana ketika transaksi GTID tertentu diketahui menyebabkan masalah. Gunakan prosedur tersimpan ini untuk melewati transaksi bermasalah. Contoh transaksi bermasalah mencakup transaksi yang menonaktifkan replikasi, menghapus data penting, atau menyebabkan instans DB menjadi tidak tersedia.

Sintaksis

```
CALL mysql.rds_skip_transaction_with_gtid (
gtid_to_skip
);
```

Parameter

gtid_to_skip

GTID dari transaksi replikasi yang akan dilewati.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_skip_transaction_with_gtid`.

Prosedur ini didukung untuk Aurora MySQL versi 2 dan 3.

Contoh-contoh

Contoh berikut melewati replikasi transaksi dengan GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`.

```
CALL mysql.rds_skip_transaction_with_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_skip_repl_error

Melewati dan menghapus kesalahan replikasi pada replika baca DB MySQL.

Sintaksis

```
CALL mysql.rds_skip_repl_error;
```

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_skip_repl_error` pada replika baca. Untuk informasi selengkapnya tentang prosedur ini, lihat Melewati kesalahan replikasi saat ini.

Untuk menentukan apakah ada kesalahan, jalankan perintah `SHOW REPLICA STATUS\G` MySQL. Jika kesalahan replikasi tidak parah, Anda dapat menjalankan `mysql.rds_skip_repl_error` untuk melewati kesalahan tersebut. Jika ada beberapa kesalahan, `mysql.rds_skip_repl_error` akan menghapus kesalahan pertama, lalu memberi peringatan bahwa ada kesalahan lain. Anda kemudian dapat menggunakan `SHOW REPLICA STATUS\G` untuk menentukan tindakan yang benar untuk kesalahan berikutnya. Untuk informasi tentang nilai yang ditampilkan, lihat SHOW REPLICA STATUS statement dalam dokumentasi MySQL.

> **Note**
>
> Versi MySQL sebelumnya menggunakan `SHOW SLAVE STATUS`, bukan `SHOW REPLICA STATUS`. Jika Anda menggunakan versi MySQL sebelum 8.0.23, gunakan `SHOW SLAVE STATUS`.

Untuk informasi selengkapnya tentang cara mengatasi kesalahan replikasi dengan Aurora MySQL, lihat Mendiagnosis dan menyelesaikan kegagalan replikasi baca MySQL.

Kesalahan replikasi terhenti

Ketika memanggil prosedur `mysql.rds_skip_repl_error`, Anda mungkin menerima pesan kesalahan yang menyatakan bahwa replika tidak berfungsi atau dinonaktifkan.

Pesan kesalahan ini muncul jika Anda menjalankan prosedur pada instans primer, bukan replika baca. Anda harus menjalankan prosedur ini pada replika baca agar prosedur berfungsi.

Pesan kesalahan ini mungkin juga muncul jika Anda menjalankan prosedur pada replika baca, tetapi replikasi tidak berhasil dimulai ulang.

Jika Anda perlu melewati sejumlah besar kesalahan, lag replikasi dapat meningkat hingga melampaui periode retensi default untuk file log biner (binlog). Dalam kasus ini, Anda mungkin mengalami kesalahan fatal karena file binlog dihapus sebelum diputar ulang di replika baca. Penghapusan ini menyebabkan replikasi berhenti, dan Anda tidak dapat lagi memanggil perintah `mysql.rds_skip_repl_error` untuk melewati kesalahan replikasi.

Anda dapat memitigasi masalah ini dengan meningkatkan jumlah jam retensi file binlog tersebut pada instans basis data sumber Anda. Setelah meningkatkan waktu retensi binlog, Anda dapat memulai ulang replikasi dan memanggil perintah `mysql.rds_skip_repl_error` sesuai kebutuhan.

Untuk mengatur waktu retensi binlog, gunakan prosedur mysql.rds_set_configuration dan tentukan parameter konfigurasi `'binlog retention hours'` bersama dengan jumlah jam untuk mempertahankan file binlog di klaster DB. Contoh berikut menetapkan periode penyimpanan file binlog menjadi 48 jam.

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

mysql.rds_start_replication

Memulai replikasi dari klaster DB Aurora MySQL.

> **Note**
>
> Anda dapat menggunakan prosedur tersimpan mysql.rds_start_replication_until (Aurora MySQL versi 3) atau mysql.rds_start_replication_until_gtid (Aurora MySQL versi 3) untuk memulai replikasi dari instans DB Aurora MySQL dan menghentikan replikasi di lokasi file log biner yang telah ditentukan.

Sintaksis

```
CALL mysql.rds_start_replication;
```

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_start_replication`.

Untuk mengimpor data dari instans MySQL yang berjalan di luar Amazon RDS, panggil `mysql.rds_start_replication` pada replika baca untuk memulai proses replikasi setelah Anda memanggil `mysql.rds_set_external_master`atau `mysql.rds_set_external_source` membangun konfigurasi replikasi. Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner).

Untuk mengekspor data ke instans MySQL yang berjalan di luar Amazon RDS, panggil `mysql.rds_start_replication` dan `mysql.rds_stop_replication` pada replika baca

untuk mengontrol beberapa tindakan replikasi, seperti membersihkan log biner. Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner).

Anda juga dapat memanggil `mysql.rds_start_replication` pada replika baca untuk memulai kembali proses replikasi apa pun yang sebelumnya Anda hentikan dengan memanggil `mysql.rds_stop_replication`. Untuk informasi selengkapnya, lihat Kesalahan replikasi terhenti.

mysql.rds_start_replication_until (Aurora MySQL versi 3)

Memulai replikasi dari klaster DB Aurora MySQL dan menghentikan replikasi di lokasi file log biner yang telah ditentukan.

Sintaksis

```
CALL mysql.rds_start_replication_until (
replication_log_file
  , replication_stop_point
);
```

Parameter

replication_log_file

Nama log biner pada instans basis data sumber yang berisi informasi replikasi.

replication_stop_point

Lokasi di log biner `replication_log_file` tempat replikasi akan berhenti.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_start_replication_until`.

Prosedur ini didukung untuk Aurora MySQL versi 3.04 dan yang lebih tinggi.

Prosedur `mysql.rds_start_replication_until` tersimpan tidak didukung untuk replikasi terkelola, yang mencakup hal-hal berikut:

- Mereplikasi klaster DB Amazon Aurora MySQL di seluruh Wilayah AWS
- Memigrasikan data dari instans DB RDS for MySQL ke klaster DB Amazon Aurora MySQL menggunakan replika baca Aurora

Nama file yang ditentukan untuk parameter `replication_log_file` harus cocok dengan nama file binlog instans basis data sumber.

Jika parameter `replication_stop_point` menentukan lokasi perhentian di masa lalu, replikasi akan segera dihentikan.

Contoh-contoh

Contoh berikut memulai replikasi dan mereplikasi perubahan hingga mencapai lokasi `120` di file log biner `mysql-bin-changelog.000777`.

```
call mysql.rds_start_replication_until(
  'mysql-bin-changelog.000777',
  120);
```

mysql.rds_start_replication_until_gtid (Aurora MySQL versi 3)

Memulai replikasi dari klaster DB Aurora MySQL dan menghentikan replikasi segera setelah pengidentifikasi transaksi global (GTID) yang ditentukan.

Sintaksis

```
CALL mysql.rds_start_replication_until_gtid(gtid);
```

Parameter

`gtid`

GTID setelah replikasi dihentikan.

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_start_replication_until_gtid`.

Prosedur ini didukung untuk Aurora MySQL versi 3.04 dan yang lebih tinggi.

Prosedur `mysql.rds_start_replication_until_gtid` tersimpan tidak didukung untuk replikasi terkelola, yang mencakup hal-hal berikut:

- Mereplikasi klaster DB Amazon Aurora MySQL di seluruh Wilayah AWS

- Memigrasikan data dari instans DB RDS for MySQL ke klaster DB Amazon Aurora MySQL menggunakan replika baca Aurora

Saat parameter `gtid` menentukan transaksi yang telah dijalankan oleh replika, replikasi akan segera dihentikan.

Contoh-contoh

Contoh berikut memulai replikasi dan mereplikasi perubahan hingga mencapai GTID `3E11FA47-71CA-11E1-9E33-C80AA9429562:23`.

```
call mysql.rds_start_replication_until_gtid('3E11FA47-71CA-11E1-9E33-C80AA9429562:23');
```

mysql.rds_stop_replication

Menghentikan replikasi dari instans DB MySQL.

Sintaksis

```
CALL mysql.rds_stop_replication;
```

Catatan penggunaan

Pengguna utama harus menjalankan prosedur `mysql.rds_stop_replication`.

Jika Anda mengonfigurasi replikasi untuk mengimpor data dari instans MySQL yang berjalan di luar Amazon RDS, Anda memanggil `mysql.rds_stop_replication` pada replika baca untuk menghentikan proses replikasi setelah impor selesai. Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner).

Jika Anda mengonfigurasi replikasi untuk mengekspor data ke instans MySQL yang berjalan di luar Amazon RDS, Anda memanggil `mysql.rds_start_replication` dan `mysql.rds_stop_replication` pada replika baca untuk mengontrol beberapa tindakan replikasi, seperti membersihkan log biner. Untuk informasi selengkapnya, lihat Replikasi antara Aurora dan MySQL atau antara Aurora dan klaster DB Aurora lainnya (replikasi log biner).

Prosedur `mysql.rds_stop_replication` tersimpan tidak didukung untuk replikasi terkelola, yang mencakup hal-hal berikut:

- Mereplikasi klaster DB Amazon Aurora MySQL di seluruh Wilayah AWS

- Memigrasikan data dari instans DB RDS for MySQL ke klaster DB Amazon Aurora MySQL menggunakan replika baca Aurora

Tabel information_schema khusus Aurora MySQL

Aurora MySQL memiliki tabel `information_schema` tertentu yang khusus untuk Aurora.

information_schema.aurora_global_db_instance_status

Tabel `information_schema.aurora_global_db_instance_status` berisi informasi tentang status semua instans DB dalam klaster DB primer dan sekunder di basis data global. Tabel berikut menunjukkan kolom yang dapat Anda gunakan. Kolom yang tersisa hanya ditujukan untuk penggunaan internal Aurora.

> **Note**
>
> Tabel skema informasi ini hanya tersedia dengan basis data global Aurora MySQL versi 3.04.0 dan lebih tinggi.

Kolom	Jenis data	Deskripsi
SERVER_ID	varchar(100)	Pengidentifikasi instans DB.
SESSION_ID	varchar(100)	Pengidentifikasi unik untuk sesi saat ini. Nilai `MASTER_SESSION_ID` mengidentifikasi instans DB Penulis (primer).
AWS_REGION	varchar(100)	Wilayah AWS tempat instans basis data global ini berjalan. Untuk daftar Wilayah, lihat Ketersediaan wilayah.
DURABLE_LSN	bigint unsigned	Nomor urutan log (LSN) yang dijadikan durabel di penyimpanan. Nomor urutan log (LSN) adalah

Kolom	Jenis data	Deskripsi
		nomor urutan unik yang mengidentifikasi catatan di log transaksi basis data. LSN diurutkan sedemikian rupa sehingga LSN yang lebih besar merepresentasikan transaksi berikutnya.
HIGHEST_LSN_RCVD	bigint unsigned	LSN tertinggi yang diterima oleh instans DB dari instans DB penulis.
OLDEST_READ_VIEW_TRX_ID	bigint unsigned	ID transaksi terlama yang dapat dibuang oleh instans DB penulis.
OLDEST_READ_VIEW_LSN	bigint unsigned	LSN terlama yang digunakan oleh instans DB untuk membaca dari penyimpanan.
VISIBILITY_LAG_IN_MSEC	float(10,0) unsigned	Untuk pembaca di klaster DB primer, seberapa jauh instans DB ini tertinggal dari instans DB penulis dalam milidetik. Untuk pembaca di DB klaster sekunder, seberapa jauh DB instans ini tertinggal dari volume sekunder dalam milidetik.

information_schema.aurora_global_db_status

Tabel `information_schema.aurora_global_db_status` berisi informasi tentang berbagai aspek lag basis data global Aurora, khususnya, lag penyimpanan Aurora yang mendasarinya (disebut lag durabilitas) dan lag di antara sasaran titik pemulihan (RPO). Tabel berikut menunjukkan kolom yang dapat Anda gunakan. Kolom yang tersisa hanya ditujukan untuk penggunaan internal Aurora.

> **Note**
>
> Tabel skema informasi ini hanya tersedia dengan basis data global Aurora MySQL versi 3.04.0 dan lebih tinggi.

Kolom	Jenis data	Deskripsi
AWS_REGION	varchar(100)	Wilayah AWS tempat instans basis data global ini berjalan. Untuk daftar Wilayah, lihat Ketersediaan wilayah.
HIGHEST_LSN_WRITTEN	bigint unsigned	Nomor urutan log (LSN) tertinggi yang saat ini ada di klaster DB ini. Nomor urutan log (LSN) adalah nomor urutan unik yang mengidentifikasi catatan di log transaksi basis data. LSN diurutkan sedemikian rupa sehingga LSN yang lebih besar merepresentasikan transaksi berikutnya.
DURABILITY_LAG_IN_MILLISECONDS	float(10,0) unsigned	Perbedaan nilai stempel waktu antara `HIGHEST_LSN_WRITTEN` di klaster DB sekunder dan `HIGHEST_LSN_WRITTEN` di klaster DB primer. Nilai ini selalu 0 pada klaster DB primer di basis data global Aurora.
RPO_LAG_IN_MILLISECONDS	float(10,0) unsigned	Lag sasaran titik pemulihan (RPO). Lag RPO adalah waktu yang dibutuhkan COMMIT

Kolom	Jenis data	Deskripsi
		transaksi pengguna terbaru untuk disimpan di klaster DB sekunder setelah disimpan di klaster DB primer dari basis data global Aurora. Nilai ini selalu 0 pada klaster DB primer di basis data global Aurora. Secara sederhana, metrik ini menghitung sasaran titik pemulihan untuk setiap klaster DB Aurora MySQL di basis data global Aurora, yaitu, berapa banyak data yang mungkin hilang jika ada pemadaman. Seperti halnya lag, RPO diukur dalam waktu.
LAST_LAG_CALCULATION_TIMESTAMP	datetime	Stempel waktu yang menentukan kapan nilai terakhir dihitung untuk `DURABILITY_LAG_IN_MILLISECONDS` dan `RPO_LAG_IN_MILLISECONDS`. Nilai waktu seperti `1970-01-01 00:00:00+00` menunjukkan bahwa ini adalah klaster DB primer.
OLDEST_READ_VIEW_TRX_ID	bigint unsigned	ID transaksi terlama yang dapat dibuang oleh instans DB penulis.

information_schema.replica_host_status

Tabel `information_schema.replica_host_status` berisi informasi replikasi. Kolom yang dapat Anda gunakan ditunjukkan pada tabel berikut. Kolom yang tersisa hanya ditujukan untuk penggunaan internal Aurora.

Kolom	Jenis data	Deskripsi
CPU	double	Persentase penggunaan CPU dari host replika.
IS_CURRENT	tinyint	Apakah replika adalah yang terbaru atau tidak.
LAST_UPDATE_TIMESTAMP	datetime(6)	Waktu pembaruan terakhir terjadi. Digunakan untuk menentukan apakah sebuah catatan sudah usang.
REPLICA_LAG_IN_MILLISECONDS	double	Lag replika dalam milidetik.
SERVER_ID	varchar(100)	ID server basis data.
SESSION_ID	varchar(100)	ID sesi basis data. Digunakan untuk menentukan apakah instans DB adalah instans penulis atau pembaca.

> **Note**
>
> Ketika instans replika tertinggal, informasi yang dikueri dari tabel `information_schema.replica_host_status` milik instans replika tersebut mungkin sudah usang. Dalam situasi ini, kami menyarankan Anda mengkueri dari instans penulis sebagai gantinya.
>
> Meskipun tabel `mysql.ro_replica_status` memiliki informasi serupa, kami tidak menyarankan Anda untuk menggunakannya.

information_schema.aurora_forwarding_processlist

Tabel `information_schema.aurora_forwarding_processlist` berisi informasi tentang proses terkait dalam penerusan penulisan.

Konten tabel ini hanya terlihat pada instans DB penulis untuk klaster DB dengan penerusan penulisan global atau penerusan penulisan dalam klaster diaktifkan. Set hasil kosong dihasilkan pada instans DB pembaca.

Bidang	Jenis data	Deskripsi
ID	bigint	Pengidentifikasi koneksi pada instans DB penulis. Pengidentifikasi ini adalah nilai yang sama yang ditampilkan di kolom `Id` untuk pernyataan `SHOW PROCESSLIST` dan dihasilkan oleh fungsi `CONNECTION_ID()` di dalam thread.
USER	varchar(32)	Pengguna MySQL yang mengeluarkan pernyataan.
HOST	varchar(255)	Klien MySQL yang mengeluarkan pernyataan. Untuk pernyataan yang diteruskan, bidang ini menunjukkan alamat host klien aplikasi yang membuat koneksi pada instans DB pembaca yang melakukan penerusan.
DB	varchar(64)	Basis data default untuk thread.
COMMAND	varchar(16)	Jenis perintah yang dijalankan thread atas nama klien, atau `Sleep` jika sesinya idle. Untuk deskripsi perintah thread, lihat Thread Command Values dalam dokumentasi MySQL.
TIME	int	Waktu dalam hitungan detik saat thread berada dalam status saat ini.
STATE	varchar(64)	Tindakan, peristiwa, atau status yang menunjukkan apa yang dilakukan thread. Untuk deskripsi nilai status, lihat General Thread States dalam dokumentasi MySQL.

Bidang	Jenis data	Deskripsi
INFO	longtext	Pernyataan bahwa thread sedang mengeksekusi pernyataan, atau `NULL` jika tidak sedang mengeksekusi pernyataan. Pernyataan ini mungkin adalah pernyataan yang dikirim ke server, atau mungkin adalah pernyataan terdalam jika mengeksekusi pernyataan lain.
IS_FORWARDED	bigint	Menunjukkan apakah thread diteruskan dari instans DB pembaca.
REPLICA_SESSION_ID	bigint	Pengidentifikasi koneksi pada Replika Aurora. Pengidentifikasi ini adalah nilai yang sama yang ditampilkan di kolom `Id` untuk pernyataan `SHOW PROCESSLIST` pada instans DB Aurora pembaca yang melakukan penerusan.
REPLICA_INSTANCE_IDENTIFIER	varchar(64)	Pengidentifikasi instans DB dari thread penerusan.
REPLICA_CLUSTER_NAME	varchar(64)	Pengidentifikasi klaster DB dari thread penerusan. Untuk penerusan penulisan dalam klaster, pengidentifikasi ini adalah klaster DB yang sama dengan instans DB penulis.
REPLICA_REGION	varchar(64)	Wilayah AWS tempat thread penerusan berasal. Untuk penerusan penulisan dalam klaster, Wilayah ini adalah Wilayah AWS yang sama dengan instans DB penulis.

Pembaruan mesin basis data untuk Amazon Aurora MySQL

Amazon Aurora merilis pembaruan secara berkala. Pembaruan diterapkan pada klaster Aurora DB selama periode pemeliharaan sistem. Pengaturan waktu saat pembaruan diterapkan tergantung pada wilayah dan pengaturan periode pemeliharaan untuk klaster DB, serta jenis pembaruannya.

Rilis Amazon Aurora tersedia untuk semua Wilayah AWS selama beberapa hari. Beberapa Wilayah mungkin untuk sementara menampilkan versi mesin yang belum tersedia di Wilayah lain.

Pembaruan diterapkan ke semua instans dalam klaster DB pada saat yang sama. Pembaruan memerlukan pengaktifan ulang basis data di semua instans dalam klaster DB, sehingga Anda akan mengalami waktu henti 20 hingga 30 detik. Setelah itu, Anda dapat melanjutkan menggunakan klaster DB Anda. Anda dapat melihat atau mengubah pengaturan periode pemeliharaan dari AWS Management Console.

Untuk detail tentang versi MySQL Aurora yang didukung oleh Amazon Aurora, lihat Catatan Rilis untuk Aurora MySQL.

Di bagian berikut ini, Anda dapat mempelajari cara memilih versi Aurora MySQL yang tepat untuk klaster Anda, cara menentukan versi saat Anda membuat atau meng-upgrade klaster, dan prosedur untuk meng-upgrade klaster dari satu versi ke versi lainnya dengan interupsi minimal.

Topik

- Nomor versi dan versi khusus Aurora MySQL
- Mempersiapkan untuk Amazon Aurora MySQL-kompatibel Edition versi 2 akhir dukungan standar
- Bersiap untuk akhir masa pakai Amazon Aurora MySQL Compatible Edition versi 1
- Meng-upgrade klaster DB Amazon Aurora MySQL
- Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 3
- Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 2
- Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 1
- Pembaruan mesin basis data untuk klaster Aurora MySQL Serverless
- Bug MySQL diperbaiki oleh pembaruan mesin basis data Aurora MySQL
- Kerentanan keamanan yang diperbaiki di Amazon Aurora MySQL

Nomor versi dan versi khusus Aurora MySQL

Meskipun Aurora MySQL-Compatible Edition kompatibel dengan mesin basis data MySQL, Aurora MySQL menyertakan fitur dan perbaikan bug yang khusus untuk versi Aurora MySQL tertentu. Developer aplikasi dapat memeriksa versi Aurora MySQL di aplikasi mereka dengan menggunakan SQL. Administrator basis data dapat memeriksa dan menentukan versi Aurora MySQL saat membuat atau meng-upgrade klaster DB dan instans DB Aurora MySQL.

Topik

- Memeriksa atau menentukan versi mesin Aurora MySQL melalui AWS
- Memeriksa versi Aurora MySQL menggunakan SQL
- Rilis dukungan jangka panjang (LTS) Aurora MySQL
- Rilis beta Aurora MySQL

Memeriksa atau menentukan versi mesin Aurora MySQL melalui AWS

Saat Anda melakukan tugas administratif menggunakan AWS Management Console, AWS CLI, atau API RDS, Anda menentukan versi Aurora MySQL dalam format alfanumerik deskriptif.

Mulai dari Aurora MySQL versi 2, versi mesin Aurora memiliki sintaksis berikut.

```
mysql-major-version.mysql_aurora.aurora-mysql-version
```

Bagian *mysql-major-version-* adalah `5.7` atau `8.0`. Nilai ini merepresentasikan versi protokol klien dan tingkat umum dukungan fitur MySQL untuk versi Aurora MySQL yang sesuai.

aurora-mysql-version adalah nilai bertitik dengan tiga bagian: Aurora MySQL versi mayor, Aurora MySQL versi kecil, dan tingkat patch. Versi mayor adalah `2` atau `3`. Nilai tersebut merepresentasikan Aurora MySQL yang masing-masing kompatibel dengan MySQL 5.7 atau 8.0. Versi minor merepresentasikan rilis fitur dalam seri 2.x atau 3.x. Tingkat patch dimulai dari `0` untuk setiap versi minor, dan merepresentasikan serangkaian perbaikan bug berikutnya yang berlaku untuk versi minor. Terkadang, fitur baru disertakan ke dalam versi minor, tetapi tidak segera terlihat. Dalam hal ini, fitur tersebut mendapatkan penyesuaian dan diterbitkan di tingkat patch berikutnya.

Semua versi mesin 2.x Aurora MySQL memiliki "wire compatibility" dengan Community MySQL 5.7.12. Semua versi mesin 3.x Aurora MySQL memiliki "wire compatibility" dengan MySQL 8.0.23

dan seterusnya. Anda dapat merujuk ke catatan rilis versi 3.x tertentu untuk menemukan versi sesuai yang kompatibel dengan MySQL.

Misalnya, versi mesin untuk Aurora MySQL 3.02.0 dan 2.11.2 adalah sebagai berikut.

```
8.0.mysql_aurora.3.02.0
5.7.mysql_aurora.2.11.2
```

> **Note**
>
> Tidak ada one-to-one korespondensi antara versi MySQL komunitas dan versi Aurora MySQL 2.x. Untuk Aurora MySQL versi 3, ada pemetaan yang lebih langsung. Untuk memeriksa perbaikan bug dan fitur baru yang ada dalam rilis Aurora MySQL tertentu, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 3 dan Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 2 dalam Catatan Rilis untuk Aurora MySQL. Untuk daftar kronologis fitur dan rilis baru, lihat Riwayat dokumen. Untuk memeriksa versi minimum yang diperlukan untuk perbaikan terkait sistem, lihat Kerentanan keamanan yang diperbaiki di Aurora MySQL dalam Catatan Rilis untuk Aurora MySQL.

Anda menentukan versi mesin Aurora MySQL di beberapa perintah AWS CLI dan operasi API RDS. Misalnya, Anda menentukan `--engine-version` opsi ketika Anda menjalankan AWS CLI perintah create-db-clusterdan modify-db-cluster. Anda menentukan parameter `EngineVersion` saat menjalankan operasi API RDS CreateDBCluster dan ModifyDBCluster.

Di Aurora MySQL versi 2 dan lebih tinggi, versi mesin di AWS Management Console juga mencakup versi Aurora. Upgrade klaster akan mengubah nilai yang ditampilkan. Perubahan ini membantu Anda menentukan dan memeriksa versi Aurora MySQL yang tepat, tanpa perlu terhubung ke klaster atau menjalankan perintah SQL apa pun.

> **Tip**
>
> Untuk klaster Aurora dikelola melalui AWS CloudFormation, perubahan ini di pengaturan `EngineVersion` dapat memicu tindakan oleh AWS CloudFormation. Untuk informasi tentang bagaimana AWS CloudFormation memperlakukan perubahan pada pengaturan `EngineVersion`, lihat Dokumentasi AWS CloudFormation.

Memeriksa versi Aurora MySQL menggunakan SQL

Nomor versi Aurora yang dapat Anda ambil di aplikasi Anda dengan kueri SQL memiliki format *<major version>*.*<minor version>*.*<patch version>*. Anda bisa mendapatkan nomor versi ini untuk instans DB apa pun di klaster Aurora MySQL Anda dengan mengkueri variabel sistem `AURORA_VERSION`. Untuk mendapatkan nomor versi ini, gunakan salah satu kueri berikut.

```
select aurora_version();
select @@aurora_version;
```

Kueri tersebut menghasilkan output seperti yang berikut ini.

```
mysql> select aurora_version(), @@aurora_version;
+------------------+------------------+
| aurora_version() | @@aurora_version |
+------------------+------------------+
| 2.11.1           | 2.11.1           |
+------------------+------------------+
```

Nomor versi yang dihasilkan oleh konsol, CLI, dan API RDS dengan menggunakan teknik yang dijelaskan dalam Memeriksa atau menentukan versi mesin Aurora MySQL melalui AWS biasanya lebih deskriptif.

Rilis dukungan jangka panjang (LTS) Aurora MySQL

Setiap Aurora MySQL versi baru akan tetap tersedia selama waktu tertentu untuk Anda gunakan saat membuat atau meng-upgrade klaster DB. Setelah periode ini, Anda harus meng-upgrade klaster yang menggunakan versi tersebut. Anda dapat meng-upgrade klaster secara manual sebelum periode dukungan berakhir, atau Aurora dapat meng-upgrade-nya secara otomatis untuk Anda saat versi Aurora MySQL-nya tidak lagi didukung.

Aurora menetapkan versi Aurora MySQL tertentu sebagai rilis dukungan jangka panjang (LTS). Klaster DB yang menggunakan rilis LTS dapat bertahan pada versi yang sama lebih lama dan menjalani siklus upgrade yang lebih sedikit dibandingkan klaster yang menggunakan rilis non-LTS. Aurora mendukung setiap rilis LTS selama setidaknya tiga tahun setelah rilis tersebut tersedia. Ketika klaster DB yang menggunakan rilis LTS harus di-upgrade, Aurora akan meng-upgrade-nya ke rilis LTS berikutnya. Dengan demikian, klaster ini tidak perlu di-upgrade lagi dalam jangka waktu lama.

Selama masa rilis LTS Aurora MySQL, tingkat patch baru memperkenalkan perbaikan untuk masalah penting. Tingkat patch tidak mencakup fitur baru apa pun. Anda dapat memilih untuk menerapkan

patch tersebut ke klaster DB yang menjalankan rilis LTS. Untuk perbaikan kritis tertentu, Amazon mungkin akan melakukan upgrade terkelola ke sebuah tingkat patch dalam rilis LTS yang sama. Upgrade terkelola tersebut dilakukan secara otomatis dalam periode pemeliharaan klaster.

Kami menyarankan agar Anda meng-upgrade ke rilis terbaru, bukan menggunakan rilis LTS, untuk sebagian besar klaster Aurora MySQL Anda. Dengan menerapkan patch tersebut, Aurora dapat dimanfaatkan sebagai layanan terkelola serta memberi Anda akses ke fitur dan perbaikan bug terbaru. Rilis LTS ditujukan untuk klaster yang memiliki karakteristik berikut:

- Anda tidak sanggup mengalami waktu henti pada aplikasi Aurora MySQL Anda untuk melakukan upgrade di luar waktu-waktu terbatas yang biasanya ditujukan untuk patch kritis.
- Siklus pengujian untuk klaster dan aplikasi terkait membutuhkan waktu lama untuk setiap pembaruan ke mesin basis data Aurora MySQL.
- Versi basis data untuk klaster Aurora MySQL Anda memiliki semua fitur mesin DB dan perbaikan bug yang dibutuhkan aplikasi Anda.

Rilis LTS saat ini untuk Aurora MySQL adalah sebagai berikut:

- Aurora MySQL versi 3.04.*. Untuk detail selengkapnya tentang versi LTS, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 3 dalam Catatan Rilis untuk Aurora MySQL.

Rilis beta Aurora MySQL

Rilis beta Aurora MySQL adalah rilis awal khusus perbaikan keamanan di Wilayah AWS yang berjumlah terbatas. Perbaikan ini kemudian di-deploy secara lebih luas ke semua Wilayah dengan rilis patch berikutnya.

Penomoran untuk rilis beta mirip dengan versi minor Aurora MySQL, tetapi dengan digit keempat tambahan, misalnya 2.12.0.1 atau 3.05.0.1.

Untuk informasi selengkapnya, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 2 dan Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 3 dalam Catatan Rilis untuk Aurora MySQL.

Mempersiapkan untuk Amazon Aurora MySQL-kompatibel Edition versi 2 akhir dukungan standar

Amazon Aurora MySQL Compatible Edition versi 2 (dengan kompatibilitas MySQL 5.7) direncanakan akan mencapai akhir dukungan standar pada 31 Oktober 2024. Kami menyarankan Anda meningkatkan semua cluster yang menjalankan Aurora MySQL versi 2 ke default Aurora MySQL versi 3 (dengan kompatibilitas MySQL 8.0) atau lebih tinggi sebelum Aurora MySQL versi 2 mencapai akhir periode dukungan standarnya. Pada 31 Oktober 2024, Amazon RDS akan secara otomatis mendaftarkan database Anda ke Amazon RDS Extended Support. Jika Anda menjalankan Amazon Aurora MySQL versi 2 (dengan kompatibilitas MySQL 5.7) dalam klaster versi 1, ini tidak berlaku untuk Anda. Aurora Serverless Jika Anda ingin memutakhirkan cluster Aurora Serverless versi 1 Anda ke Aurora MySQL versi 3, lihat. Tingkatkan jalur untuk cluster Aurora Serverless v1 DB

Anda dapat menemukan end-of-support tanggal mendatang untuk versi utama Aurora di. Versi Amazon Aurora

Jika Anda memiliki cluster yang menjalankan Aurora MySQL versi 2, Anda akan menerima pemberitahuan berkala dengan informasi terbaru tentang cara melakukan peningkatan saat kami mendekati akhir tanggal dukungan standar. Kami akan memperbarui halaman ini secara berkala dengan informasi terbaru.

Akhir dari garis waktu dukungan standar

1. Sekarang hingga 31 Oktober 2024 – Anda dapat meningkatkan klaster dari Aurora MySQL versi 2 (dengan kompatibilitas MySQL 5.7) ke Aurora MySQL versi 3 (dengan kompatibilitas MySQL 8.0).
2. 31 Oktober 2024 — Pada tanggal ini, Aurora MySQL versi 2 akan mencapai akhir dukungan standar dan Amazon RDS secara otomatis mendaftarkan cluster Anda ke Amazon RDS Extended Support.

Kami akan secara otomatis mendaftarkan Anda di RDS Extended Support. Untuk informasi selengkapnya, lihat Menggunakan Dukungan Diperpanjang Amazon RDS.

Menemukan cluster yang terpengaruh oleh proses ini end-of-life

Untuk menemukan cluster yang terpengaruh oleh end-of-life proses ini, gunakan prosedur berikut.

> **Important**
>
> Pastikan untuk melakukan instruksi ini di setiap Wilayah AWS dan untuk setiap Akun AWS tempat sumber daya Anda berada.

Konsol

Untuk menemukan klaster Aurora MySQL versi 2

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Di kotak Filter berdasarkan basis data, masukkan 5.7.
4. Periksa Aurora MySQL di kolom mesin.

AWS CLI

Untuk menemukan cluster yang terpengaruh oleh end-of-life proses ini menggunakan AWS CLI, panggil describe-db-clustersperintah. Anda dapat menggunakan contoh skrip berikut.

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?(Engine==`aurora-
mysql` && contains(EngineVersion,`5.7.mysql_aurora`))].{EngineVersion:EngineVersion,
 DBClusterIdentifier:DBClusterIdentifier, EngineMode:EngineMode}' --output table
 --region us-east-1

        +-----------------------------------------------------------------+
        |                       DescribeDBClusters                        |
        +---------------------+---------------+-------------------------+
        |         DBCI        |      EM       |          EV             |
        +---------------------+---------------+-------------------------+
        |    aurora-mysql2    |  provisioned  | 5.7.mysql_aurora.2.11.3 |
        | aurora-serverlessv1 |   serverless  | 5.7.mysql_aurora.2.11.3 |
        +---------------------+---------------+-------------------------+
```

API RDS

Untuk menemukan klaster DB Aurora MySQL yang menjalankan Aurora MySQL versi 2, gunakan operasi API RDS DescribeDBClusters dengan parameter wajib berikut:

- `DescribeDBClusters`
  - Filters.Filter.N
    - Name
      - engine
    - Values.Value.N
      - ['aurora']

Dukungan Amazon RDS yang Diperluas

Anda dapat menggunakan Amazon RDS Extended Support melalui komunitas MySQL 5.7 tanpa biaya hingga akhir tanggal dukungan, 31 Oktober 2024. Pada tanggal 31 Oktober 2024, Amazon RDS secara otomatis mendaftarkan database Anda ke RDS Extended Support untuk Aurora MySQL versi 2. RDS Extended Support for Aurora adalah layanan berbayar yang menyediakan hingga 28 bulan dukungan tambahan untuk Aurora MySQL versi 2 hingga akhir RDS Extended Support pada Februari 2027. Dukungan yang Diperluas RDS hanya akan ditawarkan untuk Aurora MySQL versi minor 2.11 dan 2.12. Untuk menggunakan Amazon Aurora MySQL versi 2 melewati akhir dukungan standar, rencanakan untuk menjalankan database Anda di salah satu versi minor ini sebelum 31 Oktober 2024.

Melakukan upgrade

Peningkatan antarversi mayor memerlukan perencanaan dan pengujian yang lebih ekstensif daripada versi minor. Prosesnya bisa memakan waktu yang cukup lama. Kami ingin menjelaskan peningkatan sebagai proses tiga langkah, dengan aktivitas sebelum peningkatan, selama peningkatan, dan setelah peningkatan.

Sebelum peningkatan:

Sebelum peningkatan, sebaiknya Anda memeriksa kompatibilitas aplikasi, performa, prosedur pemeliharaan, dan pertimbangan serupa untuk klaster yang ditingkatkan, sehingga akan mengonfirmasi bahwa setelah peningkatan, aplikasi Anda akan berfungsi seperti yang diharapkan. Berikut adalah lima rekomendasi yang akan membantu memberi Anda pengalaman peningkatan yang lebih baik.

- Pertama, sangat penting untuk dipahamiCara kerja upgrade di tempat terhadap versi mayor Aurora MySQL.
- Selanjutnya, jelajahi teknik peningkatan yang tersedia saatMeng-upgrade dari Aurora MySQL 2.x ke 3.x.
- Untuk membantu Anda memutuskan waktu dan pendekatan yang tepat untuk meningkatkan, Anda dapat mempelajari perbedaan antara Aurora MySQL versi 3 dan lingkungan Anda saat ini dengan. Perencanaan peningkatan untuk Aurora MySQL versi 3
- Setelah Anda memutuskan opsi yang nyaman dan berfungsi paling baik, coba upgrade tiruan di tempat pada kloning kloning, gunakan. Merencanakan upgrade versi mayor untuk klaster Aurora MySQL Pemeriksaan awal ini dapat menjalankan dan menentukan apakah basis data Anda dapat ditingkatkan dengan sukses, dan apakah ada masalah inkompatibilitas aplikasi pasca-peningkatan serta performa, prosedur pemeliharaan, dan pertimbangan serupa.
- Tidak semua jenis atau versi klaster Aurora MySQL dapat menggunakan mekanisme peningkatan di tempat. Untuk informasi selengkapnya, lihat Jalur upgrade versi mayor Aurora MySQL.

Jika Anda memiliki pertanyaan atau masalah, AWS Support Team tersedia di forum komunitas dan Premium Support.

Melakukan peningkatan:

Anda dapat menggunakan upgrade di tempat atau teknik upgrade biru-hijau ketersediaan tinggi, atau memulihkan dari snapshot. Jumlah waktu henti yang akan dialami sistem Anda bergantung pada teknik peningkatan yang dipilih.

- Penerapan Biru/Hijau — Untuk situasi di mana prioritas utamanya adalah mengurangi waktu henti aplikasi, Anda dapat menggunakan Amazon RDS Blue/Green Deployment untuk melakukan peningkatan versi utama di klaster Amazon Aurora DB yang disediakan. Deployment blue/green membuat lingkungan staging yang menyerupai lingkungan produksi. Anda dapat membuat perubahan tertentu pada klaster DB Aurora di lingkungan green (staging) tanpa memengaruhi beban kerja produksi. Switchover biasanya memakan waktu kurang dari satu menit tanpa kehilangan data. Untuk informasi selengkapnya, lihat Gambaran Umum Deployment Blue/Green Amazon RDS untuk Aurora. Hal ini meminimalkan waktu henti, tetapi mengharuskan Anda untuk menjalankan sumber daya tambahan saat melakukan peningkatan.
- Upgrade di tempat — Anda dapat melakukan upgrade di tempat di mana Aurora secara otomatis melakukan proses precheck untuk Anda, membuat cluster offline, mencadangkan cluster Anda, melakukan upgrade, dan menempatkan cluster Anda kembali online. Upgrade versi utama di

tempat dapat dilakukan dalam beberapa klik, dan tidak melibatkan koordinasi atau kegagalan lain dengan cluster lain, tetapi melibatkan downtime. Lihat informasi yang lebih lengkap di Cara melakukan upgrade di tempat

- Pemulihan snapshot - Anda dapat meningkatkan cluster Aurora MySQL versi 2 Anda dengan memulihkan dari snapshot Aurora MySQL versi 2 ke cluster Aurora MySQL versi 3. Untuk melakukannya, Anda harus mengikuti proses untuk mengambil snapshot dan memulihkannya. Proses ini melibatkan gangguan database karena Anda memulihkan dari snapshot.

Anda dapat meninjau fitur-fitur terbaru Aurora MySQL versi 3 yang kompatibel dengan MySQL 8.0 dan merencanakanMeningkatkan ke Aurora MySQL versi 3. Untuk detail tentang peningkatan versi utama, lihatMeng-upgrade dari Aurora MySQL 2.x ke 3.x. Untuk detail tentang perbedaan fitur, lihatPerbandingan Aurora MySQL versi 2 dan Aurora MySQL versi 3.

Setelah peningkatan:

Setelah peningkatan, Anda perlu memantau sistem Anda (aplikasi dan basis data) dan membuat perubahan penyesuaian jika perlu. Mengikuti langkah-langkah pra-peningkatan dengan cermat akan meminimalkan perubahan yang diperlukan.

Di atas, Anda dapat menemukan ringkasan tingkat tinggi dari tiga langkah untuk meningkatkan. Untuk mempelajari lebih lanjut tentang metode, perencanaan, pengujian, dan pemecahan masalah upgrade versi utama Aurora MySQL, pastikan untuk membaca secara menyeluruh, termasuk. Meng-upgrade versi mayor klaster DB Amazon Aurora MySQL Pemecahan masalah untuk upgrade di tempat Aurora MySQL Juga, perhatikan bahwa beberapa jenis instance tidak didukung untuk Aurora MySQL versi 3. Untuk informasi selengkapnya, lihat Kelas instans DB Aurora.

Tingkatkan jalur untuk cluster Aurora Serverless v1 DB

Peningkatan antarversi mayor memerlukan perencanaan dan pengujian yang lebih ekstensif daripada versi minor. Prosesnya bisa memakan waktu yang cukup lama. Kami ingin menjelaskan peningkatan sebagai proses tiga langkah, dengan aktivitas sebelum peningkatan, selama peningkatan, dan setelah peningkatan.

Aurora MySQL versi 2 (dengan kompatibilitas MySQL 5.7) akan terus menerima dukungan standar untuk cluster. Aurora Serverless v1

Jika Anda ingin meningkatkan ke Amazon Aurora MySQL 3 (dengan kompatibilitas MySQL 8.0) dan terus menjalankan Aurora Serverless Anda dapat menggunakan Amazon Aurora

Serverless v2. Untuk memahami perbedaan antara Aurora Serverless v1 danAurora Serverless v2, lihatPerbandingan Aurora Serverless v2 dan Aurora Serverless v1.

Tingkatkan keAurora Serverless v2: Anda dapat memutakhirkan Aurora Serverless v1 cluster keAurora Serverless v2. Untuk informasi selengkapnya, lihat Upgrade dari klaster Aurora Serverless v1 ke Aurora Serverless v2.

Bersiap untuk akhir masa pakai Amazon Aurora MySQL Compatible Edition versi 1

Amazon Aurora MySQL Compatible Edition versi 1 (dengan kompatibilitas MySQL 5.6) direncanakan akan mencapai akhir masa pakai pada 28 Februari 2023. Amazon menyarankan agar Anda meng-upgrade semua klaster (terprovisi dan Aurora Serverless) yang menjalankan Aurora MySQL versi 1 ke Aurora MySQL versi 2 (dengan kompatibilitas MySQL 5.7) atau Aurora MySQL versi 3 (dengan kompatibilitas MySQL 8.0). Lakukan ini sebelum Aurora MySQL versi 1 mencapai akhir periode dukungannya.

Untuk klaster DB terprovisi Aurora, Anda dapat menyelesaikan upgrade dari Aurora MySQL versi 1 ke Aurora MySQL versi 2 dengan beberapa metode. Anda dapat menemukan petunjuk untuk mekanisme upgrade di tempat dalam Cara melakukan upgrade di tempat. Cara lain untuk menyelesaikan upgrade adalah dengan mengambil snapshot dari klaster Aurora MySQL versi 1 dan memulihkan snapshot ini ke klaster Aurora MySQL versi 2. Atau, Anda dapat mengikuti proses beberapa langkah yang menjalankan klaster lama dan baru secara berdampingan. Untuk detail selengkapnya tentang setiap metode, lihat Meng-upgrade versi mayor klaster DB Amazon Aurora MySQL.

Untuk klaster DB Aurora Serverless v1, Anda dapat melakukan upgrade di tempat dari Aurora MySQL versi 1 ke Aurora MySQL versi 2. Untuk detail selengkapnya tentang metode ini, lihat Memodifikasi klaster DB Aurora Serverless v1.

Untuk klaster DB terprovisi Aurora, Anda dapat menyelesaikan upgrade dari Aurora MySQL versi 1 ke Aurora MySQL versi 3 dengan menggunakan proses upgrade dua tahap:

1. Upgrade dari Aurora MySQL versi 1 ke Aurora MySQL versi 2 menggunakan metode yang dijelaskan sebelumnya.
2. Upgrade dari Aurora MySQL versi 2 ke Aurora MySQL versi 3 menggunakan metode yang sama seperti untuk meng-upgrade dari versi 1 ke versi 2. Untuk detail selengkapnya, lihat Meng-upgrade dari Aurora MySQL 2.x ke 3.x. Perhatikan Perbedaan fitur antara Aurora MySQL versi 2 dan 3.

Anda dapat menemukan tanggal akhir masa pakai yang akan datang untuk versi mayor Aurora dalam Versi Amazon Aurora. Amazon secara otomatis meng-upgrade klaster apa pun yang tidak Anda upgrade sendiri sebelum tanggal akhir masa pakai. Setelah tanggal akhir masa pakai, upgrade otomatis ke versi mayor berikutnya akan terjadi selama periode pemeliharaan terjadwal untuk klaster.

Berikut ini adalah milestone tambahan untuk meng-upgrade klaster Aurora MySQL versi 1 (terprovisi dan Aurora Serverless) yang mencapai akhir masa pakai. Untuk masing-masing, waktu mulainya adalah pukul 00.00 Waktu Universal Terkoordinasi (UTC).

1. Sekarang hingga 28 Februari 2023 – Anda dapat setiap saat memulai upgrade klaster Aurora MySQL versi 1 (dengan kompatibilitas MySQL 5.6) ke Aurora MySQL versi 2 (dengan kompatibilitas MySQL 5.7). Dari Aurora MySQL versi 2, Anda dapat melakukan upgrade lebih lanjut ke Aurora MySQL versi 3 (dengan kompatibilitas MySQL 8.0) untuk klaster DB terprovisi Aurora.
2. 16 Januari 2023 – Setelah waktu ini, Anda tidak dapat membuat klaster atau instans Aurora MySQL versi 1 baru dari AWS Management Console atau AWS Command Line Interface (AWS CLI). Anda juga tidak dapat menambahkan Wilayah sekunder baru ke sebuah basis data global Aurora. Ini mungkin memengaruhi kemampuan Anda untuk pulih dari pemadaman yang tidak direncanakan seperti yang diuraikan dalam Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan karena Anda tidak dapat menyelesaikan langkah 5 dan 6 setelah waktu ini. Anda juga tidak akan dapat membuat replika baca lintas Wilayah baru yang menjalankan Aurora MySQL versi 1. Anda masih dapat melakukan hal berikut untuk klaster Aurora MySQL versi 1 yang ada hingga 28 Februari 2023:
   - Pulihkan snapshot yang diambil dari klaster Aurora MySQL versi 1 ke versi yang sama dengan klaster snapshot asli.
   - Tambahkan replika baca (tidak berlaku untuk klaster DB Aurora Serverless).
   - Ubah konfigurasi instans.
   - Lakukan pemulihan titik waktu.
   - Buat klon dari klaster versi 1 yang ada.
   - Buat replika baca lintas Wilayah baru yang menjalankan Aurora MySQL versi 2 atau lebih tinggi.
3. 28 Februari 2023 – Setelah waktu ini, kami berencana untuk secara otomatis meng-upgrade klaster Aurora MySQL versi 1 ke versi default Aurora MySQL versi 2 dalam periode pemeliharaan terjadwal yang dilakukan setelahnya. Pemulihan snapshot DB Aurora MySQL versi 1 akan menghasilkan upgrade otomatis klaster yang dipulihkan ke versi default Aurora MySQL versi 2 pada waktu itu.

Upgrade antarversi mayor memerlukan perencanaan dan pengujian yang lebih ekstensif daripada versi minor. Prosesnya bisa memakan waktu yang cukup lama.

Untuk situasi yang prioritas utamanya adalah mengurangi waktu henti, Anda juga dapat menggunakan deployment blue/green untuk melakukan upgrade versi mayor di klaster DB Amazon Aurora terprovisi. Deployment blue/green membuat lingkungan staging yang menyerupai lingkungan produksi. Anda dapat membuat perubahan pada klaster DB Aurora di lingkungan green (staging) tanpa memengaruhi beban kerja produksi. Switchover biasanya memakan waktu kurang dari satu menit tanpa kehilangan data dan tidak memerlukan perubahan aplikasi. Untuk informasi selengkapnya, lihat Gambaran Umum Deployment Blue/Green Amazon RDS untuk Aurora.

Setelah upgrade selesai, Anda mungkin juga memiliki pekerjaan tindak lanjut yang harus dilakukan. Misalnya, Anda mungkin perlu melakukan tindak lanjut karena perbedaan kompatibilitas SQL, cara kerja fitur terkait MySQL tertentu, atau pengaturan parameter antara versi lama dan baru.

Untuk mempelajari selengkapnya tentang metode, perencanaan, pengujian, dan pemecahan masalah upgrade versi mayor Aurora MySQL, pastikan untuk membaca Meng-upgrade versi mayor klaster DB Amazon Aurora MySQL secara menyeluruh.

Menemukan klaster yang terpengaruh oleh proses akhir masa pakai ini

Untuk menemukan klaster yang terpengaruh oleh proses akhir masa pakai ini, gunakan prosedur berikut.

> **Important**
>
> Pastikan untuk menjalankan petunjuk ini di setiap Wilayah AWS dan untuk setiap Akun AWS tempat sumber daya Anda berada.

Konsol

Untuk menemukan klaster Aurora MySQL versi 1

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Di kotak Filter berdasarkan basis data, masukkan 5.6.
4. Periksa Aurora MySQL di kolom mesin.

AWS CLI

Untuk menemukan klaster yang terpengaruh oleh proses akhir masa pakai ini menggunakan AWS CLI, panggil perintah describe-db-clusters. Anda dapat menggunakan contoh skrip berikut.

Example

```
aws rds describe-db-clusters --include-share --query 'DBClusters[?Engine==`aurora`].
{EV:EngineVersion, DBCI:DBClusterIdentifier, EM:EngineMode}' --output table --region
 us-east-1

        +-----------------------------------------+
        |            DescribeDBClusters           |
        +---------------+--------------+----------+
        |     DBCI      |     EM       |    EV    |
        +---------------+--------------+----------+
        |  my-database-1|  serverless  |  5.6.10a |
        +---------------+--------------+----------+
```

API RDS

Untuk menemukan klaster DB Aurora MySQL yang menjalankan Aurora MySQL versi 1, gunakan operasi API RDS DescribeDBClusters dengan parameter wajib berikut:

- `DescribeDBClusters`
  - Filters.Filter.N
    - Name
      - engine
    - Values.Value.N
      - ['aurora']

Meng-upgrade klaster DB Amazon Aurora MySQL

Anda dapat meng-upgrade klaster DB Aurora MySQL untuk mendapatkan perbaikan bug, fitur Aurora MySQL baru, atau untuk beralih ke versi mesin basis data yang mendasari yang sama sekali baru. Bagian berikut menunjukkan caranya.

> **Note**
>
> Jenis upgrade yang Anda lakukan bergantung pada durasi waktu henti klaster yang dapat Anda toleransi, jumlah pengujian verifikasi yang berencana Anda lakukan, seberapa penting perbaikan bug tertentu atau fitur baru untuk kasus penggunaan Anda, dan apakah Anda berencana akan sering melakukan upgrade kecil atau upgrade sesekali yang melewati beberapa versi perantara. Untuk setiap upgrade, Anda dapat mengubah versi mayor, versi minor, dan tingkat patch untuk klaster Anda. Jika Anda tidak memahami perbedaan antara Aurora MySQL versi mayor, versi minor, dan tingkat patch, Anda dapat membaca informasi latar belakang dalam Nomor versi dan versi khusus Aurora MySQL.

> **Tip**
>
> Anda dapat meminimalkan waktu henti yang diperlukan untuk upgrade klaster DB dengan menggunakan deployment blue/green. Untuk informasi selengkapnya, lihat Menggunakan Deployment Blue/Green Amazon RDS untuk pembaruan basis data.

Topik

- Meningkatkan versi minor atau tingkat patch klaster DB Aurora MySQL
- Meng-upgrade versi mayor klaster DB Amazon Aurora MySQL

Meningkatkan versi minor atau tingkat patch klaster DB Aurora MySQL

Anda dapat menggunakan metode berikut untuk meningkatkan versi minor klaster DB atau untuk mem-patch klaster DB:

- Meningkatkan Aurora MySQL dengan mengubah versi mesin (untuk Aurora MySQL versi 2 dan 3)
- Mengaktifkan peningkatan otomatis di antara versi minor Aurora MySQL

Untuk informasi tentang cara patching zero-downtime dapat mengurangi gangguan selama proses peningkatan, lihat Menggunakan zero-downtime patching.

Sebelum melakukan upgrade versi minor

Kami menyarankan Anda melakukan tindakan berikut untuk mengurangi waktu henti selama upgrade versi minor:

- Pemeliharaan cluster Aurora DB harus dilakukan selama periode lalu lintas rendah. Gunakan Performance Insights untuk mengidentifikasi periode waktu ini agar dapat mengonfigurasi jendela pemeliharaan dengan benar. Untuk informasi selengkapnya tentang Performance Insights, lihat Memantau pemuatan DB dengan Performance Insights di Amazon RDS. Untuk informasi lebih lanjut tentang jendela pemeliharaan cluster DB,Menyesuaikan periode pemeliharaan klaster DB yang dinginkan.
- Gunakan AWS SDK yang mendukung backoff eksponensial dan jitter sebagai praktik terbaik. Untuk informasi lebih lanjut, lihat Exponential Backoff And Jitter.

Meningkatkan Aurora MySQL dengan mengubah versi mesin

Memutakhirkan versi minor dari cluster DB MySQL Aurora menerapkan perbaikan tambahan dan fitur baru ke cluster yang ada.

Peningkatan semacam ini berlaku untuk cluster Aurora MySQL di mana versi asli dan versi yang ditingkatkan keduanya memiliki versi utama Aurora MySQL yang sama, baik 2 atau 3. Prosesnya cepat dan mudah karena tidak memerlukan konversi apa pun untuk metadata Aurora MySQL atau penyusunan ulang data tabel Anda.

Anda melakukan upgrade semacam ini dengan memodifikasi versi mesin dari cluster DB menggunakan AWS Management Console, AWS CLI, atau RDS API. Misalnya, jika cluster Anda menjalankan Aurora MySQL 2.x, pilih versi 2.x yang lebih tinggi.

Jika Anda melakukan peningkatan minor pada basis data global Aurora, tingkatkan semua klaster sekunder sebelum Anda meningkatkan klaster primer.

> **Note**
>
> Untuk melakukan peningkatan versi minor ke Aurora MySQL versi 3.03.* atau lebih tinggi, atau versi 2.12.*, gunakan proses berikut:
>
> 1. Hapus semua Wilayah sekunder dari klaster global. Ikuti langkah-langkahnya dalam Menghapus klaster dari basis data global Amazon Aurora.

2. Peningkatan versi mesin Wilayah primer ke versi 3.03.* atau lebih tinggi, atau versi 2.12.*, sebagaimana berlaku. Ikuti langkah-langkahnya dalam To modify the engine version of a DB cluster.
3. Tambahkan Wilayah sekunder ke klaster global. Ikuti langkah-langkahnya dalam Menambahkan Wilayah AWS ke basis data global Amazon Aurora.

Untuk mengubah versi mesin klaster DB

- Dengan menggunakan konsol – Ubah properti klaster Anda. Di jendela Modifikasi klaster DB, ubah versi mesin Aurora MySQL dalam kotak Versi mesin DB. Jika Anda tidak memahami prosedur umum untuk mengubah klaster, ikuti petunjuk dalam Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.
- Dengan menggunakan AWS CLI — Panggil modify-db-cluster AWS CLI perintah, dan tentukan nama cluster DB Anda untuk `--db-cluster-identifier` opsi dan versi mesin untuk `--engine-version` opsi tersebut.

  Misalnya, untuk meningkatkan ke Aurora MySQL versi 2.12.1, atur opsi ke. `--engine-version 5.7.mysql_aurora.2.12.1` Tentukan opsi `--apply-immediately` untuk segera memperbarui versi mesin untuk klaster DB Anda.
- Dengan menggunakan API RDS – Panggil operasi ModifyDBCluster, dan tentukan nama klaster DB Anda untuk parameter `DBClusterIdentifier` dan versi mesin untuk parameter `EngineVersion`. Tetapkan parameter `ApplyImmediately` ke `true` untuk segera memperbarui versi mesin untuk klaster DB Anda.

Mengaktifkan peningkatan otomatis di antara versi minor Aurora MySQL

Untuk klaster DB Amazon Aurora MySQL, Anda dapat menentukan agar Aurora meningkatkan klaster DB ini secara otomatis ke versi minor baru. Anda melakukannya dengan menggunakan properti upgrade versi minor otomatis dari cluster DB menggunakan AWS Management Console, AWS CLI, atau RDS API.

Peningkatan otomatis terjadi selama periode pemeliharaan. Jika instans DB individu di klaster DB memiliki periode pemeliharaan yang berbeda dari periode pemeliharaan klaster, maka periode pemeliharaan klaster diutamakan.

Peningkatan versi minor otomatis tidak berlaku untuk jenis klaster Aurora MySQL berikut:

- Klaster yang merupakan bagian dari basis data global Aurora.
- Klaster yang memiliki replika lintas Wilayah.

Durasi pemadaman bervariasi tergantung pada beban kerja, ukuran klaster, jumlah data log biner, dan apakah Aurora dapat menggunakan fitur zero-downtime patching (ZDP). Aurora mengaktifkan ulang klaster basis data, sehingga Anda mungkin mengalami periode ketidaktersediaan yang singkat sebelum melanjutkan penggunaan klaster Anda. Secara khusus, jumlah data log biner akan memengaruhi waktu pemulihan. Instans DB memproses data log biner selama pemulihan. Dengan demikian, volume tinggi data log biner akan menambah waktu pemulihan.

> **Note**
>
> Aurora hanya melakukan peningkatan otomatis jika pengaturan ini diaktifkan untuk semua instans DB di klaster DB Anda. Untuk informasi tentang cara mengatur Peningkatan versi minor otomatis, dan cara kerja pengaturan saat diterapkan di tingkat klaster dan instans, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora.
> Peningkatan versi minor otomatis dilakukan ke versi minor default.

Anda dapat menggunakan perintah CLI seperti berikut ini untuk memeriksa status pengaturan `AutoMinorVersionUpgrade` untuk semua instans DB di klaster Aurora MySQL Anda.

```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

Perintah ini menghasilkan output yang serupa dengan berikut:

```
[
  {
      "DBInstanceIdentifier": "db-t2-medium-instance",
      "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
      "AutoMinorVersionUpgrade": true
  },
  {
      "DBInstanceIdentifier": "db-t2-small-original-size",
      "DBClusterIdentifier": "cluster-57-2020-06-03-6411",
      "AutoMinorVersionUpgrade": false
  },
```

```
  {
      "DBInstanceIdentifier": "instance-2020-05-01-2332",
      "DBClusterIdentifier": "cluster-57-2020-05-01-4615",
      "AutoMinorVersionUpgrade": true
  },
... output omitted ...
```

Dalam contoh ini, opsi Aktifkan peningkatan versi minor otomatis dinonaktifkan untuk klaster DB `cluster-57-2020-06-03-6411` karena opsi ini dinonaktifkan untuk salah satu instans DB di klaster tersebut.

Menggunakan zero-downtime patching

Peningkatan klaster DB Aurora MySQL kemungkinan akan menimbulkan pemadaman layanan saat basis data dinonaktifkan dan sedang ditingkatkan. Secara default, jika Anda memulai peningkatan saat basis data sibuk, Anda akan kehilangan semua koneksi dan transaksi yang sedang diproses oleh klaster DB. Jika Anda menunggu hingga basis data idle untuk melakukan peningkatan, Anda mungkin harus menunggu lama.

Fitur zero-downtime patching (ZDP) mencoba, berdasarkan upaya terbaik, untuk menjaga koneksi klien selama peningkatan Aurora MySQL. Jika ZDP berhasil diselesaikan, sesi aplikasi dipertahankan dan mesin basis data dimulai ulang saat peningkatan sedang berlangsung. Pengaktifan ulang mesin basis data dapat menyebabkan penurunan throughput yang berlangsung selama beberapa detik hingga kira-kira satu menit.

ZDP tidak berlaku untuk hal-hal berikut ini:

- Patch dan peningkatan sistem operasi (OS)
- Peningkatan versi mayor

ZDP tidak didukung untuk basis data Aurora Serverless v1 atau basis data global Aurora.

Tabel berikut menunjukkan versi Aurora MySQL dan kelas instans DB tempat ZDP tersedia.

Versi	Kelas instans db.r*	Kelas instans db.t*	Kelas instans db.x*	Kelas instans db.serverless
2.07.9 dan versi 2.07 yang lebih tinggi	Tidak	Ya	Tidak	N/A
2.10.0 dan versi 2.x yang lebih tinggi	Ya	Ya	Ya	N/A
3.01.0 dan 3.01.1	Ya	Ya	Ya	N/A
3.02.0 dan versi 3.x yang lebih tinggi	Ya	Ya	Ya	Ya

> **Note**
>
> Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Menggunakan kelas instans T untuk pengembangan dan pengujian.

Anda dapat melihat metrik atribut penting selama ZDP dalam log kesalahan MySQL. Anda juga dapat melihat informasi tentang kapan Aurora MySQL menggunakan ZDP atau memilih untuk tidak menggunakan ZDP di halaman Peristiwa di AWS Management Console.

Di Aurora MySQL versi 2.10 dan lebih tinggi serta versi 3, Aurora dapat melakukan zero-downtime patching baik replikasi log biner diaktifkan maupun tidak. Jika replikasi log biner diaktifkan, Aurora MySQL secara otomatis memutuskan koneksi ke target binlog selama operasi ZDP. Aurora MySQL secara otomatis menghubungkan kembali ke target binlog dan melanjutkan replikasi setelah pengaktifan ulang selesai.

ZDP juga beroperasi dalam kombinasi dengan peningkatan boot ulang di Aurora MySQL 2.10 dan versi lebih tinggi. Patching terhadap instans DB penulis akan secara otomatis menjalankan patching

terhadap pembaca secara bersamaan. Setelah melakukan patching, Aurora memulihkan koneksi pada instans DB penulis dan pembaca. Sebelum Aurora MySQL 2.10, ZDP hanya berlaku untuk instans DB penulis dari sebuah klaster.

ZDP mungkin tidak berhasil diselesaikan dalam kondisi berikut:

- Kueri atau transaksi berjalan lama sedang berlangsung. Jika Aurora dapat melakukan ZDP dalam kasus ini, semua transaksi terbuka dibatalkan.
- Tabel sementara atau kunci tabel digunakan, misalnya ketika pernyataan bahasa definisi data (DDL) berjalan. Jika Aurora dapat melakukan ZDP dalam kasus ini, semua transaksi terbuka dibatalkan.
- Ada perubahan parameter tertunda.

Jika tidak tersedia jangka waktu yang sesuai untuk melakukan ZDP karena satu atau beberapa dari kondisi ini, patching akan kembali ke perilaku standar.

> **Note**
>
> Untuk Aurora MySQL versi 2 yang lebih rendah dari 2.11.0 dan versi 3 yang lebih rendah dari 3.04.0, ZDP mungkin tidak berhasil selesai ketika ada koneksi Lapisan Soket Aman (SSL) atau Keamanan Lapisan Pengangkutan (TLS).

Meskipun koneksi tetap utuh setelah operasi ZDP berhasil, beberapa variabel dan fitur akan diinisialisasi ulang. Jenis informasi berikut ini tidak dipertahankan selama pengaktifan ulang yang disebabkan oleh zero-downtime patching:

- Variabel global. Aurora memulihkan variabel sesi, tetapi tidak memulihkan variabel global setelah pengaktifan ulang.
- Variabel status. Secara khusus, nilai uptime yang dilaporkan oleh status mesin direset setelah pengaktifan ulang yang menggunakan mekanisme ZDR atau ZDP.
- `LAST_INSERT_ID`.
- Status `auto_increment` dalam memori untuk tabel. Status inkremen otomatis dalam memori diinisialisasi ulang. Untuk informasi selengkapnya tentang nilai inkremen otomatis, lihat Panduan Referensi MySQL.
- Informasi diagnostik dari tabel `INFORMATION_SCHEMA` dan `PERFORMANCE_SCHEMA`. Informasi diagnostik ini juga muncul dalam output perintah seperti `SHOW PROFILE` dan `SHOW PROFILES`.

Aktivitas berikut yang berkaitan dengan pengaktifan ulang dengan nol waktu henti akan dilaporkan di halaman Peristiwa:

- Mencoba meningkatkan basis data dengan nol waktu henti.
- Mencoba meningkatkan basis data dengan nol waktu henti selesai. Peristiwa ini melaporkan berapa lama prosesnya berjalan. Peristiwa ini juga melaporkan berapa banyak koneksi yang dipertahankan selama pengaktifan ulang dan berapa banyak koneksi yang terputus. Anda dapat melihat log kesalahan basis data untuk melihat detail selengkapnya tentang apa yang terjadi selama pengaktifan ulang.

Teknik blue/green peningkatan alternatif

Dalam situasi tertentu, prioritas utama Anda adalah melakukan switchover langsung dari klaster lama ke klaster yang ditingkatkan. Dalam situasi seperti itu, Anda dapat menggunakan proses multistep yang menjalankan cluster side-by-side lama dan baru. Di sini, Anda mereplikasi data dari klaster lama ke klaster baru hingga Anda siap untuk mengambil alih klaster baru. Untuk detailnya, lihat Menggunakan Deployment Blue/Green Amazon RDS untuk pembaruan basis data.

Meng-upgrade versi mayor klaster DB Amazon Aurora MySQL

Dalam nomor versi MySQL Aurora seperti 2.12.1, 2 mewakili versi utama. Aurora MySQL versi 2 kompatibel dengan MySQL 5.7. Aurora MySQL versi 3 kompatibel dengan MySQL 8.0.

Upgrade antarversi mayor memerlukan perencanaan dan pengujian yang lebih ekstensif daripada versi minor. Prosesnya bisa memakan waktu yang cukup lama. Setelah upgrade selesai, Anda mungkin juga memiliki pekerjaan lanjutan yang harus dilakukan. Misalnya, ini mungkin terjadi karena perbedaan kompatibilitas SQL atau cara kerja fitur terkait MySQL tertentu. Atau, ini mungkin terjadi karena pengaturan parameter yang berbeda antara versi lama dan baru.

Topik

- Meng-upgrade dari Aurora MySQL 2.x ke 3.x
- Merencanakan upgrade versi mayor untuk klaster Aurora MySQL
- Pra-pemeriksaan upgrade versi mayor untuk Aurora MySQL
- Jalur upgrade versi mayor Aurora MySQL
- Cara kerja upgrade di tempat terhadap versi mayor Aurora MySQL
- Teknik upgrade blue-green alternatif

- Cara melakukan upgrade di tempat
- Bagaimana upgrade di tempat memengaruhi grup parameter untuk klaster
- Perubahan pada properti klaster di antara versi Aurora MySQL
- Upgrade besar di tempat untuk basis data global
- Backtrack pertimbangan
- Pemecahan masalah untuk upgrade di tempat Aurora MySQL
- Tutorial upgrade di tempat Aurora MySQL

Meng-upgrade dari Aurora MySQL 2.x ke 3.x

Jika Anda memiliki klaster yang kompatibel dengan MySQL 5.7 dan ingin meng-upgrade-nya ke klaster yang kompatibel dengan MySQL 8.0, Anda dapat melakukannya dengan menjalankan proses upgrade pada klaster itu sendiri. Upgrade semacam ini adalah upgrade di tempat, berbeda dengan upgrade yang Anda lakukan dengan membuat klaster baru. Teknik ini mempertahankan titik akhir dan karakteristik lain yang sama dari klaster. Upgrade ini relatif cepat karena tidak memerlukan penyalinan semua data Anda ke volume klaster baru. Stabilitas ini membantu meminimalkan perubahan konfigurasi dalam aplikasi Anda. Hal ini juga membantu mengurangi jumlah pengujian untuk klaster yang di-upgrade. Hal ini karena jumlah instans DB dan kelas instansnya semua tetap sama.

Mekanisme upgrade di tempat mengharuskan klaster DB Anda dinonaktifkan saat operasi berlangsung. Aurora melakukan penonaktifan bersih dan menyelesaikan operasi yang tertunda seperti rollback transaksi dan pembersihan undo. Untuk informasi selengkapnya, lihat Cara kerja upgrade di tempat terhadap versi mayor Aurora MySQL.

Metode upgrade di tempat ini praktis karena mudah dilakukan dan meminimalkan perubahan konfigurasi pada aplikasi terkait. Misalnya, upgrade di tempat mempertahankan titik akhir dan kumpulan instans DB untuk klaster Anda. Namun, waktu yang diperlukan untuk upgrade di tempat dapat bervariasi tergantung pada properti skema Anda dan seberapa sibuk klasternya. Jadi, tergantung pada kebutuhan untuk klaster Anda, Anda mungkin ingin memilih di antara beberapa teknik upgrade. Ini termasuk upgrade di tempat dan pemulihan snapshot, yang dijelaskan dalam Memulihkan dari snapshot klaster DB. Ada juga teknik upgrade lainnya seperti yang dijelaskan dalam Teknik upgrade blue-green alternatif.

> **Note**
>
> Jika Anda menggunakan AWS CLI atau API RDS untuk metode upgrade pemulihan snapshot, Anda harus menjalankan operasi lanjutan untuk membuat instans DB penulis di klaster DB yang dipulihkan.

Untuk informasi umum tentang Aurora MySQL versi 3 dan fitur-fiturnya yang baru, lihat Aurora MySQL versi 3 yang kompatibel dengan MySQL 8.0. Untuk detail tentang merencanakan upgrade, lihat Perencanaan peningkatan untuk Aurora MySQL versi 3 dan Meningkatkan ke Aurora MySQL versi 3.

> **Tip**
>
> Saat Anda meng-upgrade versi mayor klaster Anda dari 2.x ke 3.x, klaster asli dan yang di-upgrade akan menggunakan nilai `aurora-mysql` yang sama untuk atribut `engine`.

Merencanakan upgrade versi mayor untuk klaster Aurora MySQL

Untuk memastikan aplikasi dan prosedur administrasi Anda berjalan dengan lancar setelah meningkatkan klaster di antara versi mayor, lakukan beberapa perencanaan dan persiapan sebelumnya. Untuk melihat jenis kode manajemen yang akan diperbarui untuk skrip AWS CLI atau aplikasi berbasis API RDS, lihat Bagaimana upgrade di tempat memengaruhi grup parameter untuk klaster. Lihat juga Perubahan pada properti klaster di antara versi Aurora MySQL.

Untuk mempelajari jenis masalah yang mungkin Anda alami selama upgrade, lihat Pemecahan masalah untuk upgrade di tempat Aurora MySQL. Untuk masalah yang mungkin menyebabkan upgrade memakan waktu lama, Anda dapat menguji kondisi tersebut terlebih dahulu dan memperbaikinya.

Anda dapat memeriksa kompatibilitas aplikasi, performa, prosedur pemeliharaan, dan pertimbangan serupa untuk klaster yang di-upgrade. Untuk melakukannya, Anda dapat melakukan simulasi upgrade sebelum melakukan upgrade yang sebenarnya. Teknik ini dapat sangat berguna untuk klaster produksi. Di sini, penting untuk meminimalkan waktu henti dan menyiapkan klaster yang telah di-upgrade agar siap digunakan segera setelah upgrade selesai.

Gunakan langkah-langkah berikut:

1. Membuat klon dari klaster asli. Ikuti prosedur dalam Mengkloning volume untuk klaster DB Amazon Aurora.
2. Siapkan satu set instans DB penulis dan pembaca yang serupa seperti di klaster asli.
3. Melakukan upgrade di tempat terhadap klaster yang dikloning. Ikuti prosedur dalam Cara melakukan upgrade di tempat.

   Mulai upgrade segera setelah membuat klon. Dengan demikian, volume klaster masih identik dengan status klaster aslinya. Jika klon dalam keadaan idle sebelum Anda melakukan upgrade, Aurora akan melakukan proses pembersihan basis data di latar belakang. Dalam hal ini, upgrade klon bukanlah simulasi yang akurat untuk upgrade klaster asli.
4. Uji kompatibilitas aplikasi, performa, prosedur administrasi, dan sebagainya, menggunakan klaster yang dikloning.
5. Jika Anda mengalami masalah apa pun, sesuaikan rencana upgrade Anda untuk mengantisipasinya. Misalnya, sesuaikan kode aplikasi apa pun agar kompatibel dengan kumpulan fitur dari versi yang lebih tinggi. Perkirakan berapa lama waktu yang dibutuhkan untuk upgrade berdasarkan jumlah data di klaster Anda. Anda juga dapat memilih untuk menjadwalkan upgrade pada saat klaster tidak sibuk.
6. Setelah Anda yakin bahwa aplikasi dan beban kerja Anda berfungsi dengan baik dengan klaster uji, Anda dapat melakukan upgrade di tempat untuk klaster produksi Anda.
7. Berusahalah untuk meminimalkan total waktu henti klaster Anda selama upgrade versi mayor. Untuk melakukannya, pastikan bahwa beban kerja di klaster rendah atau nol pada saat upgrade. Secara khusus, pastikan bahwa tidak ada transaksi berjalan lama yang sedang berlangsung saat Anda memulai upgrade.

Untuk informasi khusus tentang upgrade ke Aurora MySQL versi 3, lihat Perencanaan peningkatan untuk Aurora MySQL versi 3.

Pra-pemeriksaan upgrade versi mayor untuk Aurora MySQL

MySQL 8.0 menyertakan sejumlah inkompatibilitas dengan MySQL 5.7. Inkompatibilitas ini dapat menyebabkan masalah selama upgrade dari Aurora MySQL versi 2 ke versi 3. Beberapa persiapan mungkin diperlukan di basis data Anda agar upgrade berhasil.

Saat Anda memulai upgrade dari Aurora MySQL versi 2 ke versi 3, Amazon Aurora akan menjalankan pra-pemeriksaan secara otomatis untuk mendeteksi inkompatibilitas ini.

Pra-pemeriksaan ini wajib dilakukan. Anda tidak dapat memilih untuk melewatinya. Pra-pemeriksaan menyediakan manfaat berikut:

- Hal ini memungkinkan Anda menghindari waktu henti yang tidak direncanakan selama upgrade.
- Jika ada inkompatibilitas, Amazon Aurora akan mencegah upgrade dan menyediakan log bagi Anda untuk mempelajarinya. Kemudian, Anda dapat menggunakan log ini untuk menyiapkan basis data Anda untuk upgrade ke versi 3 dengan mengurangi inkompatibilitas. Untuk informasi terperinci tentang cara mengatasi inkompatibilitas, lihat Preparing your installation for upgrade dalam dokumentasi MySQL dan Upgrading to MySQL 8.0? Inilah yang perlu Anda ketahui... di Blog MySQL Server.

  Untuk informasi tentang upgrade ke MySQL 8.0, lihat Upgrading to MySQL 8.0 dalam dokumentasi MySQL.

Sebagian pra-pemeriksaan disertakan dengan MySQL dan sebagian lainnya dibuat secara khusus oleh tim Aurora. Untuk informasi tentang pra-pemeriksaan yang disediakan oleh MySQL, lihat Utilitas pemeriksaan upgrade.

Pra-pemeriksaan berjalan sebelum instans DB dihentikan untuk upgrade, sehingga instans tersebut tidak akan menyebabkan waktu henti ketika berjalan. Jika pra-pemeriksaan menemukan inkompatibilitas, Aurora secara otomatis membatalkan upgrade sebelum instans DB dihentikan. Aurora juga menghasilkan peristiwa untuk inkompatibilitas. Untuk informasi selengkapnya tentang peristiwa Amazon Aurora, lihat Bekerja dengan pemberitahuan peristiwa Amazon RDS.

Aurora mencatat informasi terperinci tentang setiap inkompatibilitas dalam file log `PrePatchCompatibility.log`. Dalam kebanyakan kasus, entri log berisi tautan ke dokumentasi MySQL untuk mengoreksi inkompatibilitas. Untuk informasi selengkapnya tentang format file log, lihat Melihat dan mencantumkan file log basis data.

Karena sifatnya, pra-pemeriksaan akan menganalisis objek di basis data Anda. Analisis ini mengakibatkan konsumsi sumber daya dan menambah waktu penyelesaian upgrade.

Pra-pemeriksaan upgrade MySQL komunitas

Berikut ini adalah daftar umum inkompatibilitas antara MySQL 5.7 dan 8.0:

- Klaster DB Anda yang kompatibel dengan MySQL 5.7 tidak boleh menggunakan fitur yang tidak didukung di MySQL 8.0.

  Untuk informasi selengkapnya, lihat Features removed in MySQL 8.0 dalam dokumentasi MySQL.

- Tidak boleh ada pelanggaran terkait kata kunci atau kata yang digunakan sistem. Mungkin ada beberapa kata kunci yang digunakan sistem di MySQL 8.0 yang sebelumnya tidak digunakan sistem.

  Untuk informasi selengkapnya, lihat Keywords and reserved words dalam dokumentasi MySQL.
- Untuk dukungan Unicode yang lebih baik, pertimbangkan untuk mengonversi objek yang menggunakan set karakter `utf8mb3` untuk menggunakan set karakter `utf8mb4`. Set karakter `utf8mb3` sudah tidak digunakan lagi. Selain itu, pertimbangkan untuk menggunakan `utf8mb4` untuk referensi set karakter, bukan `utf8`, karena saat ini `utf8` adalah alias untuk set karakter `utf8mb3`.

  Untuk informasi selengkapnya, lihat The utf8mb3 character set (3-byte UTF-8 unicode encoding) dalam dokumentasi MySQL.
- Tidak boleh ada tabel InnoDB dengan format baris non-default.
- Tidak boleh ada atribut jenis panjang `ZEROFILL` atau `display`.
- Tidak boleh ada tabel partisi yang menggunakan mesin penyimpanan yang tidak memiliki dukungan partisi native.
- Tidak boleh ada tabel di basis data sistem `mysql` MySQL 5.7 yang memiliki nama yang sama dengan tabel yang digunakan oleh kamus data MySQL 8.0.
- Tidak boleh ada tabel yang menggunakan jenis atau fungsi data yang usang.
- Tidak boleh ada nama pembatasan kunci asing yang lebih panjang dari 64 karakter.
- Tidak boleh ada mode SQL usang yang ditentukan dalam pengaturan variabel sistem `sql_mode`.
- Tidak boleh ada tabel atau prosedur tersimpan dengan elemen kolom `ENUM` atau `SET` individual dengan panjang melebihi 255 karakter.
- Tidak boleh ada partisi tabel yang berada dalam ruang tabel InnoDB bersama.
- Tidak boleh ada referensi sirkular di jalur file data ruang tabel.
- Tidak boleh ada kueri dan definisi program tersimpan yang menggunakan pengkualifikasi `ASC` atau `DESC` untuk klausa `GROUP BY`.
- Tidak boleh ada variabel sistem yang dihapus, dan variabel sistem harus menggunakan nilai default baru untuk MySQL 8.0.
- Tidak boleh ada nilai nol (`0`) untuk date, datetime, atau timestamp.
- Tidak boleh ada inkonsistensi skema yang dihasilkan dari penghapusan atau kerusakan file.
- Tidak boleh ada nama tabel yang berisi string karakter `FTS`.
- Tidak boleh ada tabel InnoDB yang dimiliki oleh mesin yang berbeda.

- Tidak boleh ada nama tabel atau skema yang tidak valid untuk MySQL 5.7.

Untuk informasi tentang upgrade ke MySQL 8.0, lihat Upgrading to MySQL 8.0 dalam dokumentasi MySQL.

Pra-pemeriksaan upgrade Aurora MySQL

Aurora MySQL memiliki persyaratan spesifiknya sendiri saat meng-upgrade dari versi 2 ke versi 3:

- Tidak boleh ada sintaks SQL yang sudah berhenti digunakan, seperti `SQL_CACHE`, `SQL_NO_CACHE`, dan `QUERY_CACHE`, dalam tampilan, rutinitas, pemicu, dan peristiwa.
- Tidak boleh ada kolom `FTS_DOC_ID` yang ada di tabel apa pun tanpa indeks `FTS`.
- Tidak boleh ada ketidakcocokan definisi kolom antara kamus data InnoDB dan definisi tabel yang sebenarnya.
- Semua nama basis data dan tabel harus huruf kecil ketika parameter `lower_case_table_names` diatur ke `1`.
- Peristiwa dan pemicu tidak boleh memiliki pendefinisi yang hilang atau kosong atau konteks pembuatan yang tidak valid.
- Semua nama pemicu dalam basis data harus unik.
- Pemulihan DDL dan DDL cepat tidak didukung di Aurora MySQL versi 3. Tidak boleh ada artefak dalam basis data yang terkait dengan fitur-fitur ini.
- Tabel dengan format baris `COMPACT` atau `REDUNDANT` tidak dapat memiliki indeks yang lebih besar dari 767 byte.
- Panjang awalan indeks yang ditentukan pada kolom teks `tiny` tidak boleh melebihi 255 byte. Dengan set karakter `utf8mb4`, panjang awalan yang didukung dibatasi hingga 63 karakter.

  Panjang awalan yang lebih besar diizinkan di MySQL 5.7 menggunakan parameter `innodb_large_prefix` . Parameter ini tidak digunakan lagi di MySQL 8.0.
- Tidak boleh ada inkonsistensi metadata InnoDB dalam tabel `mysql.host`.
- Tidak boleh ada inkompatibilitas jenis data kolom dalam tabel sistem.
- Tidak boleh ada transaksi XA dalam status `prepared`.
- Jika panjang daftar riwayat (HLL) untuk klaster lebih besar dari 1 juta, upgrade mungkin memakan waktu lebih lama.

  Untuk informasi selengkapnya, lihat Pemecahan masalah untuk upgrade di tempat Aurora MySQL.

- Nama kolom dalam tampilan tidak boleh lebih panjang dari 64 karakter.
- Karakter khusus dalam prosedur tersimpan tidak boleh tidak konsisten.
- Tabel tidak boleh memiliki inkonsistensi jalur file data.

Jalur upgrade versi mayor Aurora MySQL

Tidak semua jenis atau versi klaster Aurora MySQL dapat menggunakan mekanisme upgrade di tempat. Anda dapat mempelajari jalur upgrade yang sesuai untuk setiap klaster Aurora MySQL dengan melihat tabel berikut.

Jenis klaster DB Aurora MySQL	Apakah klaster dapat menggunakan upgrade di tempat?	Tindakan
Klaster yang disediakan Aurora MySQL 2.0 atau versi lebih tinggi	Ya	Upgrade di tempat didukung untuk klaster yang kompatibel dengan Aurora MySQL 5.7. Untuk informasi tentang upgrade ke Aurora MySQL versi 3, lihat Perencanaan peningkatan untuk Aurora MySQL versi 3 dan Meningkatkan ke Aurora MySQL versi 3.
Klaster yang disediakan Aurora MySQL 3.01.0 atau versi lebih tinggi	N/A	Gunakan prosedur upgrade versi minor untuk upgrade di antara Aurora MySQL versi 3.
Klaster Aurora Serverless v1	N/A	Saat ini, Aurora Serverless v1 didukung untuk Aurora MySQL hanya pada versi 2.
Klaster Aurora Serverless v2	N/A	Saat ini, Aurora Serverless v2 didukung untuk Aurora MySQL hanya pada versi 3.
Klaster dalam basis data global Aurora	Ya	Untuk meng-upgrade Aurora MySQL dari versi 2 ke versi 3, ikuti prosedur untuk melakukan upgrade di tempat

Jenis klaster DB Aurora MySQL	Apakah klaster dapat menggunakan upgrade di tempat?	Tindakan
		untuk klaster di basis data global Aurora. Lakukan upgrade pada klaster global. Aurora meng-upgrade klaster primer dan semua klaster sekunder dalam basis data global pada saat yang sama. Jika Anda menggunakan AWS CLI atau API RDS, panggil perintah `modify-global-cluster` atau operasi `ModifyGlobalCluster` bukan `modify-db-cluster` atau `ModifyDBCluster` . Anda tidak dapat melakukan upgrade di tempat dari Aurora MySQL versi 2 ke versi 3 jika parameter `lower_case_table_names` diaktifkan. Untuk informasi selengkapnya, lihat Peningkatan versi utama.
Klaster kueri paralel	Ya	Anda dapat melakukan upgrade di tempat. Dalam hal ini, pilih 2.09.1 atau lebih tinggi untuk versi Aurora MySQL.
Klaster yang merupakan target replikasi log biner	Mungkin	Jika replikasi log biner berasal dari klaster Aurora MySQL, Anda dapat melakukan upgrade di tempat. Anda tidak dapat melakukan upgrade jika replikasi log biner berasal dari instans DB RDS for MySQL atau instans DB MySQL on-premise. Dalam hal ini, Anda dapat melakukan upgrade menggunakan mekanisme pemulihan snapshot.

Jenis klaster DB Aurora MySQL	Apakah klaster dapat menggunakan upgrade di tempat?	Tindakan
Klaster dengan nol instans DB	Tidak	Menggunakan AWS CLI atau API RDS, Anda dapat membuat klaster Aurora MySQL tanpa instans DB terlampir. Dengan cara yang sama, Anda juga dapat menghapus semua instans DB dari klaster Aurora MySQL, tetapi membiarkan data dalam volume klaster tetap utuh. Meskipun sebuah klaster tidak memiliki instans DB, Anda tidak dapat melakukan upgrade di tempat. Mekanisme upgrade memerlukan instans penulis di klaster untuk melakukan konversi pada tabel sistem, file data, dan sebagainya. Dalam kasus ini, gunakan AWS CLI atau API RDS untuk membuat instans penulis untuk klaster. Kemudian, Anda dapat melakukan upgrade di tempat.
Klaster dengan pelacakan mundur diaktifkan	Ya	Anda dapat melakukan upgrade di tempat untuk klaster Aurora MySQL yang menggunakan fitur pelacakan mundur. Namun, setelah upgrade, Anda tidak dapat melacak mundur klaster ke waktu sebelum upgrade.

Cara kerja upgrade di tempat terhadap versi mayor Aurora MySQL

Aurora MySQL melakukan upgrade versi mayor sebagai proses multitahap. Anda dapat memeriksa status upgrade saat ini. Beberapa langkah upgrade juga memberikan informasi progres. Seiring setiap tahap dimulai, Aurora MySQL akan mencatat sebuah peristiwa. Anda dapat memeriksa peristiwa yang terjadi di halaman Peristiwa di konsol RDS. Untuk informasi selengkapnya tentang menggunakan peristiwa, lihat Bekerja dengan pemberitahuan peristiwa Amazon RDS.

> **Important**
>
> Setelah proses dimulai, proses ini berjalan sampai upgrade berhasil atau gagal. Anda tidak dapat membatalkan upgrade saat sedang berlangsung. Jika upgrade gagal, Aurora akan mengembalikan semua perubahan dan klaster Anda akan memiliki versi mesin, metadata, dan karakteristik lainnya yang sama seperti sebelumnya.

Proses upgrade terdiri dari tiga tahap:

1. Aurora melakukan serangkaian pra-pemeriksaan sebelum memulai proses upgrade. Klaster Anda terus berjalan saat Aurora melakukan pemeriksaan ini. Misalnya, klaster tidak dapat memiliki transaksi XA apa pun dalam status siap atau memproses pernyataan bahasa definisi data (DDL). Misalnya, Anda mungkin perlu menutup aplikasi yang mengirimkan jenis pernyataan SQL tertentu. Atau, Anda mungkin hanya perlu menunggu sampai pernyataan berjalan lama tertentu diselesaikan. Kemudian, coba mulai upgrade lagi. Beberapa pemeriksaan akan menguji kondisi yang tidak mencegah upgrade, tetapi mungkin akan membuat upgrade memakan waktu lama.

   Jika Aurora mendeteksi bahwa kondisi yang diperlukan tidak terpenuhi, ubah kondisi yang diidentifikasi dalam detail peristiwa. Ikuti panduan dalam Pemecahan masalah untuk upgrade di tempat Aurora MySQL. Jika Aurora mendeteksi kondisi yang mungkin menyebabkan upgrade yang lambat, rencanakan untuk memantau upgrade dalam waktu lama.
2. Aurora akan membuat klaster Anda menjadi offline. Kemudian, Aurora melakukan serangkaian pengujian serupa seperti pada tahap sebelumnya untuk memastikan bahwa tidak ada masalah baru yang muncul selama proses penonaktifan. Jika Aurora mendeteksi kondisi apa pun pada saat ini yang akan mencegah upgrade, Aurora akan membatalkan upgrade dan membuat klaster kembali online. Dalam hal ini, konfirmasikan jika kondisi tidak lagi berlaku dan mulai upgrade lagi.
3. Aurora membuat snapshot dari volume klaster Anda. Misalkan Anda menemukan kompatibilitas atau jenis masalah lainnya setelah upgrade selesai. Atau misalkan Anda ingin melakukan pengujian menggunakan klaster asli dan klaster yang di-upgrade. Dalam kasus tersebut, Anda dapat memulihkan dari snapshot ini untuk membuat klaster baru dengan versi mesin asli dan data asli.

   > **Tip**
   >
   > Snapshot ini adalah snapshot manual. Namun, Aurora dapat membuatnya dan melanjutkan proses upgrade bahkan jika Anda telah mencapai kuota untuk snapshot

manual. Snapshot ini tetap ada secara permanen (jika diperlukan) hingga Anda menghapusnya. Setelah Anda menyelesaikan semua pengujian pasca-upgrade, Anda dapat menghapus snapshot ini untuk meminimalkan biaya penyimpanan.

4. Aurora akan mengkloning volume klaster Anda. Kloning adalah operasi cepat yang tidak memerlukan penyalinan data tabel yang sebenarnya. Jika Aurora mengalami masalah selama upgrade, ia kembali ke data asli dari volume klaster kloning dan membawa klaster kembali online. Volume yang dikloning sementara selama upgrade tidak memiliki batas yang biasanya berlaku pada jumlah klon untuk satu volume klaster.
5. Aurora melakukan penonaktifan bersih untuk instans DB penulis. Selama penonaktifan bersih, progres peristiwa dicatat setiap 15 menit untuk operasi berikut. Anda dapat memeriksa peristiwa yang terjadi di halaman Peristiwa di konsol RDS.
   - Aurora membersihkan catatan undo untuk baris versi lama.
   - Aurora mengembalikan setiap transaksi yang tidak dikomit.
6. Aurora meng-upgrade versi mesin pada instans DB penulis:
   - Aurora menginstal biner untuk versi mesin baru pada instans DB penulis.
   - Aurora menggunakan instans DB penulis untuk meng-upgrade data Anda ke format yang kompatibel dengan MySQL 5.7. Selama tahap ini, Aurora memodifikasi tabel sistem dan melakukan konversi lain yang memengaruhi data dalam volume klaster Anda. Secara khusus, Aurora meng-upgrade metadata partisi di tabel sistem agar kompatibel dengan format partisi MySQL 5.7. Tahap ini dapat memakan waktu lama jika tabel di klaster Anda memiliki sejumlah besar partisi.

     Jika terjadi kesalahan selama tahap ini, Anda dapat menemukan detailnya dalam log kesalahan MySQL. Setelah tahap ini dimulai, jika proses upgrade gagal karena alasan apa pun, Aurora akan mengembalikan data asli dari volume klaster yang dikloning.
7. Aurora meng-upgrade versi mesin pada instans DB pembaca.
8. Proses upgrade selesai. Aurora mencatat peristiwa akhir untuk menunjukkan bahwa proses upgrade berhasil selesai. Sekarang klaster DB Anda menjalankan versi mayor baru.

Teknik upgrade blue-green alternatif

Dalam beberapa situasi, prioritas utama Anda adalah untuk melakukan switchover langsung dari klaster lama ke klaster yang di-upgrade. Dalam situasi seperti itu, Anda dapat menggunakan proses multistep yang menjalankan cluster side-by-side lama dan baru. Di sini, Anda mereplikasi data dari

klaster yang lama ke klaster baru hingga Anda siap saat klaster baru mengambil alih data. Untuk detailnya, lihat Menggunakan Deployment Blue/Green Amazon RDS untuk pembaruan basis data.

Cara melakukan upgrade di tempat

Sebaiknya Anda meninjau materi latar belakang dalam Cara kerja upgrade di tempat terhadap versi mayor Aurora MySQL.

Lakukan perencanaan dan pengujian sebelum upgrade, seperti yang dijelaskan sebagai berikut:

- Merencanakan upgrade versi mayor untuk klaster Aurora MySQL
- Perencanaan peningkatan untuk Aurora MySQL versi 3

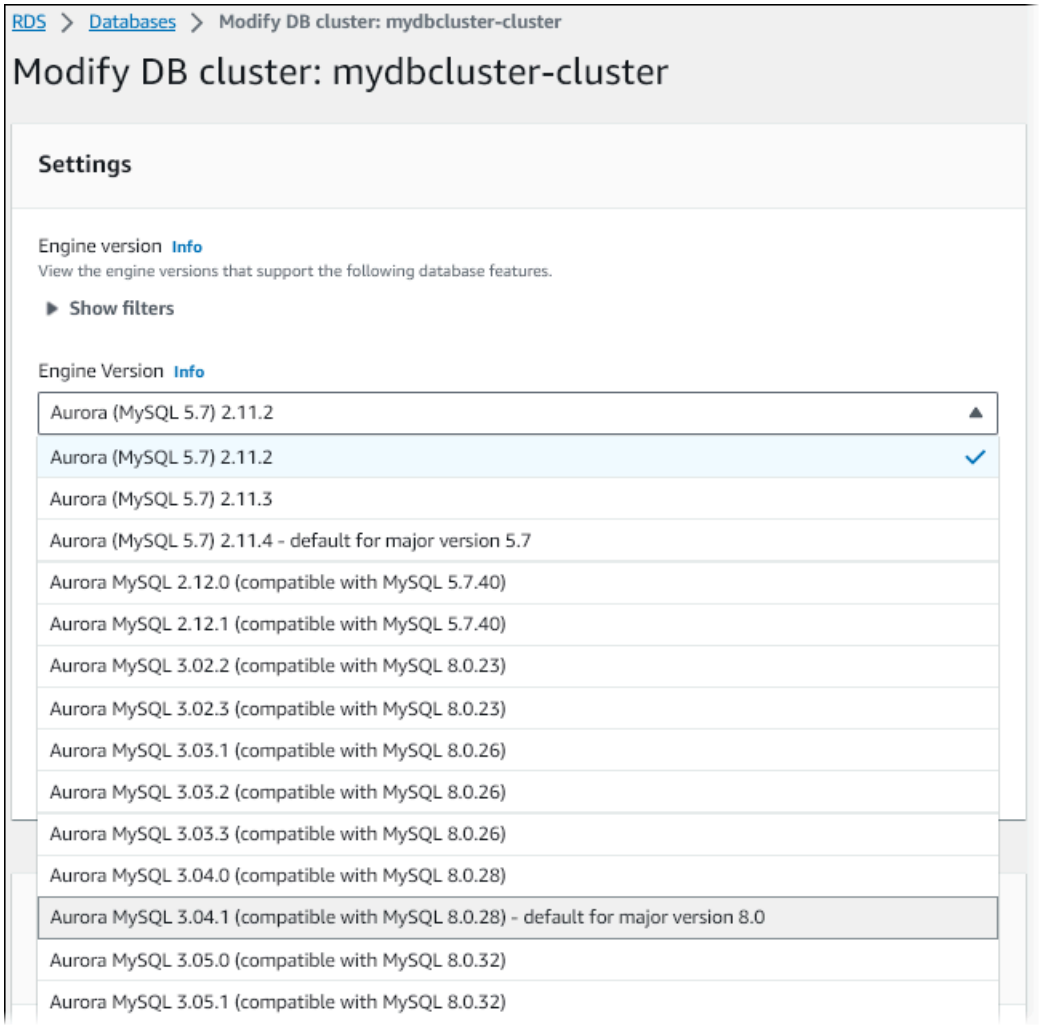
Konsol

Contoh berikut meningkatkan cluster `mydbcluster-cluster` DB ke Aurora MySQL versi 3.04.1.

Untuk meng-upgrade versi mayor klaster DB Aurora MySQL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Jika Anda menggunakan grup parameter kustom untuk klaster DB asli, buat grup parameter terkait yang kompatibel dengan versi mayor baru. Buat penyesuaian yang diperlukan untuk parameter konfigurasi di grup parameter baru tersebut. Untuk informasi selengkapnya, lihat Bagaimana upgrade di tempat memengaruhi grup parameter untuk klaster.
3. Di panel navigasi, pilih Basis Data.
4. Dalam daftar, pilih klaster DB yang ingin Anda ubah.
5. Pilih Ubah.
6. Untuk Versi, pilih versi mayor Aurora MySQL baru.

   Biasanya, kami merekomendasikan untuk menggunakan versi minor terbaru dari versi mayor. Di sini, kami memilih versi default saat ini.



7. Pilih Lanjutkan.
8. Di halaman berikutnya, tentukan kapan harus melakukan upgrade. Pilih Selama jendela pemeliharaan terjadwal berikutnya atau Segera.
9. (Opsional) Periksa secara berkala halaman Peristiwa di konsol RDS selama upgrade. Tindakan ini akan membantu Anda memantau progres upgrade dan identifikasi masalah apa pun. Jika upgrade mengalami masalah apa pun, lihat Pemecahan masalah untuk upgrade di tempat Aurora MySQL untuk langkah-langkah yang harus diambil.
10. Jika Anda membuat grup parameter baru di awal prosedur ini, kaitkan grup parameter kustom dengan klaster yang di-upgrade. Untuk informasi selengkapnya, lihat Bagaimana upgrade di tempat memengaruhi grup parameter untuk klaster.

> **Note**
>
> Langkah ini mengharuskan Anda untuk memulai ulang klaster lagi untuk menerapkan grup parameter baru.

11. (Opsional) Setelah Anda menyelesaikan setiap pengujian pasca-upgrade, hapus snapshot manual yang dibuat Aurora pada awal upgrade.

AWS CLI

Untuk memutakhirkan versi utama cluster DB MySQL Aurora, gunakan AWS CLI modify-db-clusterperintah dengan parameter yang diperlukan berikut:

- `--db-cluster-identifier`
- `--engine-version`
- `--allow-major-version-upgrade`
- `--apply-immediately` atau `--no-apply-immediately`

Jika klaster Anda menggunakan grup parameter kustom, sertakan juga salah satu atau kedua opsi berikut ini:

- `--db-cluster-parameter-group-name`, jika klaster menggunakan grup parameter klaster kustom
- `--db-instance-parameter-group-name`, jika ada instans di klaster yang menggunakan grup parameter DB kustom

Contoh berikut meningkatkan cluster `sample-cluster` DB ke Aurora MySQL versi 3.04.1. Upgrade akan segera terjadi, bukan menunggu periode pemeliharaan berikutnya.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
          --db-cluster-identifier sample-cluster \
          --engine-version 8.0.mysql_aurora.3.04.1 \
```

```
          --allow-major-version-upgrade \
          --apply-immediately
```

Untuk Windows:

```
aws rds modify-db-cluster ^
          --db-cluster-identifier sample-cluster ^
          --engine-version 8.0.mysql_aurora.3.04.1 ^
          --allow-major-version-upgrade ^
          --apply-immediately
```

Anda dapat menggabungkan perintah CLI lainnya `modify-db-cluster` untuk membuat end-to-end proses otomatis untuk melakukan dan memverifikasi peningkatan. Untuk informasi selengkapnya dan contoh tambahan, lihat Tutorial upgrade di tempat Aurora MySQL.

> **Note**
>
> Jika klaster Anda adalah bagian dari basis data global Aurora, prosedur upgrade di tempat ini sedikit berbeda. Anda memanggil operasi modify-global-clusterperintah alih-alih`modify-db-cluster`. Untuk informasi selengkapnya, lihat Upgrade besar di tempat untuk basis data global.

API RDS

Untuk meng-upgrade versi mayor klaster DB Aurora MySQL, gunakan perintah ModifyDBCluster API RDS dengan parameter wajib berikut:

- `DBClusterIdentifier`
- `Engine`
- `EngineVersion`
- `AllowMajorVersionUpgrade`
- `ApplyImmediately` (atur ke `true` atau `false`)

> **Note**
>
> Jika klaster Anda adalah bagian dari basis data global Aurora, prosedur upgrade di tempat ini sedikit berbeda. Anda memanggil ModifyGlobalClusteroperasi alih-alih`ModifyDBCluster`. Untuk informasi selengkapnya, lihat Upgrade besar di tempat untuk basis data global.

Bagaimana upgrade di tempat memengaruhi grup parameter untuk klaster

Grup parameter Aurora memiliki kumpulan pengaturan konfigurasi yang berbeda untuk klaster yang kompatibel dengan MySQL 5.7 atau 8.0. Saat Anda melakukan upgrade di tempat, klaster yang di-upgrade dan semua instans harus menggunakan grup parameter klaster dan instans yang sesuai:

Klaster dan instans Anda mungkin menggunakan grup parameter default yang kompatibel dengan 5.7. Jika demikian, klaster dan instans yang di-upgrade akan dimulai dengan grup parameter default yang kompatibel dengan 8.0. Jika klaster dan instans Anda menggunakan grup parameter kustom apa pun, pastikan untuk membuat grup parameter yang sesuai atau yang kompatibel dengan 8.0. Pastikan juga untuk menentukannya selama proses upgrade.

> **Important**
>
> Jika Anda menentukan grup parameter kustom apa pun selama proses upgrade, pastikan untuk mem-boot ulang klaster secara manual setelah upgrade selesai. Tindakan ini akan membuat klaster mulai menggunakan pengaturan parameter kustom Anda.

Perubahan pada properti klaster di antara versi Aurora MySQL

Saat Anda meng-upgrade dari Aurora MySQL versi 2 ke versi 3, pastikan untuk memeriksa aplikasi atau skrip apa pun yang Anda gunakan untuk menyiapkan atau mengelola klaster dan instans DB Aurora MySQL.

Selain itu, ubah kode Anda yang memanipulasi grup parameter untuk memperhitungkan fakta bahwa nama grup parameter default berbeda untuk klaster yang kompatibel dengan 5.7 dan 8.0. Nama grup parameter default untuk klaster Aurora MySQL versi 2 dan 3 masing-masing adalah `default.aurora-mysql5.7` dan `default.aurora-mysql8.0`.

Misalnya, Anda mungkin memiliki kode seperti berikut yang berlaku untuk klaster Anda sebelum upgrade.

```
# Check the default parameter values for MySQL 5.7-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql5.7 --
region us-east-1
```

Setelah meng-upgrade versi mayor klaster, ubah kode tersebut sebagai berikut.

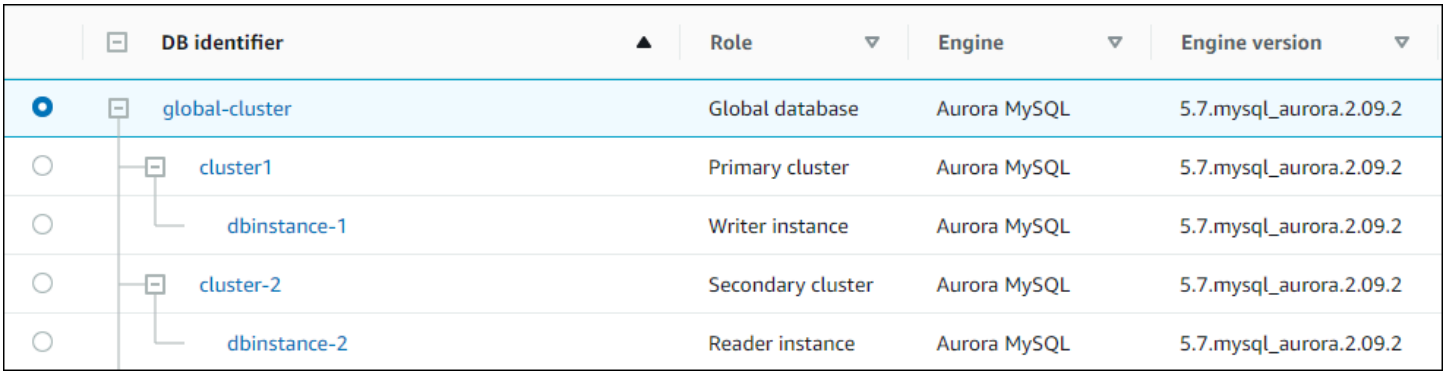
```
# Check the default parameter values for MySQL 8.0-compatible clusters.
aws rds describe-db-parameters --db-parameter-group-name default.aurora-mysql8.0 --
region us-east-1
```

Upgrade besar di tempat untuk basis data global

Untuk basis data global Aurora, Anda meng-upgrade klaster basis data global. Aurora secara otomatis meng-upgrade semua klaster pada saat yang sama dan memastikan bahwa semua klaster ini menjalankan versi mesin yang sama. Persyaratan ini berlaku karena setiap perubahan pada tabel sistem, format file data, dan sebagainya akan secara otomatis direplikasi ke semua klaster sekunder.

Ikuti petunjuk dalam Cara kerja upgrade di tempat terhadap versi mayor Aurora MySQL. Saat Anda menentukan hal yang akan di-upgrade, pastikan untuk memilih klaster basis data global, bukan salah satu klaster yang terdapat di dalamnya.

Jika Anda menggunakan AWS Management Console, pilih item dengan peran Basis data global.



Jika Anda menggunakan AWS CLI atau RDS API, mulai proses upgrade dengan memanggil modify-global-clusterperintah atau ModifyGlobalClusteroperasi. Anda akan menggunakan salah satunya, bukan `modify-db-cluster` atau `ModifyDBCluster`.

> **Note**
>
> Anda tidak dapat menentukan grup parameter kustom untuk klaster basis data global saat Anda melakukan upgrade versi mayor basis data global Aurora tersebut. Buat grup parameter

kustom Anda di setiap Wilayah klaster global. Kemudian, terapkan secara manual ke klaster Regional setelah upgrade.

Untuk meng-upgrade versi utama dari cluster database global MySQL Aurora dengan menggunakanAWS CLI, gunakan modify-global-clusterperintah dengan parameter yang diperlukan berikut:

- `--global-cluster-identifier`
- `--engine aurora-mysql`
- `--engine-version`
- `--allow-major-version-upgrade`

Contoh berikut meng-upgrade klaster basis data global ke Aurora MySQL versi 2.10.2.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-global-cluster \
          --global-cluster-identifier global_cluster_identifier \
          --engine aurora-mysql \
          --engine-version 5.7.mysql_aurora.2.10.2 \
          --allow-major-version-upgrade
```

Untuk Windows:

```
aws rds modify-global-cluster ^
          --global-cluster-identifier global_cluster_identifier ^
          --engine aurora-mysql ^
          --engine-version 5.7.mysql_aurora.2.10.2 ^
          --allow-major-version-upgrade
```

Backtrack pertimbangan

Jika klaster yang Anda upgrade memiliki fitur Pelacakan Mundur diaktifkan, Anda tidak dapat melacak mundur klaster yang di-upgrade ke waktu sebelum upgrade.

Pemecahan masalah untuk upgrade di tempat Aurora MySQL

Gunakan tips berikut untuk membantu memecahkan masalah terkait upgrade di tempat Aurora MySQL. Tips ini tidak berlaku untuk klaster DB Aurora Serverless.

Alasan upgrade di tempat dibatalkan atau lambat	Efek	Solusi untuk memungkinkan upgrade di tempat selesai dalam periode pemeliharaan
Klaster memiliki transaksi XA dalam status siap	Aurora membatalkan upgrade.	Komit atau kembalikan semua transaksi XA yang disiapkan.
Klaster sedang memproses pernyataan bahasa definisi data (DDL)	Aurora membatalkan upgrade.	Pertimbangkan untuk menunggu dan melakukan upgrade setelah semua pernyataan DDL selesai.
Klaster memiliki perubahan yang tidak dikomit untuk banyak baris	Upgrade mungkin memerlukan waktu lama.	Proses upgrade mengembalikan perubahan yang tidak dikomit. Indikator untuk kondisi ini adalah nilai `TRX_ROWS_MODIFIED` dalam tabel `INFORMATION_SCHEMA.INNODB_TRX` . Pertimbangkan untuk melakukan upgrade hanya setelah semua transaksi besar dilakukan atau dikembalikan.
Klaster memiliki jumlah catatan undo yang tinggi	Upgrade mungkin memerlukan waktu lama.	Bahkan jika transaksi yang tidak dikomit tidak memengaruhi sejumlah besar baris, transaksi ini mungkin berkaitan dengan volume besar data. Misalnya, Anda mungkin memasukkan BLOB besar. Aurora tidak akan secara otomatis mendeteksi atau menghasilkan peristiwa untuk jenis aktivitas transaksi ini. Indikator untuk kondisi ini adalah panjang daftar riwayat (HLL). Proses upgrade mengembalikan perubahan yang tidak dikomit. Anda dapat memeriksa HLL dalam output dari perintah SQL `SHOW ENGINE INNODB`

Alasan upgrade di tempat dibatalkan atau lambat	Efek	Solusi untuk memungkinkan upgrade di tempat selesai dalam periode pemeliharaan
		STATUS, atau langsung dengan menggunakan kueri SQL berikut: `SELECT count FROM information_schema .innodb_metrics WHERE name = 'trx_rseg_history_len';` Anda juga dapat memantau `RollbackSegmentHistoryListLength` metrik di Amazon CloudWatch. Pertimbangkan untuk melakukan upgrade hanya setelah HLL menjadi lebih kecil.
Klaster sedang dalam proses melakukan komit transaksi log biner besar	Upgrade mungkin memerlukan waktu lama.	Proses upgrade menunggu sampai perubahan log biner diterapkan. Transaksi atau pernyataan DDL lainnya dapat dimulai selama periode ini, sehingga akan makin memperlambat proses upgrade. Jadwalkan proses upgrade saat klaster tidak sibuk menghasilkan perubahan replikasi log biner. Aurora tidak akan secara otomatis mendeteksi atau membuat peristiwa untuk kondisi ini.

Alasan upgrade di tempat dibatalkan atau lambat	Efek	Solusi untuk memungkinkan upgrade di tempat selesai dalam periode pemeliharaan
Inkonsistensi skema yang dihasilkan dari penghapusan atau kerusakan file	Aurora membatalkan upgrade.	Ubah mesin penyimpanan default untuk tabel sementara dari MyISAM menjadi InnoDB. Lakukan langkah-langkah berikut ini: 1. Ubah parameter DB `default_tmp_storage_engine` menjadi `InnoDB`. 2. Boot ulang klaster DB. 3. Setelah boot ulang, konfirmasikan bahwa parameter DB `default_tmp_storage_engine` diatur ke `InnoDB`. Gunakan perintah berikut ini. `show global variables like 'default_tmp_storage_engine';` 4. Pastikan untuk tidak membuat tabel sementara yang menggunakan mesin penyimpanan MyISAM. Kami menyarankan agar Anda menjeda beban kerja basis data apa pun dan tidak membuat koneksi basis data baru karena Anda akan segera melakukan upgrade. 5. Coba lagi upgrade di tempat.

Alasan upgrade di tempat dibatalkan atau lambat	Efek	Solusi untuk memungkinkan upgrade di tempat selesai dalam periode pemeliharaan

Pengguna master dihapus	Aurora membatalkan upgrade.	**Important**
		Jangan hapus pengguna master.
		
		Namun, jika karena alasan tertentu Anda kebetulan menghapus pengguna master, kembalikan pengguna master menggunakan perintah SQL berikut:
		
		`CREATE USER 'master_username '@'% IDENTIFIED BY ' master_user_password ' REQUIRE NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK;`
		
		`GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, PROCESS, REFERENCES, INDEX, ALTER, SHOW DATABASES, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, LOAD FROM S3, SELECT INTO S3, INVOKE LAMBDA, INVOKE SAGEMAKER, INVOKE COMPREHEND ON *.* TO 'master_username '@'% WITH GRANT OPTION;`

Anda dapat menggunakan langkah-langkah berikut untuk melakukan pemeriksaan Anda sendiri untuk beberapa kondisi di tabel sebelumnya. Dengan demikian, Anda dapat menjadwalkan upgrade pada saat Anda mengetahui basis data dalam keadaan yang memungkinkan upgrade berhasil diselesaikan dengan cepat.

- Anda dapat memeriksa transaksi XA terbuka dengan menjalankan pernyataan `XA RECOVER`. Anda kemudian dapat mengkomit atau mengembalikan transaksi XA sebelum memulai upgrade.

- Anda dapat memeriksa pernyataan DDL dengan menjalankan pernyataan `SHOW PROCESSLIST` dan mencari pernyataan `CREATE`, `DROP`, `ALTER`, `RENAME`, dan `TRUNCATE` dalam output. Tunggu semua pernyataan DDL selesai sebelum memulai upgrade.
- Anda dapat memeriksa jumlah total baris yang tidak dikomit dengan mengkueri tabel `INFORMATION_SCHEMA.INNODB_TRX`. Tabel ini berisi satu baris untuk setiap transaksi. Kolom `TRX_ROWS_MODIFIED` berisi jumlah baris yang diubah atau dimasukkan oleh transaksi.
- Anda dapat memeriksa panjang daftar riwayat InnoDB dengan menjalankan pernyataan `SHOW ENGINE INNODB STATUS SQL` dan mencari `History list length` dalam output. Anda juga dapat memeriksa nilai secara langsung dengan menjalankan kueri berikut:

  ```
  SELECT count FROM information_schema.innodb_metrics WHERE name =
   'trx_rseg_history_len';
  ```

  Panjang daftar riwayat sesuai dengan jumlah informasi undo yang disimpan oleh basis data untuk menerapkan kontrol konkurensi multiversi (MVCC).

Tutorial upgrade di tempat Aurora MySQL

Contoh Linux berikut menunjukkan cara Anda dapat melakukan langkah-langkah umum untuk prosedur upgrade di tempat menggunakan AWS CLI.

Contoh pertama ini membuat klaster DB Aurora yang menjalankan Aurora MySQL versi 2.x. Klaster ini mencakup instans DB penulis dan instans DB pembaca. Perintah `wait db-instance-available` dijeda hingga instans DB penulis tersedia. Pada saat itulah, klaster siap untuk di-upgrade.

```
aws rds create-db-cluster --db-cluster-identifier mynewdbcluster --engine aurora-mysql
 \
  --db-cluster-version 5.7.mysql_aurora.2.10.2
...
aws rds create-db-instance --db-instance-identifier mynewdbcluster-instance1 \
  --db-cluster-identifier mynewdbcluster --db-instance-class db.t4g.medium --engine
 aurora-mysql
...
aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

Aurora MySQL versi 3.x yang dapat Anda upgrade untuk klaster akan bergantung pada versi 2.x yang sedang dijalankan klaster dan Wilayah AWS tempat klaster berada. Perintah pertama, dengan

`--output text`, hanya menunjukkan versi target yang tersedia. Perintah kedua menunjukkan output lengkap JSON dari respons. Dalam respons tersebut, Anda dapat melihat detail seperti nilai `aurora-mysql` yang Anda gunakan untuk parameter `engine`. Anda juga dapat melihat fakta bahwa upgrade ke 3.02.0 merupakan upgrade versi mayor.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
5.7.mysql_aurora.2.10.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.10.2 \
  --query '*[].[ValidUpgradeTarget]'
...
{
    "Engine": "aurora-mysql",
    "EngineVersion": "8.0.mysql_aurora.3.02.0",
    "Description": "Aurora MySQL 3.02.0 (compatible with MySQL 8.0.23)",
    "AutoUpgrade": false,
    "IsMajorVersionUpgrade": true,
    "SupportedEngineModes": [
        "provisioned"
    ],
    "SupportsParallelQuery": true,
    "SupportsGlobalDatabases": true,
    "SupportsBabelfish": false
},
...
```

Contoh ini menunjukkan bahwa jika Anda memasukkan nomor versi target yang bukan merupakan target upgrade yang valid untuk klaster, Aurora tidak akan melakukan upgrade. Aurora juga tidak melakukan upgrade versi mayor kecuali jika Anda menyertakan parameter `--allow-major-version-upgrade`. Dengan demikian, Anda tidak dapat tanpa disengaja melakukan upgrade yang berpotensi akan memerlukan pengujian dan perubahan ekstensif terhadap kode aplikasi Anda.

```
aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 5.7.mysql_aurora.2.09.2 --apply-immediately
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: Cannot find upgrade target from 5.7.mysql_aurora.2.10.2 with requested
 version 5.7.mysql_aurora.2.09.2.

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.02.0 --region us-east-1 --apply-immediately
```

```
An error occurred (InvalidParameterCombination) when calling the ModifyDBCluster
 operation: The AllowMajorVersionUpgrade flag must be present when upgrading to a new
 major version.

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.02.0 --apply-immediately --allow-major-version-
upgrade
{
  "DBClusterIdentifier": "mynewdbcluster",
  "Status": "available",
  "Engine": "aurora-mysql",
  "EngineVersion": "5.7.mysql_aurora.2.10.2"
}
```

Dibutuhkan beberapa saat hingga status klaster dan instans DB terkait berubah menjadi `upgrading`. Nomor versi untuk klaster dan instans DB hanya berubah saat upgrade selesai. Sekali lagi, Anda dapat menggunakan perintah `wait db-instance-available` untuk instans DB penulis untuk menunggu hingga upgrade selesai sebelum melanjutkan.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[Status,EngineVersion]' --output text
upgrading 5.7.mysql_aurora.2.10.2

aws rds describe-db-instances --db-instance-identifier mynewdbcluster-instance1 \
  --query '*[].
{DBInstanceIdentifier:DBInstanceIdentifier,DBInstanceStatus:DBInstanceStatus} | [0]'
{
    "DBInstanceIdentifier": "mynewdbcluster-instance1",
    "DBInstanceStatus": "upgrading"
}

aws rds wait db-instance-available --db-instance-identifier mynewdbcluster-instance1
```

Pada tahap ini, nomor versi untuk klaster akan cocok dengan nomor versi yang ditentukan untuk upgrade.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].[EngineVersion]' --output text

8.0.mysql_aurora.3.02.0
```

Contoh sebelumnya melakukan upgrade langsung dengan menentukan parameter `--apply-immediately`. Agar upgrade terjadi pada waktu yang tepat saat klaster diperkirakan tidak sibuk, Anda dapat menentukan parameter `--no-apply-immediately`. Tindakan ini akan membuat upgrade dimulai selama periode pemeliharaan berikutnya untuk klaster. Periode pemeliharaan menentukan periode saat operasi pemeliharaan dapat dimulai. Operasi yang berjalan lama mungkin tidak selesai selama periode pemeliharaan. Dengan demikian, Anda tidak perlu menentukan periode pemeliharaan yang lebih panjang bahkan jika Anda memperkirakan bahwa upgrade mungkin akan memakan waktu lama.

Contoh berikut meng-upgrade klaster yang awalnya menjalankan Aurora MySQL versi 2.10.2. Dalam output `describe-db-engine-versions`, nilai `False` dan `True` merepresentasikan properti `IsMajorVersionUpgrade`. Dari versi 2.10.2, Anda dapat meng-upgrade ke beberapa versi 2.* lainnya. Upgrade tersebut tidak dianggap sebagai upgrade versi mayor sehingga tidak memerlukan upgrade di tempat. Upgrade di tempat hanya tersedia untuk upgrade ke versi 3.* yang ditampilkan dalam daftar.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster \
  --query '*[].{EngineVersion:EngineVersion}' --output text
5.7.mysql_aurora.2.10.2

aws rds describe-db-engine-versions --engine aurora-mysql --engine-version
 5.7.mysql_aurora.2.10.2 \
  --query '*[].[ValidUpgradeTarget]|[0][0]|[*].[EngineVersion,IsMajorVersionUpgrade]'
 --output text

5.7.mysql_aurora.2.10.3 False
5.7.mysql_aurora.2.11.0 False
5.7.mysql_aurora.2.11.1 False
8.0.mysql_aurora.3.01.1 True
8.0.mysql_aurora.3.02.0 True
8.0.mysql_aurora.3.02.2 True


aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster \
  --engine-version 8.0.mysql_aurora.3.02.0 --no-apply-immediately --allow-major-
version-upgrade
...
```

Ketika sebuah klaster dibuat tanpa periode pemeliharaan yang ditentukan, Aurora akan memilih hari acak dalam seminggu. Dalam hal ini, perintah `modify-db-cluster` dikirimkan pada hari

Senin. Dengan demikian, kita mengubah periode pemeliharaan menjadi Selasa pagi. Semua waktu ditampilkan dalam zona waktu UTC. Periode `tue:10:00-tue:10:30` sama dengan 02.00-02.30 waktu Pasifik. Perubahan periode pemeliharaan akan langsung berlaku.

```
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "sat:08:20-sat:08:50"
    ]
]

aws rds modify-db-cluster --db-cluster-identifier mynewdbcluster --preferred-
maintenance-window tue:10:00-tue:10:30"
aws rds describe-db-clusters --db-cluster-identifier mynewdbcluster --query '*[].
[PreferredMaintenanceWindow]'
[
    [
        "tue:10:00-tue:10:30"
    ]
]
```

Contoh berikut menunjukkan cara mendapatkan laporan dari peristiwa yang dihasilkan oleh upgrade. Argumen `--duration` merepresentasikan jumlah menit untuk mengambil informasi peristiwa. Argumen ini diperlukan karena secara default `describe-events` hanya menghasilkan peristiwa dari satu jam terakhir.

```
aws rds describe-events --source-type db-cluster --source-identifier mynewdbcluster --
duration 20160
{
    "Events": [
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "DB cluster created",
            "EventCategories": [
                "creation"
            ],
            "Date": "2022-11-17T01:24:11.093000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
```

```
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing online pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:08.450000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Performing offline pre-upgrade checks.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T22:57:59.519000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Creating pre-upgrade snapshot [preupgrade-
mynewdbcluster-5-7-mysql-aurora-2-10-2-to-8-0-mysql-aurora-3-02-0-2022-11-18-22-55].",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:00:22.318000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Cloning volume.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:01:45.428000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
```

```
            "Message": "Upgrade in progress: Purging undo records for old row versions.
 Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:02:25.141000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Purging undo records for old row versions.
 Records remaining: 164",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:23.036000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Upgrade in progress: Upgrading database objects.",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:06:48.208000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        },
        {
            "SourceIdentifier": "mynewdbcluster",
            "SourceType": "db-cluster",
            "Message": "Database cluster major version has been upgraded",
            "EventCategories": [
                "maintenance"
            ],
            "Date": "2022-11-18T23:10:28.999000+00:00",
            "SourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:mynewdbcluster"
        }
    ]
}
```

Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 3

Untuk informasi yang sebelumnya ada dalam panduan ini, lihat halaman yang sesuai dalam Catatan rilis untuk Amazon Aurora MySQL-Compatible Edition.

Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 2

Untuk informasi yang sebelumnya ada dalam panduan ini, lihat halaman yang sesuai dalam Catatan rilis untuk Amazon Aurora MySQL-Compatible Edition.

Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 1

Untuk informasi yang sebelumnya ada dalam panduan ini, lihat halaman yang sesuai dalam Catatan rilis untuk Amazon Aurora MySQL-Compatible Edition.

Pembaruan mesin basis data untuk klaster Aurora MySQL Serverless

Untuk informasi yang sebelumnya ada dalam panduan ini, lihat halaman yang sesuai dalam Catatan rilis untuk Amazon Aurora MySQL-Compatible Edition.

Bug MySQL diperbaiki oleh pembaruan mesin basis data Aurora MySQL

Untuk informasi yang sebelumnya ada dalam panduan ini, lihat halaman yang sesuai dalam Catatan rilis untuk Amazon Aurora MySQL-Compatible Edition.

Kerentanan keamanan yang diperbaiki di Amazon Aurora MySQL

Untuk informasi yang sebelumnya ada dalam panduan ini, lihat halaman yang sesuai dalam Catatan rilis untuk Amazon Aurora MySQL-Compatible Edition.

Menggunakan Amazon Aurora PostgreSQL

Amazon Aurora PostgreSQL adalah mesin basis data relasional yang terkelola sepenuhnya, kompatibel dengan PostgreSQL, dan memenuhi standar ACID. Mesin basis data ini menggabungkan kecepatan, keandalan, dan pengelolaan Amazon Aurora dengan kesederhanaan dan efektivitas biaya basis data sumber terbuka. Aurora PostgreSQL adalah pengganti "drop-in" untuk PostgreSQL serta memudahkan dan menghemat biaya untuk menyiapkan, mengoperasikan, dan menskalakan deployment PostgreSQL baru dan yang sudah ada, sehingga Anda bebas untuk fokus pada bisnis dan aplikasi Anda. Untuk mempelajari selengkapnya tentang Aurora secara umum, lihat Apa itu Amazon Aurora?.

Selain manfaat Aurora, Aurora PostgreSQL menawarkan jalur migrasi praktis dari Amazon RDS ke Aurora, dengan alat migrasi yang hanya memerlukan beberapa klik untuk mengonversi aplikasi RDS for PostgreSQL Anda yang sudah ada menjadi aplikasi Aurora PostgreSQL. Tugas basis data rutin seperti penyediaan, patching, pencadangan, pemulihan, deteksi kegagalan, dan perbaikan juga mudah dikelola dengan Aurora PostgreSQL.

Aurora PostgreSQL dapat memenuhi berbagai standar industri. Misalnya, Anda dapat menggunakan basis data Aurora PostgreSQL untuk membuat aplikasi yang memenuhi standar HIPAA dan untuk menyimpan informasi terkait layanan kesehatan, termasuk informasi kesehatan yang dilindungi (PHI), berdasarkan Perjanjian Rekanan Bisnis (BAA) dengan AWS.

Aurora PostgreSQL memenuhi syarat FedRAMP tingkat TINGGI. Untuk detail tentang AWS dan upaya kepatuhan, lihat Layanan AWS dalam cakupan berdasarkan program kepatuhan.

Topik

- Bekerja dengan Lingkungan Pratinjau Basis Data
- Keamanan dengan Amazon Aurora PostgreSQL
- Memperbarui aplikasi untuk terhubung ke klaster DB Aurora PostgreSQL menggunakan sertifikat SSL/TLS baru
- Menggunakan autentikasi Kerberos dengan Aurora PostgreSQL
- Memigrasikan data ke Amazon Aurora dengan kompatibilitas PostgreSQL
- Meningkatkan performa kueri untuk Aurora PostgreSQL dengan Aurora Optimized Reads
- Menggunakan Babelfish for Aurora PostgreSQL
- Mengelola Amazon Aurora PostgreSQL
- Menyetel dengan peristiwa tunggu di Aurora PostgreSQL

- Menyesuaikan Aurora PostgreSQL dengan wawasan proaktif Amazon DevOps Guru
- Praktik terbaik dengan Amazon Aurora PostgreSQL
- Replikasi dengan Amazon Aurora PostgreSQL
- Menggunakan Aurora PostgreSQL sebagai Basis Pengetahuan untuk Amazon Bedrock
- Mengintegrasikan Amazon Aurora PostgreSQL dengan layanan AWS lainnya
- Memantau rencana eksekusi kueri untuk Aurora PostgreSQL
- Mengelola rencana eksekusi kueri untuk Aurora PostgreSQL
- Menangani ekstensi dan pembungkus data asing
- Bekerja dengan Ekstensi Bahasa Tepercaya untuk PostgreSQL
- Referensi Amazon Aurora PostgreSQL
- Pembaruan Amazon Aurora PostgreSQL

Bekerja dengan Lingkungan Pratinjau Basis Data

Komunitas PostgreSQL merilis versi utama PostgreSQL yang baru setiap tahun. Demikian pula, Amazon Aurora menyediakan versi utama PostgreSQL sebagai rilis Pratinjau. Hal ini memungkinkan Anda membuat klaster DB menggunakan versi Pratinjau dan menguji fiturnya di Lingkungan Pratinjau Basis Data.

Klaster DB Aurora PostgreSQL di Lingkungan Pratinjau Basis Data secara fungsional mirip dengan klaster DB Aurora PostgreSQL lainnya. Namun, Anda tidak dapat menggunakan versi Pratinjau untuk produksi.

Perhatikan batasan penting berikut:

- Semua instans DB dan klaster DB dihapus 60 hari setelah Anda membuatnya, beserta semua cadangan dan snapshot.
- Anda hanya dapat membuat instans DB di Cloud Privat Virtual (VPC) berdasarkan layanan Amazon VPC.
- Anda tidak dapat menyalin snapshot instans DB ke lingkungan produksi.

Opsi berikut didukung oleh Pratinjau.

- Anda dapat membuat instans DB hanya menggunakan jenis instans r5, r6g, r6i, r7g, x2g, t3 dan t4g. Untuk informasi selengkapnya tentang kelas instans, lihat Kelas instans DB Aurora.

- Anda dapat menggunakan deployment single-AZ dan multi-AZ.
- Anda dapat menggunakan fungsi dump dan load PostgreSQL standar untuk mengekspor basis data dari atau mengimpor basis data ke Lingkungan Pratinjau Basis Data.

Jenis kelas instans DB yang didukung

Amazon Aurora PostgreSQL mendukung kelas instans DB berikut di wilayah pratinjau:

Kelas Memori yang Dioptimalkan

- db.r5 – kelas instans memori yang dioptimalkan
- db.r6g – kelas instans memori yang dioptimalkan yang didukung oleh prosesor AWS Graviton2
- db.r6i – kelas instans memori yang dioptimalkan
- db.x2g – kelas instans memori yang dioptimalkan yang didukung oleh prosesor AWS Graviton2

> **Note**
>
> Untuk informasi selengkapnya tentang daftar kelas instans, lihat Jenis kelas instans DB.

Kelas yang dapat melonjak

- db.t3.medium
- db.t3.large
- db.t4g.medium
- db.t4g.large

Fitur yang tidak didukung di Lingkungan Pratinjau

Fitur berikut ini tidak tersedia di Lingkungan Pratinjau:

- Aurora Serverless v1 dan v2
- Peningkatan versi utama (MVU)
- Tidak ada versi minor baru yang akan dirilis di wilayah pratinjau
- Replikasi masuk RDS for PostgreSQL ke Aurora PostgreSQL

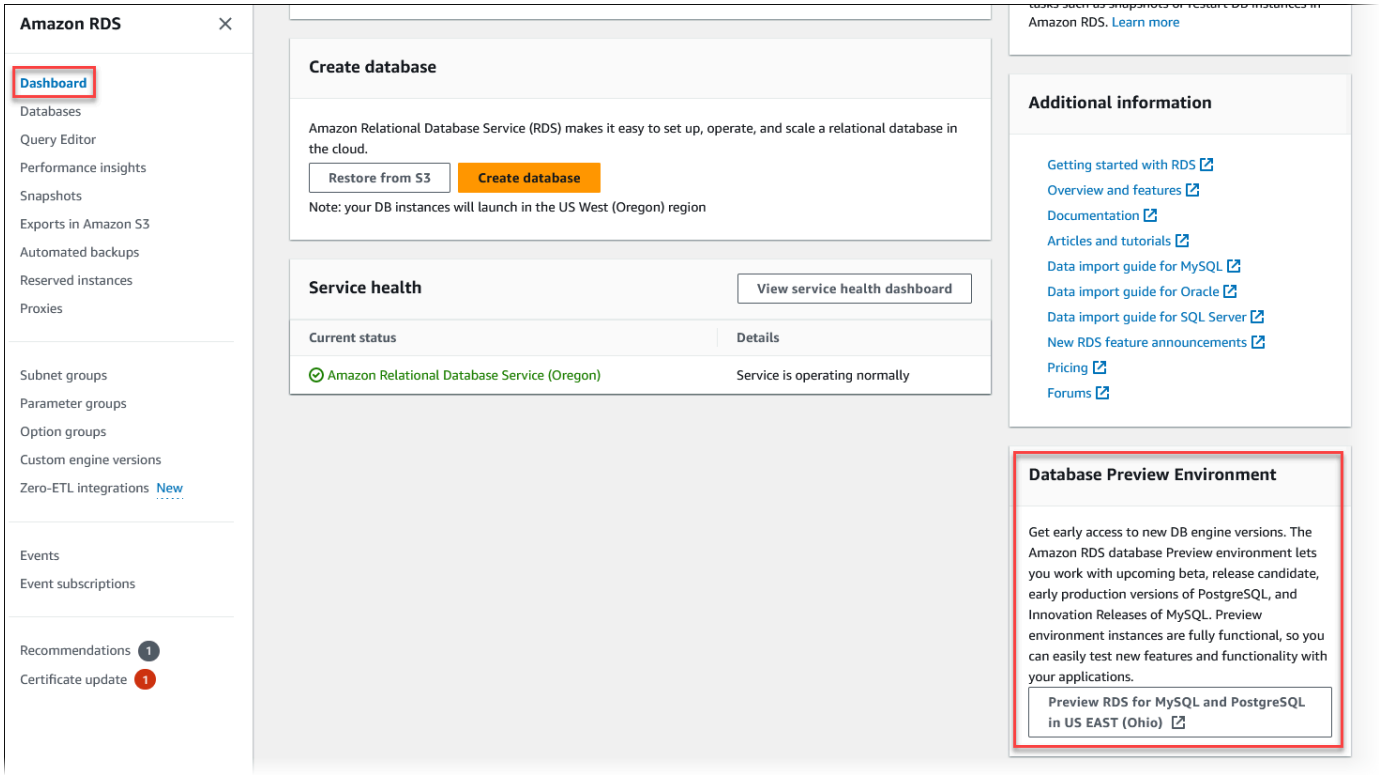
- Deployment Blue/Green Amazon RDS
- Salinan snapshot lintas-wilayah
- Basis data global Aurora
- Aliran aktivitas basis data (DAS), Proxy RDS, dan AWS DMS
- Replika baca penskalaan otomatis
- AWS Bedrock
- Ekspor RDS
- Wawasan Kinerja
- Penerusan tulis global
- Pembacaan yang Dioptimalkan
- Babelfish
- Titik akhir khusus
- Salinan snapshot

Membuat klaster DB baru di Preview Environment

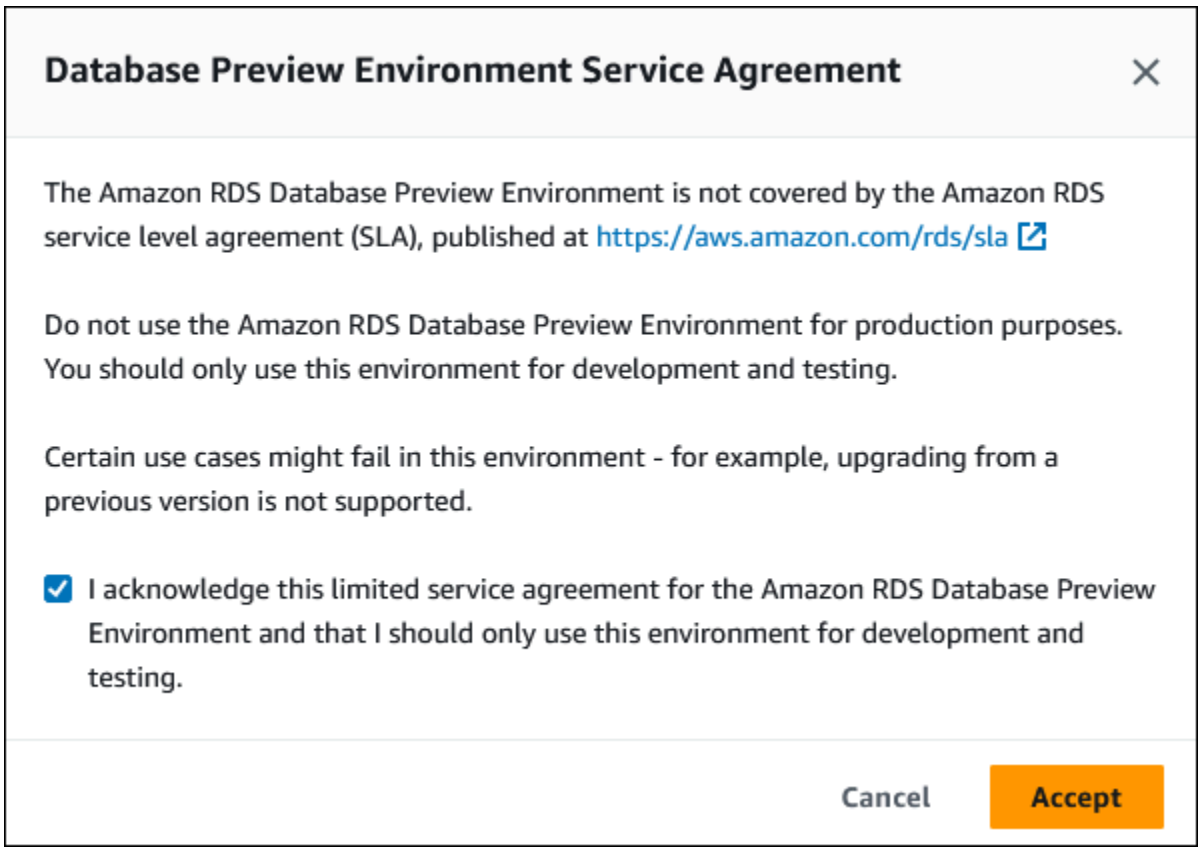
Gunakan prosedur berikut untuk membuat klaster DB di Preview Environment.

Untuk membuat klaster DB di Preview Environment

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Dasbor dari panel navigasi.
3. Di halaman Dasbor, temukan bagian Lingkungan Pratinjau Basis Data, seperti yang ditunjukkan pada gambar berikut.



Anda dapat langsung menuju Lingkungan pratinjau basis data. Sebelum melanjutkan, Anda harus memahami dan menerima batasan.



4. Untuk membuat klaster DB PostgreSQL Aurora, ikuti proses yang sama seperti saat membuat klaster DB Aurora. Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.

Untuk membuat instans di Lingkungan Pratinjau Basis Data menggunakan API Aurora atau AWS CLI, gunakan titik akhir berikut.

```
rds-preview.us-east-2.amazonaws.com
```

PostgreSQL versi 16 di lingkungan Pratinjau Basis Data

⚠ Ini adalah dokumentasi pratinjau untuk Aurora PostgreSQL versi 16. Dokumentasi dapat berubah.

PostgreSQL versi 16.0 kini tersedia di lingkungan Pratinjau Basis Data Amazon RDS. PostgreSQL versi 16 mencakup beberapa peningkatan yang dijelaskan di dokumentasi PostgreSQL berikut:

- PostgreSQL 16 Dirilis

Untuk informasi tentang Lingkungan Pratinjau Basis Data, lihat Bekerja dengan Lingkungan Pratinjau Basis Data. Untuk mengakses Lingkungan Pratinjau dari konsol, pilih https://console.aws.amazon.com/rds-preview/.

> **Note**
>
> Tidak disarankan untuk menggunakan PostgreSQL versi 16.0 di lingkungan Pratinjau Database karena Aurora PostgreSQL versi 16.1 sekarang tersedia secara umum. Untuk informasi selengkapnya, lihat pembaruan Amazon Aurora PostgreSQL.

Keamanan dengan Amazon Aurora PostgreSQL

Untuk gambaran umum tentang keamanan Aurora, lihat. Keamanan dalam Amazon Aurora Anda dapat mengelola keamanan untuk Amazon Aurora PostgreSQL pada beberapa level yang berbeda:

- Untuk mengendalikan siapa yang dapat melakukan tindakan manajemen Amazon RDS di klaster DB dan instans DB Aurora PostgreSQL, Anda menggunakan AWS Identity and Access Management (IAM). IAM menangani autentikasi identitas pengguna sebelum pengguna dapat mengakses layanan. IAM juga menangani otorisasi, yaitu, apakah pengguna diizinkan untuk melakukan apa yang mereka coba lakukan. Autentikasi database IAM adalah metode autentikasi tambahan yang dapat dipilih saat Anda membuat klaster DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Amazon Aurora.

  Jika Anda menggunakan IAM dengan klaster DB Aurora PostgreSQL, masuk ke AWS Management Console dengan kredensial IAM Anda terlebih dahulu, sebelum membuka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
- Pastikan untuk membuat klaster DB Aurora di cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. Untuk mengendalikan perangkat dan instans Amazon EC2 mana yang dapat membuka koneksi ke titik akhir dan port instans DB untuk klaster Aurora DB di VPC, gunakan grup keamanan VPC. Anda dapat membuat titik akhir dan koneksi port ini menggunakan Lapisan Soket Aman (SSL). Selain itu, aturan firewall di perusahaan Anda dapat mengendalikan perangkat yang berjalan di perusahaan Anda untuk dapat membuka koneksi ke instans DB. Untuk informasi selengkapnya tentang VPC, lihat Amazon VPC dan Amazon Aurora.

  Penghunian VPC yang didukung tergantung pada kelas instans DB yang digunakan oleh klaster DB Aurora PostgreSQL Anda. Dengan penghunian VPC `default`, klaster DB berjalan di perangkat keras bersama. Dengan penghunian VPC `dedicated`, klaster DB berjalan di instans

perangkat keras khusus. Kelas instans DB kinerja yang dapat melonjak hanya mendukung penghunian VPC default. Kelas instans DB kinerja yang dapat melonjak mencakup kelas instans DB db.t3 dan db.t4g. Semua kelas instans DB Aurora PostgreSQL lainnya mendukung penghunian VPC default dan khusus.

Untuk informasi selengkapnya tentang kelas instans, lihat Kelas instans DB Aurora. Untuk informasi selengkapnya tentang penghunian VPC `default` dan `dedicated`, lihat Instans khusus dalam Panduan Pengguna Amazon Elastic Compute Cloud.

- Guna memberikan izin ke basis data PostgreSQL yang berjalan di klaster DB Amazon Aurora, Anda dapat menggunakan pendekatan umum yang sama seperti instans PostgreSQL yang berdiri sendiri. Perintah seperti `CREATE ROLE`, `ALTER ROLE`, `GRANT`, dan `REVOKE` bekerja sama seperti dalam basis data on-premise, seperti mengubah langsung basis data, skema, dan tabel.

  PostgreSQL mengelola hak istimewa dengan menggunakan peran. Peran `rds_superuser` adalah peran yang paling istimewa di klaster DB Aurora PostgreSQL. Peran ini dibuat secara otomatis, dan diberikan kepada pengguna yang membuat klaster DB (akun pengguna master, `postgres` secara default). Untuk mempelajari selengkapnya, lihat Memahami peran dan izin PostgreSQL.

Semua versi Aurora PostgreSQL yang tersedia, termasuk versi 10, 11, 12, 13, 14, dan rilis yang lebih tinggi mendukung Salted Challenge Response Authentication Mechanism (SCRAM) untuk kata sandi sebagai alternatif untuk pencernaan pesan (MD5). Kami menyarankan Anda menggunakan SCRAM karena lebih aman daripada MD5. Untuk informasi selengkapnya, termasuk cara memigrasi kata sandi pengguna basis data dari MD5 ke SCRAM, lihat Menggunakan SCRAM untuk enkripsi kata sandi PostgreSQL.

Memahami peran dan izin PostgreSQL

Bila Anda membuat Aurora PostgreSQL DB cluster DB menggunakan, akun administrator dibuat pada saat yang sama. AWS Management Console Secara default, namanya adalah `postgres`, seperti yang ditunjukkan dalam tangkapan layar berikut:

Anda dapat memilih nama lain daripada harus menerima default (`postgres`). Jika ya, nama yang Anda pilih harus dimulai dengan huruf dan berada antara 1 dan 16 karakter alfanumerik. Untuk memudahkan, kami merujuk ke akun pengguna utama ini berdasarkan nilai default-nya (`postgres`) di seluruh panduan ini.

Jika Anda menggunakan `create-db-cluster` AWS CLI bukan AWS Management Console, Anda membuat nama pengguna dengan meneruskannya dengan `master-username` parameter. Lihat informasi yang lebih lengkap di Langkaah 2: Membuat klaster DB Aurora PostgreSQL.

Baik Anda menggunakan AWS Management Console, API AWS CLI, atau Amazon RDS, dan apakah Anda menggunakan `postgres` nama default atau memilih nama yang berbeda, akun pengguna database pertama ini adalah anggota `rds_superuser` grup dan memiliki `rds_superuser` hak istimewa.

Topik

- Memahami peran rds_superuser
- Mengontrol akses pengguna ke basis data PostgreSQL
- Mendelegasikan dan mengendalikan pengelolaan kata sandi pengguna
- Menggunakan SCRAM untuk enkripsi kata sandi PostgreSQL

Memahami peran rds_superuser

Di PostgreSQL, peran dapat menentukan pengguna, grup, atau sekumpulan izin khusus yang diberikan kepada grup atau pengguna untuk berbagai objek dalam basis data. Perintah PostgreSQL ke `CREATE USER` dan `CREATE GROUP` telah digantikan oleh perintah yang lebih umum, `CREATE`

ROLE dengan properti khusus untuk membedakan pengguna basis data. Pengguna basis data dapat dianggap sebagai peran dengan hak istimewa LOGIN.

> **Note**
>
> Perintah CREATE USER dan CREATE GROUP masih dapat digunakan. Untuk informasi selengkapnya, lihat Database Role dalam dokumentasi PostgreSQL.

Pengguna postgres adalah pengguna basis data yang paling istimewa di klaster DB Aurora PostgreSQL. Pengguna tersebut memiliki karakteristik yang ditentukan oleh pernyataan CREATE ROLE berikut.

```
CREATE ROLE postgres WITH LOGIN NOSUPERUSER INHERIT CREATEDB CREATEROLE NOREPLICATION
 VALID UNTIL 'infinity'
```

Properti NOSUPERUSER, NOREPLICATION, INHERIT, dan VALID UNTIL 'infinity' merupakan opsi default untuk CREATE ROLE, kecuali ditentukan lain.

Secara default, postgres memiliki hak istimewa yang diberikan untuk peran rds_superuser, serta izin untuk membuat peran dan basis data. Peran rds_superuser memungkinkan pengguna postgres untuk melakukan berbagai hal berikut:

- Menambahkan ekstensi yang tersedia untuk digunakan dengan Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menangani ekstensi dan pembungkus data asing.
- Membuat peran untuk pengguna dan memberikan hak istimewa kepada pengguna. Untuk informasi selengkapnya, lihat CREATE ROLE dan GRANT dalam dokumentasi PostgreSQL.
- Membuat basis data. Untuk informasi selengkapnya, lihat CREATE DATABASE dalam dokumentasi PostgreSQL.
- Memberikan hak istimewa rds_superuser untuk peran pengguna yang tidak memiliki hak istimewa ini, dan mencabut hak istimewa sesuai kebutuhan. Sebaiknya Anda memberikan peran ini hanya kepada pengguna yang melakukan tugas pengguna super. Dengan kata lain, Anda dapat memberikan peran ini kepada administrator basis data (DBA) atau administrator sistem.
- Memberikan (dan mencabut) peran rds_replication kepada pengguna basis data yang tidak memiliki peran rds_superuser.
- Memberikan (dan mencabut) peran rds_password kepada pengguna basis data yang tidak memiliki peran rds_superuser.

- Mendapatkan informasi status semua koneksi basis data menggunakan tampilan `pg_stat_activity`. Jika diperlukan, `rds_superuser` dapat menghentikan koneksi apa pun menggunakan `pg_terminate_backend` atau `pg_cancel_backend`.

Dalam pernyataan `CREATE ROLE postgres...`, Anda dapat melihat bahwa peran pengguna `postgres` secara khusus melarang izin `superuser` PostgreSQL. Aurora PostgreSQL adalah layanan terkelola sehingga Anda tidak dapat mengakses OS host, dan tidak dapat terhubung menggunakan akun `superuser` PostgreSQL. Banyak tugas yang memerlukan akses `superuser` pada PostgreSQL mandiri dikelola secara otomatis oleh Aurora.

Untuk informasi pemberian hak istimewa selengkapnya, lihat GRANT dalam dokumentasi PostgreSQL.

Peran `rds_superuser` adalah salah satu dari beberapa peran yang telah ditentukan dalam klaster DB Aurora PostgreSQL.

> **Note**
>
> Dalam PostgreSQL 13 dan rilis sebelumnya, peran yang telah ditentukan dikenal sebagai peran default.

Dalam daftar berikut, Anda akan menemukan beberapa peran standar lainnya yang dibuat secara otomatis untuk klaster DB Aurora PostgreSQL baru. Peran yang telah ditentukan dan hak istimewa mereka tidak dapat diubah. Anda tidak dapat menghapus, mengganti nama, atau memodifikasi hak istimewa untuk peran yang telah ditentukan ini. Mencoba untuk melakukannya akan menghasilkan kesalahan.

- rds_password – Peran yang dapat mengubah kata sandi dan mengatur batasan kata sandi untuk pengguna basis data. `rds_superuser`Peran diberikan dengan peran ini secara default, dan dapat memberikan peran tersebut kepada pengguna database. Untuk informasi selengkapnya, lihat Mengontrol akses pengguna ke basis data PostgreSQL.
  - Untuk RDS untuk PostgreSQL versi yang lebih tua dari 14`rds_password`, peran dapat mengubah kata sandi dan mengatur batasan kata sandi untuk pengguna database dan pengguna dengan peran. `rds_superuser` Dari RDS untuk PostgreSQL versi 14 dan yang lebih baru`rds_password`, peran dapat mengubah kata sandi dan mengatur batasan kata sandi hanya untuk pengguna database. Hanya pengguna dengan `rds_superuser` peran yang dapat melakukan tindakan ini pada pengguna lain yang memiliki `rds_superuser` peran.

- rdsadmin – Peran yang dibuat untuk menangani banyak tugas pengelolaan yang akan dilakukan oleh administrator dengan hak istimewa `superuser` di basis data PostgreSQL mandiri. Peran Aurora PostgreSQL untuk banyak tugas pengelolaan.

Untuk melihat semua peran yang telah ditentukan sebelumnya, Anda dapat terhubung ke instans utama klaster DB Aurora PostgreSQL dan menggunakan metacommand `psql \du`. Output akan terlihat sebagai berikut:

```
List of roles
  Role name   |      Attributes                    |      Member of
--------------+------------------------------------+-----------------------------------
postgres      | Create role, Create DB            +| {rds_superuser}
              | Password valid until infinity      |
rds_superuser | Cannot login                       | {pg_monitor,pg_signal_backend,
              |                                   +|   rds_replication,rds_password}
...
```

Dalam output, Anda dapat melihat bahwa `rds_superuser` bukan merupakan peran pengguna basis data (tidak dapat masuk), tetapi memiliki hak istimewa dari banyak peran lainnya. Anda juga dapat melihat bahwa pengguna basis data `postgres` adalah anggota peran `rds_superuser`. Seperti yang disebutkan sebelumnya, `postgres` adalah nilai default di halaman Buat basis data konsol Amazon RDS. Jika Anda memilih nama lain, nama tersebut akan ditampilkan dalam daftar peran sebagai gantinya.

> **Note**
>
> Aurora PostgreSQL versi 15.2 dan 14.7 memperkenalkan perilaku restriktif peran `rds_superuser`. Pengguna PostgreSQL Aurora perlu diberi hak istimewa `CONNECT` pada basis data yang sesuai untuk terhubung meskipun pengguna diberi peran `rds_superuser`. Sebelum Aurora PostgreSQL versi 14.7 dan 15.2, pengguna dapat terhubung ke basis data dan tabel sistem jika diberi peran `rds_superuser`. Perilaku restriktif ini sejalan dengan komitmen Amazon Aurora AWS dan Amazon Aurora untuk peningkatan keamanan yang berkelanjutan.
>
> Harap perbarui logika masing-masing dalam aplikasi Anda jika peningkatan di atas memberikan dampak.

Mengontrol akses pengguna ke basis data PostgreSQL

Basis data baru di PostgreSQL selalu dibuat dengan serangkaian hak istimewa default dalam skema `public` basis data yang memungkinkan semua pengguna basis data dan peran untuk membuat objek. Dengan hak istimewa ini, pengguna basis data dapat terhubung ke basis data, misalnya, dan membuat tabel sementara saat terhubung.

Agar dapat lebih baik dalam mengontrol akses pengguna ke instans basis data yang Anda buat di simpul primer klaster DB Aurora PostgreSQL , sebaiknya Anda mencabut hak istimewa `public` default. Setelah melakukannya, Anda kemudian dapat memberikan hak khusus untuk pengguna basis data dengan lebih terperinci, seperti yang diperlihatkan dalam prosedur berikut ini.

Untuk mengatur peran dan hak istimewa instans basis data baru

Misalkan Anda sedang menyiapkan basis data di klaster DB Aurora PostgreSQL untuk digunakan oleh beberapa peneliti yang semuanya membutuhkan akses baca-tulis ke basis data.

1. Gunakan `psql` (or pgAdmin) untuk terhubun ke instans DB utama di klaster DB Aurora PostgreSQL:

   ```
   psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

   Saat diminta, masukkan kata sandi Anda. Klien `psql` menghubungkan dan menampilkan basis data koneksi administratif default, `postgres=>`, sebagai prompt.

2. Untuk mencegah pengguna basis data membuat objek dalam skema `public`, lakukan hal berikut:

   ```
   postgres=> REVOKE CREATE ON SCHEMA public FROM PUBLIC;
   REVOKE
   ```

3. Selanjutnya, Anda akan membuat instans basis data baru:

   ```
   postgres=> CREATE DATABASE lab_db;
   CREATE DATABASE
   ```

4. Cabut semua hak istimewa dari skema PUBLIC pada basis data baru ini.

   ```
   postgres=> REVOKE ALL ON DATABASE lab_db FROM public;
   ```

```
REVOKE
```

5. Buat peran untuk pengguna basis data.

```
postgres=> CREATE ROLE lab_tech;
CREATE ROLE
```

6. Beri kemampuan untuk terhubung ke basis data kepada pengguna basis data yang memiliki peran ini.

```
postgres=> GRANT CONNECT ON DATABASE lab_db TO lab_tech;
GRANT
```

7. Beri semua hak istimewa pada basis data ini kepada semua pengguna dengan peran `lab_tech`.

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_db TO lab_tech;
GRANT
```

8. Buat pengguna basis data, sebagai berikut:

```
postgres=> CREATE ROLE lab_user1 LOGIN PASSWORD 'change_me';
CREATE ROLE
postgres=> CREATE ROLE lab_user2 LOGIN PASSWORD 'change_me';
CREATE ROLE
```

9. Beri hak istimewa yang terkait dengan peran lab_tech kepada dua pengguna ini:

```
postgres=> GRANT lab_tech TO lab_user1;
GRANT ROLE
postgres=> GRANT lab_tech TO lab_user2;
GRANT ROLE
```

Di titik ini, `lab_user1` dan `lab_user2` dapat terhubung ke basis data `lab_db`. Contoh ini tidak mengikuti praktik terbaik untuk penggunaan perusahaan yang mungkin termasuk membuat beberapa instans basis data, skema yang berbeda, dan pemberian izin terbatas. Untuk informasi lengkap dan skenario tambahan selengkapnya, lihat Mengelola Pengguna dan Peran PostgreSQL.

Untuk informasi hak istimewa di basis data PostgreSQL selengkapnya, lihat perintah GRANT dalam dokumentasi PostgreSQL.

Mendelegasikan dan mengendalikan pengelolaan kata sandi pengguna

Sebagai DBA, Anda mungkin ingin mendelegasikan pengelolaan kata sandi pengguna. Mungkin Anda juga ingin mencegah pengguna basis data mengubah kata sandi mereka atau mengonfigurasi ulang batasan kata sandi, seperti masa pakai kata sandi. Untuk memastikan bahwa hanya pengguna basis data terpilih yang dapat mengubah pengaturan kata sandi, Anda dapat mengaktifkan fitur pengelolaan kata sandi terbatas. Saat Anda mengaktifkan fitur ini, hanya pengguna basis data yang telah diberi peran `rds_password` yang dapat mengelola kata sandi.

> **Note**
>
> Untuk menggunakan manajemen kata sandi terbatas, klaster DB Aurora PostgreSQL harus menjalankan Amazon Aurora PostgreSQL 10.6 atau yang lebih baru.

Secara default, fitur ini dalam keadaan `off`, seperti yang ditunjukkan berikut:

```
postgres=> SHOW rds.restrict_password_commands;
  rds.restrict_password_commands
--------------------------------
 off
(1 row)
```

Untuk mengaktifkan fitur ini, Anda harus menggunakan grup parameter khusus dan mengubah pengaturan `rds.restrict_password_commands` ke 1. Pastikan untuk melakukan booting ulang instans DB utama Aurora PostgreSQL agar pengaturan dapat berlaku.

Dalam keadaan fitur ini aktif, hak istimewa `rds_password` diperlukan untuk perintah SQL berikut:

```
CREATE ROLE myrole WITH PASSWORD 'mypassword';
CREATE ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2023-01-01';
ALTER ROLE myrole WITH PASSWORD 'mypassword';
ALTER ROLE myrole VALID UNTIL '2023-01-01';
ALTER ROLE myrole RENAME TO myrole2;
```

Mengganti nama peran (`ALTER ROLE myrole RENAME TO newname`) juga dibatasi jika kata sandi menggunakan algoritma hashing MD5.

Jika fitur ini dalam keadaan aktif, mencoba salah satu perintah SQL ini tanpa izin peran `rds_password` akan menghasilkan kesalahan berikut:

```
ERROR: must be a member of rds_password to alter passwords
```

Sebaiknya Anda memberikan `rds_password` hanya untuk beberapa peran yang Anda gunakan semata untuk pengelolaan kata sandi. Jika memberikan hak istimewa `rds_password` kepada pengguna basis data yang tidak memiliki hak istimewa `rds_superuser`, Anda juga harus memberi mereka atribut `CREATEROLE`.

Pastikan Anda memverifikasi persyaratan kata sandi seperti kedaluwarsa dan kompleksitas yang diperlukan di sisi klien. Jika Anda menggunakan utilitas sisi klien Anda sendiri untuk perubahan terkait kata sandi, utilitas harus menjadi anggota `rds_password` dan memiliki hak istimewa `CREATE ROLE`.

Menggunakan SCRAM untuk enkripsi kata sandi PostgreSQL

Salted Challenge Response Authentication Mechanism (SCRAM) adalah alternatif untuk algoritma intisari pesan (MD5) default PostgreSQL untuk mengenkripsi kata sandi. Mekanisme autentikasi SCRAM dianggap lebih aman daripada MD5. Untuk mempelajari dua pendekatan berbeda guna mengamankan kata sandi ini selengkapnya, lihat Password Authentication dalam dokumentasi PostgreSQL.

Sebaiknya menggunakan SCRAM daripada MD5 sebagai skema enkripsi kata sandi untuk klaster DB Aurora PostgreSQL. Pada rilis Aurora PostgreSQL 14, SCRAM didukung di semua versi Aurora PostgreSQL yang tersedia, termasuk versi 10, 11, 12, 13, dan 14. Ini adalah mekanisme respons tantangan kriptografi yang menggunakan algoritma scram-sha-256 untuk autentikasi dan enkripsi kata sandi.

Anda mungkin perlu memperbarui pustaka untuk aplikasi klien Anda agar dapat mendukung SCRAM. Misalnya, versi JDBC sebelum 42.2.0 tidak mendukung SCRAM. Untuk informasi selengkapnya, lihat PostgreSQL JDBC Driver dalam dokumentasi Driver JDBC PostgreSQL. Untuk daftar driver PostgreSQL lainnya dan dukungan SCRAM, lihat List of drivers dalam dokumentasi PostgreSQL.

> **Note**
>
> Aurora PostgreSQL versi 14 dan yang lebih baru mendukung scram-sha-256 untuk enkripsi kata sandi klaster DB baru secara default. Artinya, grup parameter klaster DB default

(`default.aurora-postgresql14`) memiliki nilai `password_encryption` yang disetel ke scram-sha-256.

Menyiapkan klaster DB Aurora PostgreSQL agar meminta SCRAM

Untuk Aurora PostgreSQL 14.3 dan versi yang lebih baru, Anda dapat meminta klaster DB Aurora PostgreSQL untuk hanya menerima kata sandi yang menggunakan algoritma scram-sha-256.

Important

Untuk Proksi RDS yang ada dengan basis data PostgreSQL, jika Anda mengubah autentikasi basis data untuk menggunakan `SCRAM` saja, proksi akan menjadi tidak tersedia selama maksimal 60 detik. Untuk menghindari masalah ini, lakukan salah satu hal berikut:

- Pastikan basis data memungkinkan autentikasi `SCRAM` dan `MD5`.
- Untuk hanya menggunakan autentikasi `SCRAM`, buat proksi baru, migrasi lalu lintas aplikasi Anda ke proksi baru, lalu hapus proksi yang sebelumnya terkait dengan basis data.

Sebelum melakukan perubahan pada sistem, pastikan Anda telah memahami proses lengkapnya sebagai berikut:

- Dapatkan informasi semua peran dan enkripsi kata sandi untuk semua pengguna basis data.
- Periksa kembali pengaturan parameter klaster DB Aurora PostgreSQL untuk parameter yang mengontrol enkripsi kata sandi.
- Jika klaster DB Aurora PostgreSQL menggunakan grup parameter default, Anda harus membuat grup parameter klaster DB khusus lalu mengaplikasikannya ke klaster DB Aurora PostgreSQL agar Anda dapat memodifikasi parameter saat saat diperlukan. Jika klaster DB Aurora PostgreSQL menggunakan grup parameter khusus, Anda dapat memodifikasi parameter yang diperlukan nanti sesuai kebutuhan saat proses berlangsung.
- Ubah parameter `password_encryption` ke `scram-sha-256`.
- Beri tahu semua pengguna basis data bahwa mereka perlu memperbarui kata sandi. Lakukan hal yang sama untuk akun `postgres` Anda. Kata sandi baru dienkripsi dan disimpan menggunakan algoritma scram-sha-256.
- Verifikasi bahwa semua kata sandi dienkripsi menggunakan jenis enkripsi.

- Jika semua kata sandi menggunakan scram-sha-256, Anda dapat mengubah parameter `rds.accepted_password_auth_method` dari `md5+scram` ke `scram-sha-256`.

> **Warning**
>
> Setelah Anda mengubah `rds.accepted_password_auth_method` ke scram-sha-256, setiap pengguna (peran) dengan kata sandi terenkripsi `md5` tidak dapat terhubung.

Penyiapan meminta SCRAM untuk klaster DB Aurora PostgreSQL

Sebelum membuat perubahan apa pun pada klaster DB Aurora PostgreSQL periksa semua akun pengguna basis data yang ada. Periksa juga jenis enkripsi yang digunakan untuk kata sandi. Anda dapat melakukan tugas-tugas ini menggunakan ekstensi `rds_tools`. Ekstensi ini didukung di Aurora PostgreSQL 13.1 dan rilis yang lebih baru.

Untuk mendapatkan daftar pengguna basis data (peran) dan metode enkripsi kata sandi

1. Gunakan `psql` untuk terhubung ke instans utama klaster DB Aurora PostgreSQL , seperti yang ditunjukkan di bawah ini.

   ```
   psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

2. Instal ekstensi `rds_tools`.

   ```
   postgres=> CREATE EXTENSION rds_tools;
   CREATE EXTENSION
   ```

3. Dapatkan daftar peran dan enkripsi.

   ```
   postgres=> SELECT * FROM
         rds_tools.role_password_encryption_type();
   ```

   Anda akan melihat output yang mirip dengan berikut ini.

   ```
          rolname        | encryption_type
   ----------------------+-----------------
    pg_monitor           |
    pg_read_all_settings |
   ```

```
 pg_read_all_stats    |
 pg_stat_scan_tables  |
 pg_signal_backend    |
 lab_tester           | md5
 user_465             | md5
 postgres             | md5
(8 rows)
```

Membuat grup parameter klaster DB khusus

> **Note**
>
> Jika klaster DB Aurora PostgreSQL sudah menggunakan grup parameter khusus, Anda tidak perlu membuat grup parameter yang baru.

Untuk ringkasan grup parameter Aurora, lihat Membuat grup parameter klaster DB.

Jenis enkripsi kata sandi yang digunakan untuk kata sandi diatur dalam satu parameter, `password_encryption`. Enkripsi yang diizinkan klaster DB Aurora PostgreSQL diatur dalam parameter lain, `rds.accepted_password_auth_method`. Mengubah salah satu dari nilai default ini mengharuskan Anda membuat grup parameter klaster DB khusus dan mengaplikasikannya ke klaster.

Anda juga dapat menggunakan AWS Management Console atau RDS API untuk membuat grup parameter cluster DB kustom grup parameter . Untuk informasi selengkapnya, lihat Membuat grup parameter klaster DB.

Anda kini dapat mengaitkan grup parameter khusus dengan instans DB.

Untuk membuat grup parameter klaster DB khusus

1. Gunakan perintah CLI `create-db-cluster-parameter-group` untuk membuat grup parameter khusus klaster. Contoh berikut menggunakan `aurora-postgresql13` sebagai sumber untuk grup parameter khusus ini.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-lab-scram-passwords' \
   ```

```
  --db-parameter-group-family aurora-postgresql13  --description 'Custom DB cluster
 parameter group for SCRAM'
```

Untuk Windows:

```
aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-
lab-scram-passwords" ^
  --db-parameter-group-family aurora-postgresql13  --description "Custom DB cluster
 parameter group for SCRAM"
```

Anda kini dapat mengaitkan grup parameter khusus dengan klaster.

2. Gunakan perintah CLI `modify-db-cluster` untuk menerapkan grup parameter khusus ini ke klaster DB Aurora PostgreSQL.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds modify-db-cluster --db-cluster-identifier 'your-instance-name' \
           --db-cluster-parameter-group-name "docs-lab-scram-passwords
   ```

   Untuk Windows:

   ```
   aws rds modify-db-cluster --db-cluster-identifier "your-instance-name" ^
           --db-cluster-parameter-group-name "docs-lab-scram-passwords
   ```

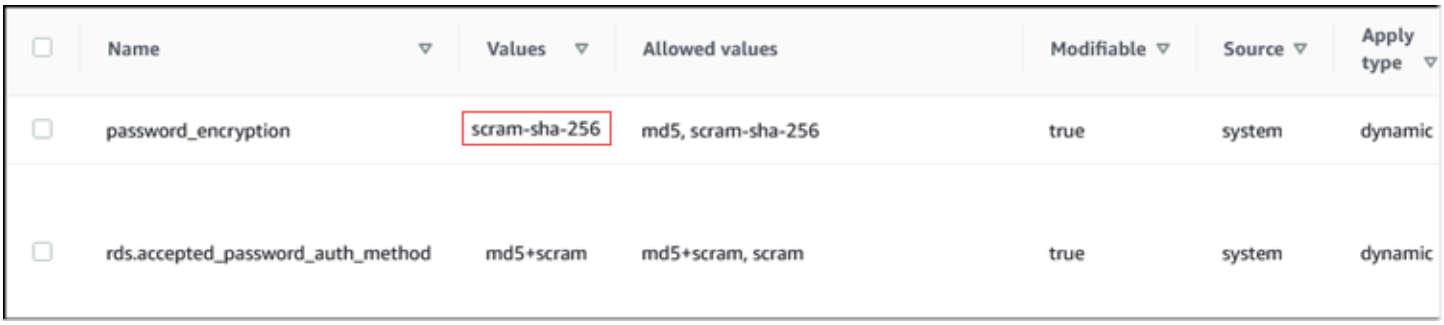
   Untuk menyinkronkan ulang klaster DB Aurora PostgreSQL Anda dengan grup parameter klaster DB khusus, boot ulang instans utama dan semua instans lain klaster.

Mengonfigurasi enkripsi kata sandi agar menggunakan SCRAM

Mekanisme enkripsi kata sandi yang digunakan oleh klaster DB Aurora PostgreSQL diatur dalam grup parameter klaster DB di parameter `password_encryption`. Nilai yang diizinkan tidak disetel, `md5`, atau `scram-sha-256`. Nilai default bergantung pada versi Aurora PostgreSQL , sebagai berikut:

- Aurora PostgreSQL 14 – Default adalah `scram-sha-256`
- Aurora PostgreSQL 13 – Default adalah `md5`

Dengan grup parameter klaster DB khusus yang dilampirkan ke klaster DB Aurora PostgreSQL, Anda dapat memodifikasi nilai untuk parameter enkripsi kata sandi.



Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

Untuk mengubah pengaturan enkripsi kata sandi menjadi scram-sha-256

- Ubah nilai enkripsi kata sandi menjadi scram-sha-256, seperti yang ditunjukkan berikut. Perubahan dapat diterapkan segera karena parameter bersifat dinamis sehingga tidak perlu memulai ulang agar perubahan diterapkan.

  Untuk Linux, macOS, atau Unix:

  ```
  aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name \
    'docs-lab-scram-passwords' --parameters
   'ParameterName=password_encryption,ParameterValue=scram-
  sha-256,ApplyMethod=immediate'
  ```

  Untuk Windows:

  ```
  aws rds modify-db-parameter-group --db-parameter-group-name ^
    "docs-lab-scram-passwords" --parameters
   "ParameterName=password_encryption,ParameterValue=scram-
  sha-256,ApplyMethod=immediate"
  ```

Memigrasikan kata sandi untuk peran pengguna ke SCRAM

Anda dapat memigrasikan kata sandi untuk peran pengguna ke SCRAM seperti yang dijelaskan berikut.

Untuk memigrasikan kata sandi pengguna (peran) basis data dari MD5 ke SCRAM

1. Masuk sebagai pengguna administrator (nama pengguna default, `postgres`) seperti yang ditunjukkan berikut.

```
psql --host=cluster-name-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password
```

2. Periksa pengaturan parameter `password_encryption` pada instans DB RDS for PostgreSQL menggunakan perintah berikut.

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
 md5
 (1 row)
```

3. Ubah nilai parameter ini menjadi scram-sha-256. Ini adalah parameter dinamis sehingga Anda tidak perlu melakukan boot ulang instans setelah melakukan perubahan ini. Periksa kembali nilainya untuk memastikan parameter sekarang telah diatur ke `scram-sha-256`, sebagai berikut.

```
postgres=> SHOW password_encryption;
 password_encryption
---------------------
 scram-sha-256
 (1 row)
```

4. Beri tahu semua pengguna basis data untuk mengubah kata sandi mereka. Pastikan juga untuk mengubah kata sandi Anda sendiri untuk akun `postgres` (pengguna basis data dengan hak istimewa `rds_superuser`).

```
labdb=> ALTER ROLE postgres WITH LOGIN PASSWORD 'change_me';
ALTER ROLE
```

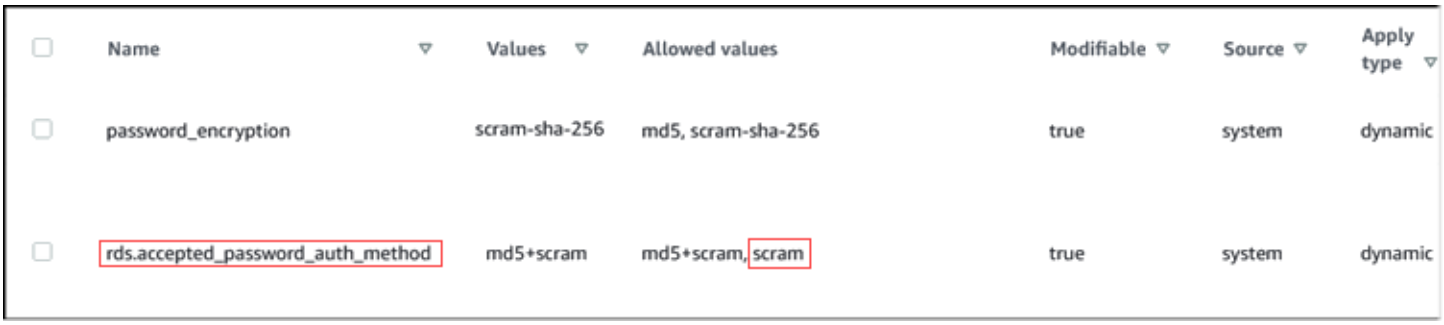
5. Ulangi proses untuk semua basis data di klaster DB Aurora PostgreSQL Anda.

Mengubah parameter agar memerlukan SCRAM

Ini adalah langkah terakhir dalam prosesnya. Setelah Anda membuat perubahan dalam prosedur berikut, setiap akun pengguna (peran) yang masih menggunakan enkripsi `md5` untuk kata sandi tidak dapat masuk ke klaster DB Aurora PostgreSQL.

`rds.accepted_password_auth_method` menentukan metode enkripsi yang diterima klaster DB Aurora PostgreSQL untuk kata sandi pengguna selama proses login. Nilai default-nya adalah

`md5+scram`, yang berarti bahwa salah satu metode diterima. Pada gambar berikut, Anda dapat menemukan pengaturan default untuk parameter ini.



Name	Values	Allowed values	Modifiable	Source	Apply type
password_encryption	scram-sha-256	md5, scram-sha-256	true	system	dynamic
rds.accepted_password_auth_method	md5+scram	md5+scram, scram	true	system	dynamic

Nilai yang diizinkan untuk parameter ini adalah `md5+scram` atau `scram`. Mengubah nilai parameter ini ke `scram` akan menjadikannya sebagai persyaratan.

Untuk mengubah nilai parameter agar memerlukan autentikasi SCRAM untuk kata sandi

1. Verifikasi bahwa semua kata sandi pengguna basis data untuk semua basis data di klaster DB Aurora PostgreSQL menggunakan `scram-sha-256` untuk enkripsi kata sandi. Untuk melakukannya, kueri `rds_tools` untuk peran (pengguna) dan jenis enkripsi, sebagai berikut.

```
postgres=> SELECT * FROM rds_tools.role_password_encryption_type();
  rolname        | encryption_type
  ---------------------+-----------------
  pg_monitor           |
  pg_read_all_settings |
  pg_read_all_stats    |
  pg_stat_scan_tables  |
  pg_signal_backend    |
  lab_tester           | scram-sha-256
  user_465             | scram-sha-256
  postgres             | scram-sha-256
  ( rows)
```

2. Ulangi kueri di semua instans DB di klaster DB Aurora PostgreSQL.

   Jika semua kata sandi menggunakan scram-sha-256, Anda dapat melanjutkan.

3. Ubah nilai autentikasi kata sandi yang diterima menjadi scram-sha-256, sebagai berikut.

   Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name 'docs-
lab-scram-passwords' \
```

```
  --parameters
 'ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name "docs-
lab-scram-passwords" ^
  --parameters
 "ParameterName=rds.accepted_password_auth_method,ParameterValue=scram,ApplyMethod=immediat
```

Mengamankan data Aurora PostgreSQL dengan SSL/TLS

Amazon RDS mendukung enkripsi Lapisan Soket Aman (SSL) dan Keamanan Lapisan Pengangkutan (TLS) untuk klaster DB Aurora PostgreSQL. Dengan SSL/TLS, Anda dapat mengenkripsi koneksi antara aplikasi dan klaster DB Aurora PostgreSQL. Anda juga dapat memaksa semua koneksi ke klaster DB Aurora PostgreSQL Anda untuk menggunakan SSL/TLS. Amazon Aurora PostgreSQL mendukung Keamanan Lapisan Pengangkutan (TLS) versi 1.1 dan 1.2. Kami merekomendasikan menggunakan TLS 1.2 untuk koneksi terenkripsi. Kami telah menambahkan dukungan untuk TLSv1.3 dari versi Aurora PostgreSQL berikut:

- Versi 15.3 dan semua yang lebih tinggi
- Versi 14.8 dan versi 14 yang lebih tinggi
- Versi 13.11 dan versi 13 yang lebih tinggi
- Versi 12.15 dan versi 12 yang lebih tinggi
- Versi 11.20 dan versi 11 yang lebih tinggi

Untuk informasi umum tentang dukungan SSL/TLS dan basis data PostgreSQL, lihat Dukungan SSL dalam dokumentasi PostgreSQL. Untuk informasi tentang menggunakan koneksi SSL/TLS melalui JDBC, lihat Mengonfigurasi klien dalam dokumentasi PostgreSQL.

Topik

- Memerlukan klaster DB Aurora PostgreSQL melalui SSL/TLS
- Menentukan status koneksi SSL/TLS
- Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora PostgreSQL

Dukungan SSL/TLS tersedia di semua Wilayah AWS untuk Aurora PostgreSQL. Amazon RDS membuat sertifikat SSL/TLS untuk klaster DB Aurora PostgreSQL Anda saat klaster DB dibuat. Jika Anda mengaktifkan verifikasi sertifikat SSL/TLS, maka sertifikat SSL/TLS mencakup titik akhir klaster DB sebagai Nama Umum (CN) untuk sertifikat SSL/TLS untuk melindungi dari serangan spoofing.

Untuk menghubungkan ke klaster DB Aurora PostgreSQL melalui SSL/TLS

1. Unduh sertifikatnya.

   Untuk informasi tentang mengunduh sertifikat, lihat .
2. Impor sertifikat ke dalam sistem operasi Anda.
3. Hubungkan ke klaster DB Aurora PostgreSQL melalui SSL/TLS.

   Saat Anda menghubungkan menggunakan SSL/TLS, klien Anda dapat memilih untuk memverifikasi rantai sertifikat atau tidak. Jika parameter koneksi Anda menentukan `sslmode=verify-ca` atau `sslmode=verify-full`, maka klien Anda mengharuskan sertifikat RDS CA berada di penyimpanan kepercayaan atau direferensikan di URL koneksi. Persyaratan ini untuk memverifikasi rantai sertifikat yang menandatangani sertifikat basis data Anda.

   Ketika klien, seperti psql atau JDBC, dikonfigurasikan dengan dukungan SSL/TLS, klien mencoba untuk terhubung ke basis data dengan SSL/TLS secara default terlebih dahulu. Jika klien tidak dapat terhubung dengan SSL/TLS, maka akan kembali menghubungkan tanpa SSL/TLS. Secara default, opsi `sslmode` untuk klien berbasis JDBC dan libpq diatur ke `prefer`.

   Gunakan parameter `sslrootcert` untuk mereferensikan sertifikat, misalnya `sslrootcert=rds-ssl-ca-cert.pem`.

Berikut ini adalah contoh penggunaan psql untuk terhubung ke klaster DB Aurora PostgreSQL.

```
$ psql -h testpg.cdhmuqifdpib.us-east-1.rds.amazonaws.com -p 5432 \
    "dbname=testpg user=testuser sslrootcert=rds-ca-2015-root.pem sslmode=verify-full"
```

Memerlukan klaster DB Aurora PostgreSQL melalui SSL/TLS

Anda dapat meminta koneksi tersebut ke klaster DB Aurora PostgreSQL Anda dengan SSL/TLS menggunakan parameter `rds.force_ssl`. Secara default, parameter `rds.force_ssl` diatur ke 0 (nonaktif). Anda dapat mengatur parameter `rds.force_ssl` ke 1 (aktif) untuk meminta SSL/TLS koneksi ke klaster DB Anda. Memperbarui parameter `rds.force_ssl` juga mengatur parameter

`ssl` PostgreSQL ke 1 (aktif) dan mengubah file `pg_hba.conf` klaster DB Anda untuk mendukung konfigurasi SSL/TLS baru.

Anda dapat mengatur nilai parameter `rds.force_ssl` dengan memperbarui grup parameter klaster DB untuk klaster DB Anda. Jika grup parameter klaster DB bukan default, dan parameter `ssl` sudah diatur ke 1 saat Anda mengatur `rds.force_ssl` ke 1, Anda tidak perlu melakukan boot ulang klaster DB Anda. Jika tidak, Anda harus melakukan boot ulang klaster DB Anda agar perubahan dapat diterapkan. Untuk informasi selengkapnya tentang grup parameter, lihat Bekerja dengan grup parameter.

Saat `rds.force_ssl` parameter diatur ke 1 untuk klaster DB, Anda melihat output yang serupa dengan yang berikut ini saat Anda terhubung, menunjukkan bahwa SSL/TLS sekarang diperlukan:

```
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql (9.3.12, server 9.4.4)
WARNING: psql major version 9.3, server major version 9.4.
Some psql features might not work.
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

Menentukan status koneksi SSL/TLS

Status terenkripsi dari koneksi Anda ditunjukkan pada banner logon saat Anda terhubung ke klaster DB.

```
Password for user master:
psql (9.3.12)
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
Type "help" for help.

postgres=>
```

Anda juga dapat memuat ekstensi `sslinfo`, lalu memanggil fungsi `ssl_is_used()` untuk menentukan apakah SSL/TLS digunakan. Fungsi akan kembali `t` jika koneksi menggunakan SSL/TLS, jika tidak akan menghasilkan `f`.

```
postgres=> create extension sslinfo;
CREATE EXTENSION

postgres=> select ssl_is_used();
 ssl_is_used
---------
t
(1 row)
```

Anda dapat menggunakan perintah `select ssl_cipher()` untuk menentukan cipher SSL/TLS:

```
postgres=> select ssl_cipher();
ssl_cipher
--------------------
DHE-RSA-AES256-SHA
(1 row)
```

Jika Anda mengaktifkan `set rds.force_ssl` dan memulai ulang klaster DB, koneksi non-SSL ditolak dengan pesan berikut ini:

```
$ export PGSSLMODE=disable
$ psql postgres -h SOMEHOST.amazonaws.com -p 8192 -U someuser
psql: FATAL: no pg_hba.conf entry for host "host.ip", user "someuser", database
 "postgres", SSL off
$
```

Untuk informasi tentang opsi `sslmode`, lihat Fungsi kontrol koneksi basis data dalam dokumentasi PostgreSQL.

Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora PostgreSQL

Dengan menggunakan cipher suite yang dapat dikonfigurasi, Anda dapat memiliki kontrol lebih besar atas keamanan koneksi basis data. Anda dapat menentukan daftar cipher suite yang ingin Anda izinkan untuk mengamankan koneksi TLS/SSL klien ke basis data Anda. Dengan cipher suite yang dapat dikonfigurasi, Anda dapat mengontrol enkripsi koneksi yang diterima server basis data Anda. Melakukan hal ini membantu mencegah penggunaan cipher yang tidak aman atau usang.

Suite cipher yang dapat dikonfigurasi didukung di Aurora PostgreSQL versi 11.8 dan lebih tinggi.

Untuk menentukan daftar cipher yang diizinkan untuk mengenkripsi koneksi, modifikasi parameter klaster `ssl_ciphers`. Atur parameter `ssl_ciphers` ke string nilai cipher yang dipisahkan koma dalam grup parameter klaster menggunakan AWS Management Console, AWS CLI, atau RDS API. Untuk mengatur parameter klaster, lihat Mengubah parameter dalam grup parameter klaster DB.

Tabel berikut menunjukkan cipher yang didukung untuk versi mesin Aurora PostgreSQL yang valid.

Versi mesin Aurora PostgreSQL	Cipher yang didukung
9.6, 10.20 dan lebih rendah, 11.15 dan lebih rendah, 12.10 dan lebih rendah, 13.6 dan lebih rendah	• DHE-RSA-AES128-SHA • DHE-RSA-AES128-SHA256 • DHE-RSA-AES128-GCM-SHA256 • DHE-RSA-AES256-SHA • DHE-RSA-AES256-SHA256 • DHE-RSA-AES256-GCM-SHA384 • ECDHE-ECDSA-AES256-SHA • ECDHE-ECDSA-AES256-GCM-SHA384 • ECDHE-RSA-AES256-SHA384 • ECDHE-RSA-AES128-SHA • ECDHE-RSA-AES128-SHA256 • ECDHE-RSA-AES128-GCM-SHA256 • ECDHE-RSA-AES256-SHA • ECDHE-RSA-AES256-GCM-SHA384
10.21, 11.16, 12.11, 13.7, 14.3, dan 14.4	• DHE-RSA-AES128-SHA • DHE-RSA-AES128-SHA256 • DHE-RSA-AES128-GCM-SHA256 • DHE-RSA-AES256-SHA • DHE-RSA-AES256-SHA256 • DHE-RSA-AES256-GCM-SHA384 • ECDHE-ECDSA-AES256-SHA

Versi mesin Aurora PostgreSQL	Cipher yang didukung
	• ECDHE-ECDSA-AES256-GCM-SHA384 • ECDHE-RSA-AES256-SHA384 • ECDHE-RSA-AES128-SHA • ECDHE-RSA-AES128-GCM-SHA256 • ECDHE-RSA-AES256-SHA • ECDHE-RSA-AES256-GCM-SHA384 • TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA • TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 • TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA • TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 • TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA • TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 • TLS_RSA_WITH_AES_256_GCM_SHA384 • TLS_RSA_WITH_AES_256_CBC_SHA • TLS_RSA_WITH_AES_128_GCM_SHA256 • TLS_RSA_WITH_AES_128_CBC_SHA • TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256

Versi mesin Aurora PostgreSQL	Cipher yang didukung
10.22 dan lebih tinggi, 11.17 dan lebih tinggi, 12.12 dan lebih tinggi, 13.8 dan lebih tinggi, 14.5 dan lebih tinggi, serta 15.2 dan lebih tinggi	• DHE-RSA-AES128-SHA • DHE-RSA-AES128-SHA256 • DHE-RSA-AES128-GCM-SHA256 • DHE-RSA-AES256-SHA • DHE-RSA-AES256-SHA256 • DHE-RSA-AES256-GCM-SHA384 • ECDHE-ECDSA-AES256-SHA • ECDHE-ECDSA-AES256-GCM-SHA384 • ECDHE-RSA-AES256-SHA384 • ECDHE-RSA-AES128-SHA • ECDHE-RSA-AES128-SHA256 • ECDHE-RSA-AES128-GCM-SHA256 • ECDHE-RSA-AES256-SHA • ECDHE-RSA-AES256-GCM-SHA384 • TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA • TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 • TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA • TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 • TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 • TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

Versi mesin Aurora PostgreSQL	Cipher yang didukung
	• TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 • TLS_RSA_WITH_AES_256_GCM_SHA384 • TLS_RSA_WITH_AES_256_CBC_SHA • TLS_RSA_WITH_AES_128_GCM_SHA256 • TLS_RSA_WITH_AES_128_CBC_SHA256 • TLS_RSA_WITH_AES_128_CBC_SHA • TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256

Versi mesin Aurora PostgreSQL	Cipher yang didukung
15.3, 14.8, 13.11, 12.15, dan 11.20	• DHE-RSA-AES128-SHA • DHE-RSA-AES128-SHA256 • DHE-RSA-AES128-GCM-SHA256 • DHE-RSA-AES256-SHA • DHE-RSA-AES256-SHA256 • DHE-RSA-AES256-GCM-SHA384 • ECDHE-ECDSA-AES256-SHA • ECDHE-ECDSA-AES256-GCM-SHA384 • ECDHE-RSA-AES256-SHA384 • ECDHE-RSA-AES128-SHA • ECDHE-RSA-AES128-SHA256 • ECDHE-RSA-AES128-GCM-SHA256 • ECDHE-RSA-AES256-SHA • ECDHE-RSA-AES256-GCM-SHA384 • TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA • TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 • TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA • TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 • TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 • TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

Versi mesin Aurora PostgreSQL	Cipher yang didukung
	• TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 • TLS_RSA_WITH_AES_256_GCM_SHA384 • TLS_RSA_WITH_AES_256_CBC_SHA • TLS_RSA_WITH_AES_128_GCM_SHA256 • TLS_RSA_WITH_AES_128_CBC_SHA256 • TLS_RSA_WITH_AES_128_CBC_SHA • TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 • TLS_AES_128_GCM_SHA256 • TLS_AES_256_GCM_SHA384

Anda juga dapat menggunakan perintah CLI ddescribe-engine-default-cluster-parameters untuk menentukan cipher suite mana yang saat ini didukung untuk keluarga grup parameter tertentu. Contoh berikut menunjukkan cara mendapatkan nilai yang diizinkan untuk parameter klaster `ssl_cipher` untuk Aurora PostgreSQL 11.

```
aws rds describe-engine-default-cluster-parameters --db-parameter-group-family aurora-
postgresql11

    ...some output truncated...
 {
  "ParameterName": "ssl_ciphers",
  "Description": "Sets the list of allowed TLS ciphers to be used on secure
 connections.",
  "Source": "engine-default",
  "ApplyType": "dynamic",
  "DataType": "list",
  "AllowedValues": "DHE-RSA-AES128-SHA,DHE-RSA-AES128-SHA256,DHE-RSA-AES128-GCM-
SHA256,DHE-RSA-AES256-SHA,DHE-RSA-AES256-SHA256,DHE-RSA-AES256-GCM-SHA384,
```

```
  ECDHE-RSA-AES128-SHA,ECDHE-RSA-AES128-SHA256,ECDHE-RSA-AES128-GCM-
SHA256,ECDHE-RSA-AES256-SHA,ECDHE-RSA-AES256-SHA384,ECDHE-RSA-AES256-GCM-
SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384,

 TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_128_GCM_SHA256,TLS_RSA_WITH_AES_128_CBC_SHA256,T

 TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_AE
  "IsModifiable": true,
  "MinimumEngineVersion": "11.8",
  "SupportedEngineModes": [
   "provisioned"
  ]
 },
    ...some output truncated...
```

Parameter default `ssl_ciphers` ke semua suite cipher yang diizinkan. Untuk informasi selengkapnya tentang cipher, lihat variabel ssl_ciphers dalam dokumentasi PostgreSQL.

Memperbarui aplikasi untuk terhubung ke klaster DB Aurora PostgreSQL menggunakan sertifikat SSL/TLS baru

Sejak 13 Januari 2023, Amazon RDS telah menerbitkan serifikat Otoritas Sertifikat (CA) baru untuk terhubung ke klaster DB Aurora menggunakan Lapisan Soket Aman atau Keamanan Lapisan Pengangkutan (SSL/TLS). Setelah itu, Anda dapat menemukan informasi tentang pembaruan aplikasi untuk menggunakan sertifikat baru.

Topik ini dapat membantu Anda menentukan apakah aplikasi klien menggunakan SSL/TLS untuk terhubung ke klaster DB Anda. Jika demikian, Anda dapat memeriksa lebih lanjut apakah aplikasi tersebut memerlukan verifikasi sertifikat untuk terhubung.

> **Note**
>
> Beberapa aplikasi dikonfigurasi untuk terhubung ke klaster DB Aurora PostgreSQL hanya jika aplikasi tersebut berhasil memverifikasi sertifikat pada server.
> Untuk aplikasi tersebut, Anda harus memperbarui penyimpanan kepercayaan aplikasi klien untuk menyertakan sertifikat CA baru.

Setelah memperbarui sertifikat CA di penyimpanan kepercayaan aplikasi klien, Anda dapat merotasi sertifikat di klaster DB Anda. Sebaiknya Anda menguji prosedur ini di lingkungan pengembangan dan penahapan sebelum menerapkannya di lingkungan produksi Anda.

Untuk informasi selengkapnya tentang rotasi sertifikat, lihat Merotasi sertifikat SSL/TLS. Untuk informasi selengkapnya tentang cara mengunduh sertifikat, lihat . Untuk informasi tentang menggunakan SSL/TLS dengan klaster DB PostgreSQL, lihat Mengamankan data Aurora PostgreSQL dengan SSL/TLS.

Topik

- Menentukan apakah aplikasi terhubung ke klaster DB Aurora PostgreSQL menggunakan SSL
- Menentukan apakah klien memerlukan verifikasi sertifikat agar dapat terhubung
- Memperbarui penyimpanan kepercayaan aplikasi Anda
- Menggunakan koneksi SSL/TLS untuk berbagai jenis aplikasi

Menentukan apakah aplikasi terhubung ke klaster DB Aurora PostgreSQL menggunakan SSL

Periksa konfigurasi klaster DB untuk nilai parameter `rds.force_ssl`. Secara default, parameter `rds.force_ssl` diatur ke `0` (nonaktif). Jika parameter `rds.force_ssl` diatur ke 1 (aktif), klien harus menggunakan SSL/TLS untuk koneksi. Untuk informasi selengkapnya tentang grup parameter, lihat Bekerja dengan grup parameter.

Jika `rds.force_ssl` tidak diatur ke 1 (aktif), buat kueri ke tampilan `pg_stat_ssl` untuk memeriksa koneksi menggunakan SSL. Misalnya, kueri berikut hanya menampilkan koneksi SSL dan informasi tentang klien menggunakan SSL.

```
select datname, usename, ssl, client_addr from pg_stat_ssl inner join pg_stat_activity
 on pg_stat_ssl.pid = pg_stat_activity.pid where ssl is true and usename<>'rdsadmin';
```

Hanya baris yang menggunakan koneksi SSL/TLS yang ditampilkan dengan informasi mengenai koneksi. Berikut ini adalah contoh outputnya.

```
 datname  | usename | ssl | client_addr
----------+---------+-----+-------------
 benchdb  | pgadmin | t   | 53.95.6.13
 postgres | pgadmin | t   | 53.95.6.13
```

```
(2 rows)
```

Kueri sebelumnya hanya menampilkan koneksi saat ini pada waktu kueri. Tidak adanya hasil tidak menunjukkan bahwa tidak ada aplikasi yang menggunakan koneksi SSL. Koneksi SSL lainnya mungkin terhubung pada waktu yang lain.

Menentukan apakah klien memerlukan verifikasi sertifikat agar dapat terhubung

Ketika klien, seperti psql atau JDBC, dikonfigurasikan dengan dukungan SSL, klien pertama kali mencoba untuk terhubung ke basis data dengan SSL secara default. Jika klien tidak dapat terhubung dengan SSL, maka akan kembali ke koneksi tanpa SSL. Mode `sslmode` default yang digunakan berbeda antara klien berbasis libpq (seperti psql) dan JDBC. Klien berbasis libpq secara default diatur ke `prefer`, di mana klien JDBC secara default diatur ke `verify-full`. Sertifikat di server diverifikasi hanya ketika `sslrootcert` disediakan dengan `sslmode` diatur ke `verify-ca` atau `verify-full`. Kesalahan akan muncul jika sertifikat tidak valid.

Gunakan `PGSSLROOTCERT` untuk memverifikasi sertifikat dengan variabel lingkungan `PGSSLMODE`, dengan `PGSSLMODE` diatur ke `verify-ca` atau `verify-full`.

```
PGSSLMODE=verify-full PGSSLROOTCERT=/fullpath/ssl-cert.pem psql -h
 pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com -U primaryuser -d postgres
```

Gunakan argumen `sslrootcert` untuk memverifikasi sertifikat dengan `sslmode` dalam format string koneksi, dengan `sslmode` diatur ke `verify-ca` atau `verify-full`.

```
psql "host=pgdbidentifier.cxxxxxxxx.us-east-2.rds.amazonaws.com sslmode=verify-full
 sslrootcert=/full/path/ssl-cert.pem user=primaryuser dbname=postgres"
```

Misalnya, dalam kasus sebelumnya, jika Anda menggunakan sertifikat root yang tidak valid, Anda akan melihat kesalahan yang serupa dengan yang berikut pada klien Anda.

```
psql: SSL error: certificate verify failed
```

Memperbarui penyimpanan kepercayaan aplikasi Anda

Untuk informasi tentang cara memperbarui penyimpanan kepercayaan untuk aplikasi PostgreSQL, lihat Mengamankan koneksi TCP/IP SSL dalam dokumentasi PostgreSQL.

> **Note**
>
> Saat memperbarui penyimpanan kepercayaan, Anda dapat mempertahankan sertifikat lama selain menambahkan sertifikat baru.

Memperbarui penyimpanan kepercayaan aplikasi Anda untuk JDBC

Anda dapat memperbarui penyimpanan kepercayaan untuk aplikasi yang menggunakan JDBC untuk koneksi SSL/TLS.

Untuk informasi tentang cara mengunduh sertifikat root, lihat .

Untuk contoh skrip yang mengimpor sertifikat, lihat Contoh skrip untuk mengimpor sertifikat ke trust store Anda.

Menggunakan koneksi SSL/TLS untuk berbagai jenis aplikasi

Berikut ini informasi tentang menggunakan koneksi SSL/TLS untuk berbagai jenis aplikasi:

- psql

  Klien diinvokasi dari baris perintah dengan menentukan opsi sebagai string koneksi atau sebagai variabel lingkungan. Untuk koneksi SSL/TLS, opsi yang relevan adalah `sslmode` (variabel lingkungan `PGSSLMODE`), `sslrootcert` (variabel lingkungan `PGSSLROOTCERT`).

  Untuk daftar lengkap opsi, lihat Kata kunci parameter dalam dokumentasi PostgreSQL. Untuk daftar lengkap variabel lingkungan, lihat Variabel lingkungan dalam dokumentasi PostgreSQL.

- pgAdmin

  Klien berbasis browser ini adalah antarmuka yang lebih mudah untuk terhubung ke basis data PostgreSQL.

  Untuk informasi tentang konfigurasi koneksi, lihat dokumentasi pgAdmin.

- JDBC

  JDBC memungkinkan koneksi basis data dengan aplikasi Java.

Untuk informasi umum tentang cara terhubung ke basis data PostgreSQL dengan JDBC, lihat Menghubungkan ke basis data dalam dokumentasi PostgreSQL. Untuk informasi tentang terhubung dengan SSL/TLS, lihat Mengonfigurasi klien dalam dokumentasi PostgreSQL.

- Python

  Perpustakaan Python yang populer untuk terhubung ke basis data PostgreSQL adalah `psycopg2`.

  Untuk informasi tentang menggunakan `psycopg2`, lihat Dokumentasi psycopg2. Untuk tutorial pendek tentang cara menghubungkan ke basis data PostgreSQL, lihat Tutorial Psycopg2. Anda dapat menemukan informasi tentang opsi penerimaan perintah koneksi di Konten modul psycopg2.

> **Important**
>
> Setelah Anda menentukan bahwa koneksi database Anda menggunakan SSL/TLS dan telah memperbarui toko kepercayaan aplikasi Anda, Anda dapat memperbarui database Anda untuk menggunakan sertifikat 2048-g1. rds-ca-rsa Untuk petunjuk, lihat langkah 3 dalam Memperbarui sertifikat CA Anda dengan memodifikasi instans cluster DB.

Menggunakan autentikasi Kerberos dengan Aurora PostgreSQL

Anda dapat menggunakan Kerberos untuk mengautentikasi pengguna saat terhubung ke klaster DB Anda yang menjalankan PostgreSQL. Untuk melakukannya, konfigurasi klaster DB Anda agar menggunakan AWS Directory Service for Microsoft Active Directory untuk autentikasi Kerberos. AWS Directory Service for Microsoft Active Directory juga disebut AWS Managed Microsoft AD. Ini adalah fitur yang tersedia dengan AWS Directory Service. Untuk mempelajari selengkapnya, lihat Apa itu AWS Directory Service? dalam Panduan Administrasi AWS Directory Service.

Untuk memulai, buat direktori AWS Managed Microsoft AD untuk menyimpan kredensial pengguna. Kemudian, berikan domain Active Directory dan informasi lainnya ke klaster DB PostgreSQL Anda. Saat pengguna mengautentikasi dengan klaster DB PostgreSQL, permintaan autentikasi diteruskan ke direktori AWS Managed Microsoft AD.

Dengan menyimpan semua kredensial Anda di direktori yang sama, Anda dapat menghemat waktu dan tenaga. Anda memiliki sebuah lokasi terpusat untuk menyimpan dan mengelola kredensial bagi beberapa klaster DB. Penggunaan direktori juga dapat meningkatkan profil keamanan Anda secara keseluruhan.

Selain itu, Anda dapat mengakses kredensial dari Microsoft Active Directory on-premise Anda sendiri. Untuk melakukannya, buat hubungan domain tepercaya sehingga direktori AWS Managed Microsoft AD mempercayai Microsoft Active Directory on-premise Anda. Dengan cara ini, pengguna Anda dapat mengakses klaster PostgreSQL Anda dengan pengalaman masuk tunggal (SSO) Windows yang sama seperti ketika mereka mengakses beban kerja di jaringan on-premise Anda.

Basis data dapat menggunakan Kerberos, AWS Identity and Access Management (IAM), atau autentikasi Kerberos dan IAM. Namun, karena autentikasi Kerberos dan IAM menyediakan metode autentikasi yang berbeda, pengguna basis data tertentu dapat masuk ke basis data hanya menggunakan salah satu metode autentikasi, dan tidak bisa keduanya. Untuk informasi selengkapnya tentang autentikasi IAM, lihat Autentikasi basis data IAM.

Topik

- Ketersediaan Wilayah dan versi
- Ikhtisar autentikasi Kerberos untuk klaster DB PostgreSQL
- Menyiapkan autentikasi Kerberos untuk klaster DB PostgreSQL
- Mengelola klaster DB di Domain
- Menghubungkan ke PostgreSQL dengan autentikasi Kerberos
- Menggunakan grup keamanan AD untuk kontrol akses Aurora PostgreSQL

Ketersediaan Wilayah dan versi

Ketersediaan dan dukungan fitur bervariasi di seluruh versi khusus dari setiap mesin basis data, dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang Ketersediaan wilayah dan versi Aurora PostgreSQL dengan autentikasi Kerberos, lihat Autentikasi Kerberos dengan Aurora PostgreSQL.

Ikhtisar autentikasi Kerberos untuk klaster DB PostgreSQL

Untuk menyiapkan autentikasi Kerberos untuk klaster DB PostgreSQL, lakukan langkah-langkah berikut, yang dijelaskan secara lebih mendetail nanti:

1. Gunakan AWS Managed Microsoft AD untuk membuat direktori AWS Managed Microsoft AD. Anda dapat menggunakan AWS Management Console, AWS CLI, atau AWS Directory Service API untuk membuat direktori. Pastikan untuk membuka port keluar yang relevan pada grup keamanan direktori sehingga direktori dapat berkomunikasi dengan klaster.

2. Buat peran yang memberikan akses ke Amazon Aurora untuk melakukan panggilan ke direktori AWS Managed Microsoft AD Anda. Untuk melakukannya, buat peran (IAM) AWS Identity and Access Management yang menggunakan kebijakan IAM terkelola `AmazonRDSDirectoryServiceAccess`.

   Agar peran IAM mengizinkan akses, titik akhir AWS Security Token Service (AWS STS) harus diaktifkan di Wilayah AWS yang tepat untuk akun AWS Anda. Titik akhir AWS STS aktif secara default di semua Wilayah AWS, dan Anda dapat menggunakannya tanpa tindakan lebih lanjut. Lihat informasi selengkapnya di Mengaktifkan dan menonaktifkan AWS STS di Wilayah AWS dalam Panduan Pengguna IAM.

3. Buat dan konfigurasikan pengguna dalam direktori AWS Managed Microsoft AD menggunakan alat Microsoft Active Directory. Untuk mengetahui informasi selengkapnya tentang cara membuat pengguna di Active Directory, lihat Mengelola pengguna dan grup di Microsoft AD Terkelola AWS dalam Panduan Administrasi AWS Directory Service.

4. Jika Anda ingin menemukan direktori dan instans DB dalam akun AWS atau cloud privat virtual (VPC) yang berbeda, konfigurasikan peering VPC. Untuk informasi selengkapnya, lihat Apa yang dimaksud peering VPC? di Panduan Peering Amazon VPC.

5. Buat atau modifikasi klaster DB PostgreSQL dari konsol, CLI, atau RDS API menggunakan salah satu metode berikut:

   - Membuat dan terhubung ke klaster DB Aurora PostgreSQL
   - Memodifikasi klaster DB Amazon Aurora
   - Memulihkan dari snapshot klaster DB
   - Memulihkan klaster DB ke waktu tertentu

   Anda dapat menemukan klaster di Cloud Privat Virtual (VPC) Amazon yang sama dengan direktori atau di VPC atau akun AWS yang berbeda. Saat membuat atau memodifikasi klaster DB PostgreSQL, lakukan hal berikut:

   - Sediakan pengidentifikasi domain (pengidentifikasi `d-*`) yang dihasilkan saat Anda membuat direktori.
   - Beri nama peran IAM yang Anda buat.
   - Pastikan bahwa grup keamanan instans DB dapat menerima lalu lintas masuk dari grup keamanan direktori.

6. Gunakan kredensial pengguna utama RDS untuk terhubung ke klaster DB PostgreSQL. Buat pengguna dalam PostgreSQL untuk diidentifikasi secara eksternal. Pengguna yang diidentifikasi secara eksternal dapat masuk ke klaster DB PostgreSQL menggunakan autentikasi Kerberos.

Menyiapkan autentikasi Kerberos untuk klaster DB PostgreSQL

Untuk menyiapkan autentikasi Kerberos, lakukan langkah berikut.

Topik

- Langkah 1: Buat direktori menggunakan AWS Managed Microsoft AD
- Langkah 2: (Opsional) Buat hubungan kepercayaan antara Active Directory lokal dan AWS Directory Service
- Langkah 3: Buat peran IAM untuk Amazon RDS untuk mengakses AWS Directory Service
- Langkah 4: Buat dan konfigurasikan pengguna
- Langkah 5: Aktifkan lalu lintas antar-VPC antara direktori dan instans DB
- Langkah 6: Buat atau modifikasi klaster DB PostgreSQL
- Langkah 7: Buat pengguna PostgreSQL untuk pengguna utama Kerberos Anda
- Langkah 8: Konfigurasikan klien PostgreSQL

Langkah 1: Buat direktori menggunakan AWS Managed Microsoft AD

AWS Directory Service membuat Direktori Aktif yang dikelola sepenuhnya di AWS Cloud. Saat Anda membuat AWS Managed Microsoft AD direktori, AWS Directory Service buat dua pengontrol domain dan server DNS untuk Anda. Server-server direktori dibuat di subnet yang berbeda di VPC. Redundansi ini membantu memastikan bahwa direktori Anda tetap dapat diakses meskipun terjadi kegagalan.

Saat Anda membuat AWS Managed Microsoft AD AWS direktori, Directory Service melakukan tugas-tugas berikut atas nama Anda:

- Menyiapkan Active Directory di dalam VPC Anda.
- Membuat akun administrator direktori dengan nama pengguna `Admin` dan kata sandi yang ditentukan. Anda menggunakan akun ini untuk mengelola direktori.

> **Important**
>
> Pastikan untuk menyimpan kata sandi ini. AWS Directory Service tidak menyimpan kata sandi ini, dan tidak dapat diambil atau diatur ulang.

- Membuat grup keamanan untuk pengendali direktori. Grup keamanan harus mengizinkan komunikasi dengan klaster DB PostgreSQL.

Saat Anda meluncurkan AWS Directory Service for Microsoft Active Directory, AWS buat Unit Organisasi (OU) yang berisi semua objek direktori Anda. OU ini memiliki nama NetBIOS yang Anda masukkan saat membuat direktori, dan terletak di root domain. Root domain dimiliki dan dikelola oleh AWS.

`Admin`Akun yang dibuat dengan AWS Managed Microsoft AD direktori Anda memiliki izin untuk kegiatan administratif yang paling umum untuk OU Anda:

- Membuat, memperbarui, atau menghapus pengguna
- Menambahkan sumber daya ke domain Anda seperti server file atau cetak, lalu menetapkan izin untuk sumber daya tersebut kepada pengguna di OU Anda
- Membuat OU dan kontainer tambahan
- Melimpahkan kewenangan
- Memulihkan objek-objek yang dihapus dari Keranjang Sampah Active Directory
- Jalankan modul Active Directory dan Domain Name Service (DNS) untuk Windows PowerShell pada Layanan Web Direktori Aktif

Akun `Admin` juga memiliki hak untuk melakukan aktivitas di seluruh domain berikut:

- Mengelola konfigurasi DNS (menambahkan, menghapus, atau memperbarui catatan, zona, dan penerus)
- Melihat log peristiwa DNS
- Melihat log peristiwa keamanan

Untuk membuat direktori dengan AWS Managed Microsoft AD

1. Di panel navigasi konsol AWS Directory Service, pilih Direktori, lalu pilih Siapkan direktori.
2. Pilih AWS Managed Microsoft AD. AWS Managed Microsoft AD adalah satu-satunya opsi yang saat ini didukung untuk digunakan dengan Amazon Aurora.
3. Pilih Berikutnya.
4. Di halaman Masukkan informasi direktori, berikan informasi berikut:

Edisi

Pilih edisi sesuai kebutuhan Anda.

Nama DNS direktori

Nama berkualifikasi penuh untuk direktori, seperti **`corp.example.com`**.

Nama NetBIOS direktori

Nama pendek opsional untuk direktori, seperti `CORP`.

Deskripsi direktori

Deskripsi opsional untuk direktori.

Kata sandi admin

Kata sandi administrator direktori. Proses pembuatan direktori menciptakan akun administrator dengan nama pengguna `Admin` dan kata sandi ini.

Kata sandi administrator direktori tidak boleh menyertakan kata "admin". Kata sandi peka terhadap huruf besar/kecil dan harus terdiri dari 8–64 karakter. Kata sandi juga harus berisi setidaknya satu karakter dari tiga di antara empat kategori berikut:

- Huruf kecil (a-z)
- Huruf besar (A-Z)
- Angka (0–9)
- Karakter non-alfanumerik (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/)

Konfirmasi kata sandi

Ketik ulang kata sandi administrator.

> **Important**
>
> Pastikan Anda menyimpan kata sandi ini. AWS Directory Service tidak menyimpan kata sandi ini, dan tidak dapat diambil atau diatur ulang.

5. Pilih Berikutnya.
6. Di halaman Pilih VPC dan subnet, berikan informasi berikut:

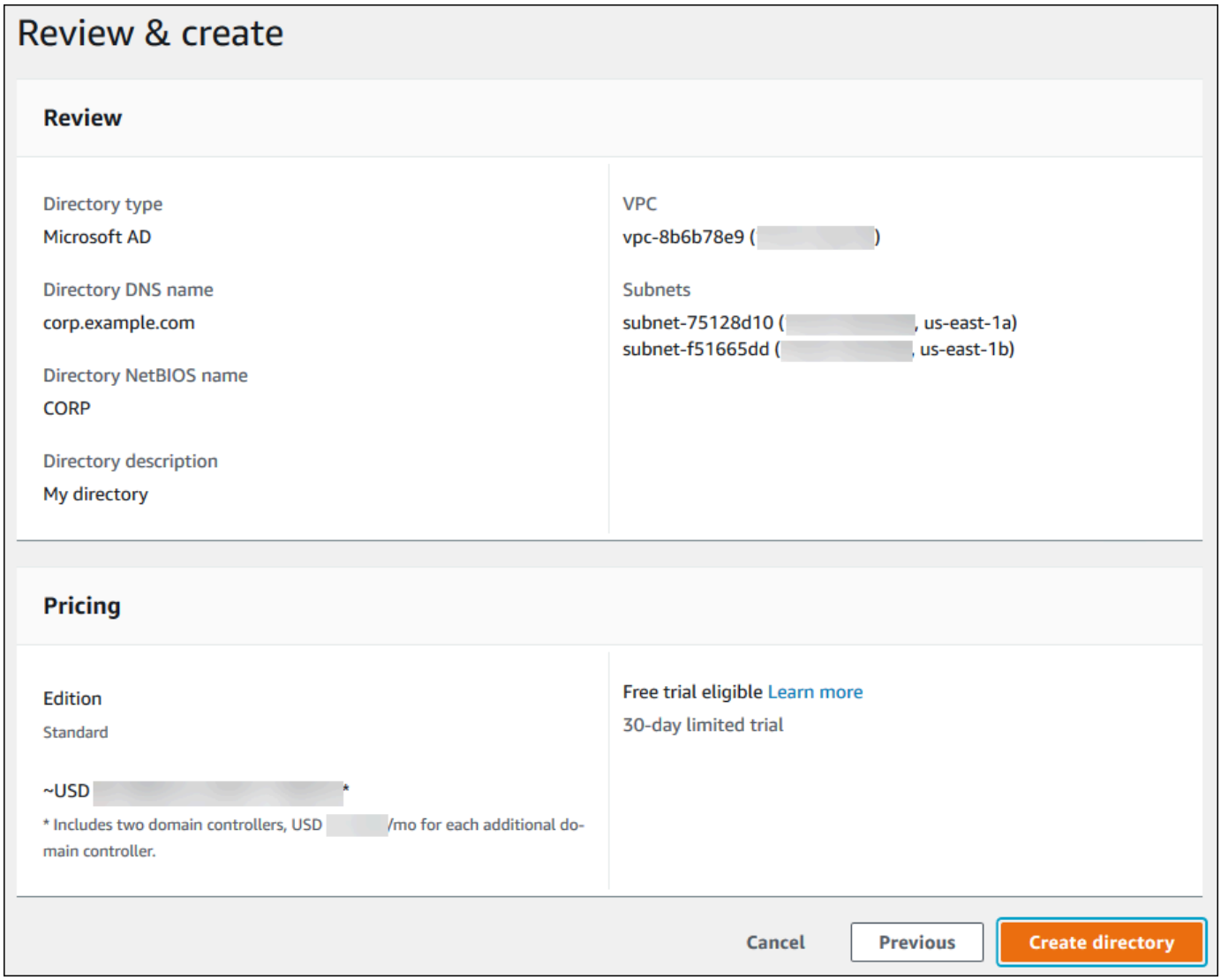
VPC

Pilih VPC untuk direktori. Anda dapat membuat klaster DB PostgreSQL dalam VPC yang sama ini atau dalam VPC yang berbeda.

Subnet

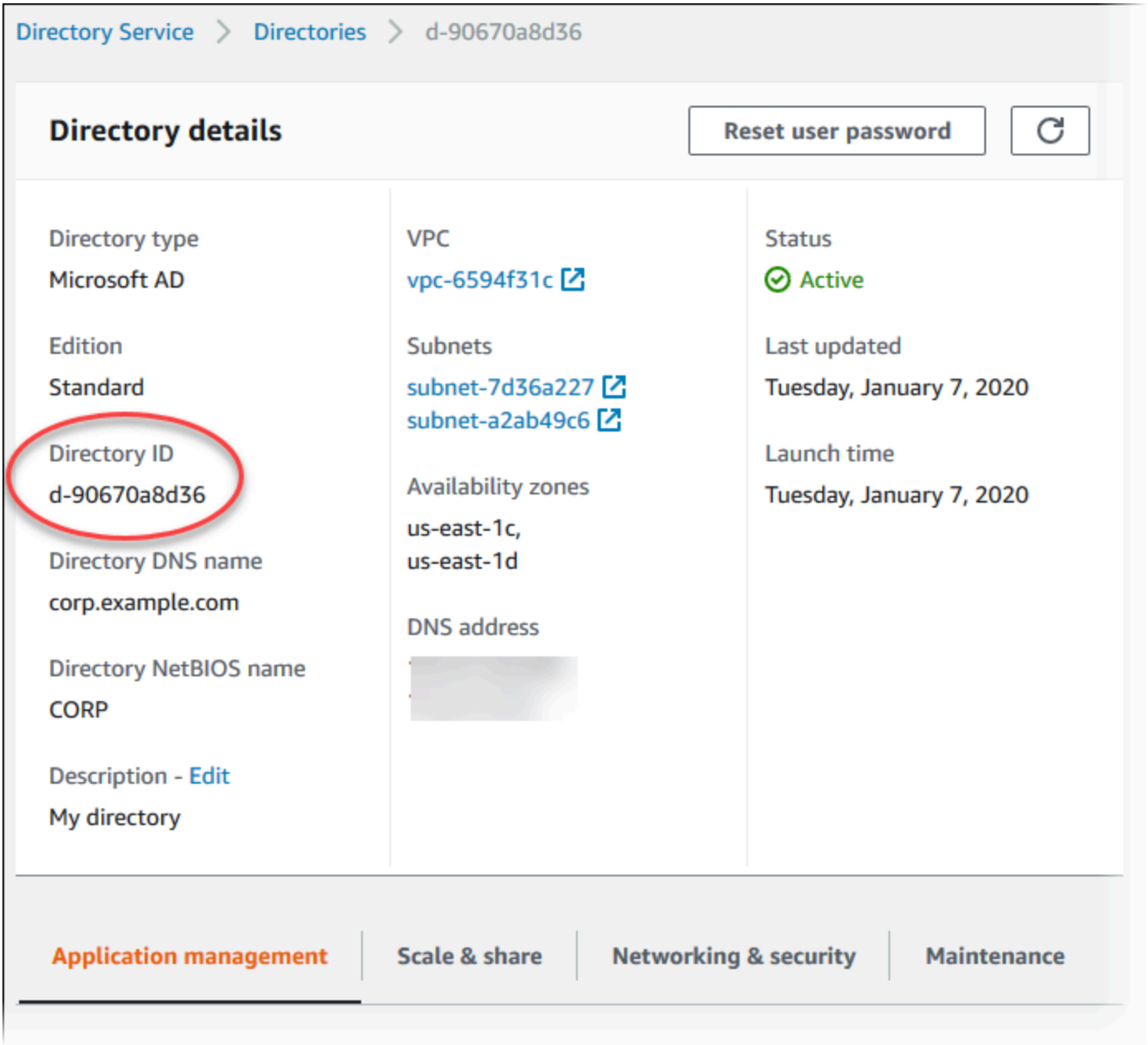
Pilih subnet untuk server direktori. Kedua subnet harus berada di Zona Ketersediaan yang berbeda.

7. Pilih Berikutnya.
8. Tinjau informasi direktori. Jika ada yang perlu diubah, pilih Sebelumnya dan lakukan perubahan. Jika informasi sudah benar, pilih Buat direktori.



Pembuatan direktori memerlukan waktu beberapa menit. Setelah berhasil dibuat, nilai Status berubah menjadi Aktif.

Untuk melihat informasi tentang direktori Anda, pilih ID direktori di daftar direktori. Buat catatan tentang nilai ID Direktori. Anda memerlukan nilai ini saat membuat atau mengubah instans DB PostgreSQL.



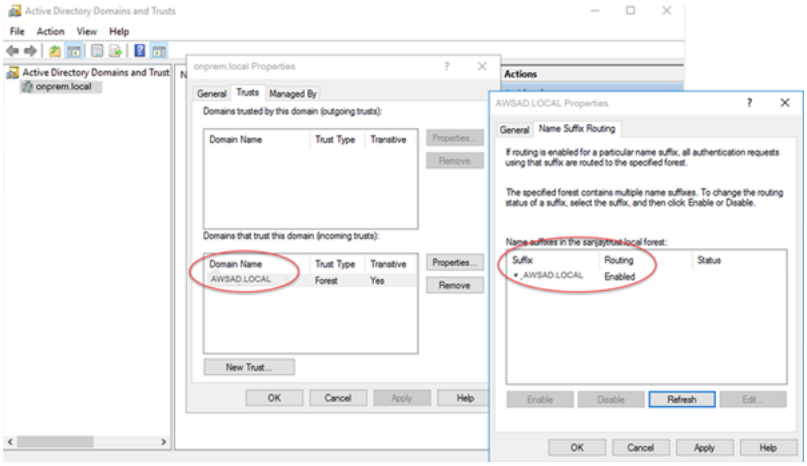
Langkah 2: (Opsional) Buat hubungan kepercayaan antara Active Directory lokal dan AWS Directory Service

Jika Anda tidak berencana untuk menggunakan Microsoft Active Directory on-premise Anda sendiri, langsung ke Langkah 3: Buat peran IAM untuk Amazon RDS untuk mengakses AWS Directory Service.

Untuk mendapatkan autentikasi Kerberos menggunakan Active Directory lokal, Anda perlu membuat relasi domain kepercayaan menggunakan trust hutan antara Microsoft Active Directory lokal dan direktori (dibuat di AWS Managed Microsoft AD). Langkah 1: Buat direktori menggunakan AWS Managed Microsoft AD Kepercayaan bisa satu arah, di mana AWS Managed Microsoft AD direktori mempercayai Microsoft Active Directory lokal. Kepercayaan juga dapat bersifat dua arah, di mana kedua Active Directory saling mempercayai. Untuk informasi selengkapnya tentang menyiapkan trust menggunakan AWS Directory Service, lihat Kapan membuat hubungan kepercayaan di Panduan AWS Directory Service Administrasi.

> **Note**
>
> Jika Anda menggunakan Microsoft Active Directory on-premise:
>
> - Klien Windows harus terhubung menggunakan nama domain AWS Directory Service di titik akhir daripada rds.amazonaws.com. Untuk informasi selengkapnya, lihat Menghubungkan ke PostgreSQL dengan autentikasi Kerberos.
> - Klien Windows tidak dapat terhubung menggunakan titik akhir kustom Aurora. Untuk mempelajari selengkapnya, lihat Manajemen koneksi Amazon Aurora.
> - Untuk basis data global:
>   - Klien Windows dapat terhubung menggunakan titik akhir instance atau titik akhir cluster di primer Wilayah AWS database global saja.
>   - Klien Windows tidak dapat terhubung menggunakan titik akhir cluster di sekunder Wilayah AWS.

Pastikan bahwa nama domain Microsoft Active Directory on-premise Anda mencakup perutean akhiran DNS yang sesuai dengan hubungan kepercayaan yang baru dibuat. Tangkapan layar berikut menunjukkan sebuah contoh.



Langkah 3: Buat peran IAM untuk Amazon RDS untuk mengakses AWS Directory Service

Agar Amazon Aurora Amazon memanggil AWS Directory Service Anda, AWS akun Anda memerlukan peran IAM yang menggunakan kebijakan IAM terkelola. `AmazonRDSDirectoryServiceAccess` Peran ini membuat Amazon Aurora dapat melakukan panggilan ke AWS Directory Service. (Perhatikan bahwa peran IAM ini untuk mengakses berbeda dari peran IAM yang digunakan.) AWS Directory Service Autentikasi basis data IAM

Saat Anda membuat instans DB menggunakan AWS Management Console dan akun pengguna konsol Anda memiliki `iam:CreateRole` izin, konsol akan membuat peran IAM yang diperlukan secara otomatis. Dalam hal ini, nama perannya adalah `rds-directoryservice-kerberos-access-role`. Jika tidak, Anda harus membuat peran IAM secara manual. Saat Anda membuat peran IAM ini, pilih`Directory Service`, dan lampirkan kebijakan AWS terkelola `AmazonRDSDirectoryServiceAccess` padanya.

Untuk informasi selengkapnya tentang membuat peran IAM untuk layanan, lihat Membuat peran untuk mendelegasikan izin ke AWS layanan di Panduan Pengguna IAM.

> **Note**
>
> Peran IAM yang digunakan untuk Autentikasi Windows untuk RDS for Microsoft SQL Server tidak dapat digunakan untuk Amazon Aurora.

Sebagai alternatif untuk menggunakan kebijakan terkelola `AmazonRDSDirectoryServiceAccess`, Anda dapat membuat kebijakan dengan izin yang diperlukan. Dalam hal ini, peran IAM harus memiliki kebijakan kepercayaan IAM berikut.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "directoryservice.rds.amazonaws.com",
          "rds.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Peran ini juga harus memiliki kebijakan peran IAM berikut.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ds:DescribeDirectories",
        "ds:AuthorizeApplication",
        "ds:UnauthorizeApplication",
        "ds:GetAuthorizedApplicationDetails"
      ],
    "Effect": "Allow",
    "Resource": "*"
    }
  ]
}
```

Langkah 4: Buat dan konfigurasikan pengguna

Anda dapat membuat pengguna dengan alat Active Directory Users and Computers. Alat ini merupakan salah satu alat Active Directory Domain Services dan Active Directory Lightweight Directory Services. Untuk informasi selengkapnya, lihat Add Users and Computers to the Active Directory domain dalam dokumentasi Microsoft. Dalam hal ini, pengguna adalah individu atau

entitas lain, seperti komputer mereka yang merupakan bagian dari domain dan yang identitasnya dipertahankan dalam direktori.

Untuk membuat pengguna di AWS Directory Service direktori, Anda harus terhubung ke instans Amazon EC2 berbasis Windows yang merupakan anggota direktori. AWS Directory Service Pada saat yang sama, Anda harus masuk sebagai pengguna yang memiliki hak untuk membuat pengguna. Untuk informasi selengkapnya, lihat Membuat pengguna dalam Panduan Administrasi AWS Directory Service .

Langkah 5: Aktifkan lalu lintas antar-VPC antara direktori dan instans DB

Jika Anda ingin menemukan direktori dan klaster DB dalam VPC yang sama, lewati langkah ini dan lanjutkan ke Langkah 6: Buat atau modifikasi klaster DB PostgreSQL.

Jika Anda ingin menemukan direktori dan instans DB di VPC yang berbeda, konfigurasikan lalu lintas antar-VPC menggunakan peering VPC atau AWS Transit Gateway.

Prosedur berikut mengaktifkan lalu lintas antar-VPC menggunakan peering VPC. Ikuti petunjuk di Apa yang dimaksud dengan peering VPC? dalam Panduan Peering Amazon Virtual Private Cloud.

Untuk mengaktifkan lalu lintas VPC menggunakan peering VPC

1. Siapkan aturan perutean VPC yang sesuai untuk memastikan lalu lintas jaringan dapat berjalan dua arah.
2. Pastikan bahwa grup keamanan instans DB dapat menerima lalu lintas masuk dari grup keamanan direktori.
3. Pastikan tidak ada aturan daftar kontrol akses (ACL) jaringan yang memblokir lalu lintas.

Jika AWS akun lain memiliki direktori, Anda harus berbagi direktori.

Untuk berbagi direktori antar AWS akun

1. Mulai berbagi direktori dengan AWS akun tempat instans DB akan dibuat dengan mengikuti petunjuk di Tutorial: Berbagi direktori AD Microsoft AWS Terkelola Anda untuk Domain EC2 yang mulus-Bergabung dalam Panduan Administrasi.AWS Directory Service
2. Masuk ke AWS Directory Service konsol menggunakan akun untuk instans DB, dan pastikan domain memiliki `SHARED` status sebelum melanjutkan.

3. Saat masuk ke AWS Directory Service konsol menggunakan akun untuk instans DB, perhatikan nilai ID Direktori. Anda menggunakan ID direktori ini untuk menggabungkan instans DB ke domain.

Langkah 6: Buat atau modifikasi klaster DB PostgreSQL

Buat atau modifikasi klaster DB PostgreSQL untuk digunakan dengan direktori Anda. Anda dapat menggunakan konsol, CLI, atau RDS API untuk mengaitkan klaster DB dengan direktori. Anda dapat melakukannya dengan salah satu cara berikut:

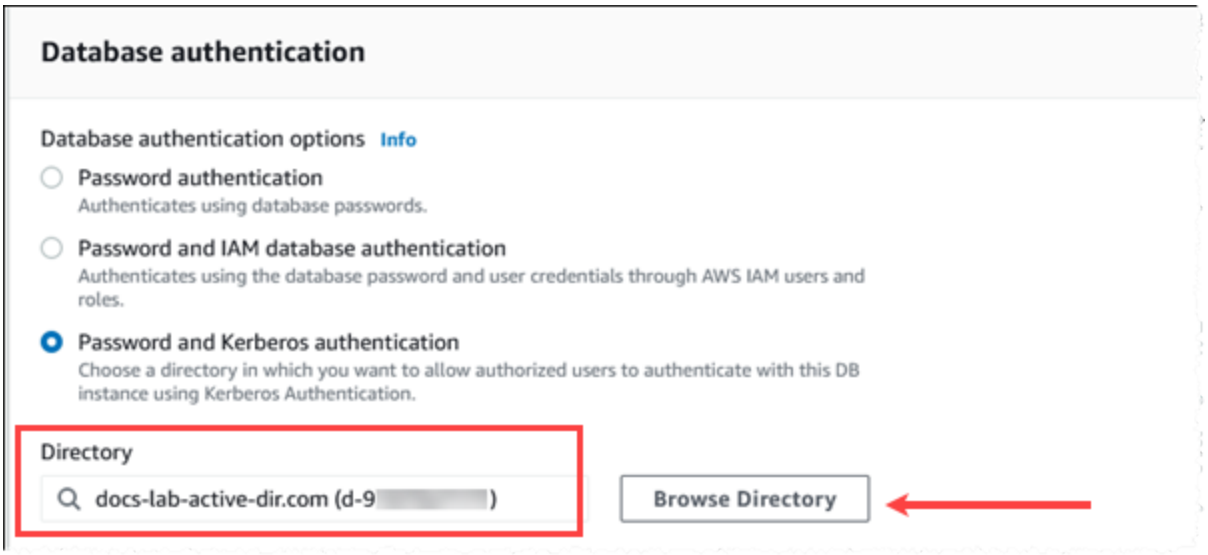
- Buat cluster PostgreSQL DB baru menggunakan konsol, perintah create-db-clusterCLI, atau operasi CreateDBCluster RDS API. Untuk petunjuk, lihat Membuat dan terhubung ke klaster DB Aurora PostgreSQL.
- Ubah cluster PostgreSQL DB yang ada menggunakan konsol, perintah modify-db-clusterCLI, atau operasi ModifyDBCluster RDS API. Untuk petunjuk, lihat Memodifikasi klaster DB Amazon Aurora.
- Pulihkan cluster PostgreSQL DB dari snapshot DB menggunakan konsol, perintah CLI restore-db-cluster-from-db-snapshot, atau operasi RestoreDB dbSnapshot RDS API. ClusterFrom Untuk petunjuk, lihat Memulihkan dari snapshot klaster DB.
- Kembalikan cluster PostgreSQL DB ke point-in-time menggunakan konsol, perintah restore-db-instance-to- point-in-time CLI, atau operasi RestoreDB RDS API. ClusterToPointInTime Untuk petunjuknya, lihat Memulihkan klaster DB ke waktu tertentu.

Autentikasi Kerberos hanya didukung untuk klaster DB PostgreSQL dalam sebuah VPC. Klaster DB boleh berada dalam VPC yang sama dengan direktori, atau dalam VPC yang berbeda. Klaster DB harus menggunakan grup keamanan yang memungkinkan ingress and egress di dalam VPC direktori, sehingga klaster DB dapat berkomunikasi dengan direktori.

> **Note**
>
> Mengaktifkan otentikasi Kerberos saat ini tidak didukung di klaster Aurora PostgreSQL DB selama migrasi dari RDS untuk PostgreSQL. Anda dapat mengaktifkan otentikasi Kerberos hanya pada cluster Aurora PostgreSQL DB mandiri.

Konsol

Saat Anda menggunakan konsol untuk membuat, memodifikasi, atau memulihkan klaster DB, pilih Autentikasi Kerberos di bagian Autentikasi basis data. Kemudian pilih Cari Direktori. Pilih direktori atau pilih Buat direktori baru untuk menggunakan Directory Service.



AWS CLI

Saat Anda menggunakan AWS CLI, parameter berikut diperlukan untuk cluster DB agar dapat menggunakan direktori yang Anda buat:

- Untuk parameter `--domain`, gunakan pengidentifikasi domain (pengidentifikasi "d-*") yang dihasilkan saat Anda membuat direktori.
- Untuk parameter `--domain-iam-role-name`, gunakan peran yang Anda buat yang menggunakan kebijakan IAM terkelola `AmazonRDSDirectoryServiceAccess`.

Misalnya, perintah CLI berikut memodifikasi klaster DB untuk menggunakan direktori.

```
aws rds  modify-db-cluster --db-cluster-identifier mydbinstance --domain d-Directory-ID
 --domain-iam-role-name role-name
```

> **Important**
>
> Jika Anda memodifikasi klaster DB untuk mengaktifkan autentikasi Kerberos, boot ulang klaster DB setelah membuat perubahan.

Langkah 7: Buat pengguna PostgreSQL untuk pengguna utama Kerberos Anda

Pada titik ini, klaster DB Aurora PostgreSQL Anda digabungkan ke domain AWS Managed Microsoft AD . Pengguna yang Anda buat di direktori Langkah 4: Buat dan konfigurasikan pengguna perlu diatur sebagai pengguna basis data PostgreSQL dan diberi hak istimewa untuk masuk ke basis data. Anda dapat melakukannya dengan masuk sebagai pengguna basis data dengan hak istimewa `rds_superuser`. Misalnya, jika Anda menerima default saat membuat klaster DB Aurora PostgreSQL, , gunakan `postgres`, seperti yang ditunjukkan dalam langkah berikut.

Untuk membuat pengguna basis data PostgreSQL untuk pengguna utama Kerberos

1. Gunakan `psql` untuk menghubungkan ke titik akhir klaster DB Aurora PostgreSQL instans DB menggunakan `psql`. Contoh berikut menggunakan akun `postgres` default untuk peran `rds_superuser`.

   ```
   psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

2. Buat nama pengguna basis data untuk setiap pengguna utama Kerberos (nama pengguna Active Directory) yang ingin Anda beri akses ke basis data. Gunakan nama pengguna (identitas) kanonik seperti yang didefinisikan dalam instans Active Directory, yaitu `alias` dalam huruf kecil (nama pengguna di Active Directory) dan nama domain Active Directory dalam huruf besar untuk nama pengguna tersebut. Nama pengguna Active Directory adalah pengguna yang diautentikasi secara eksternal, jadi gunakan tanda kutip pada nama ini seperti yang ditunjukkan berikut.

   ```
   postgres=> CREATE USER "username@CORP.EXAMPLE.COM" WITH LOGIN;
   CREATE ROLE
   ```

3. Beri peran `rds_ad` kepada pengguna basis data.

   ```
   postgres=> GRANT rds_ad TO "username@CORP.EXAMPLE.COM";
   GRANT ROLE
   ```

Setelah Anda selesai membuat semua pengguna PostgreSQL untuk identitas pengguna Active Directory Anda, pengguna dapat mengakses klaster DB Aurora PostgreSQL menggunakan kredensial Kerberos mereka.

Diasumsikan bahwa pengguna basis data yang melakukan autentikasi menggunakan Kerberos melakukannya dari mesin klien yang merupakan anggota domain Active Directory.

Pengguna basis data yang telah diberi peran `rds_ad` tidak dapat memiliki peran `rds_iam`. Aturan ini juga berlaku untuk keanggotaan bertingkat. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.

Mengonfigurasi klaster DB Aurora PostgreSQL Anda untuk nama pengguna yang tidak peka huruf besar/kecil

Aurora PostgreSQL versi 14.5, 13.8, 12.12, dan 11.17 mendukung parameter `krb_caseins_users` PostgreSQL. Parameter ini mendukung nama pengguna Active Directory yang tidak peka huruf besar/kecil. Secara default, parameter ini diatur ke salah, sehingga nama pengguna ditafsirkan sebagai tidak peka huruf besar/kecil oleh Aurora PostgreSQL. Hal tersebut merupakan perilaku default di semua versi lama Aurora PostgreSQL. Namun, Anda dapat mengatur parameter ini ke `true` dalam grup parameter klaster DB kustom Anda dan mengizinkan klaster DB Aurora PostgreSQL Anda untuk menafsirkan nama pengguna sebagai tidak peka huruf besar/kecil. Sebaiknya lakukan hal ini agar memudahkan pengguna basis data Anda, yang terkadang salah mengetik kapitalisasi nama pengguna saat melakukan autentikasi menggunakan Active Directory.

Untuk mengubah parameter `krb_caseins_users`, klaster DB Aurora PostgreSQL Anda harus menggunakan grup parameter klaster DB kustom. Untuk informasi tentang penggunaan grup parameter klaster DB kustom, lihat Bekerja dengan grup parameter.

Anda dapat menggunakan AWS CLI atau AWS Management Console untuk mengubah pengaturan. Untuk informasi selengkapnya, lihat Mengubah parameter dalam grup parameter klaster DB.

Langkah 8: Konfigurasikan klien PostgreSQL

Untuk mengonfigurasi klien PostgreSQL, lakukan langkah berikut:

- Buat file krb5.conf (atau yang setara) untuk menunjuk ke domain.
- Verifikasi bahwa lalu lintas dapat mengalir antara host klien dan AWS Directory Service. Gunakan utilitas jaringan seperti Netcat untuk hal berikut:
  - Memeriksa lalu lintas melalui DNS untuk port 53.
  - Periksa lalu lintas atas TCP/UDP untuk port 53 dan untuk Kerberos, yang mencakup port 88 dan 464 untuk AWS Directory Service.
- Memastikan lalu lintas dapat mengalir di antara host klien dan instans DB melalui port basis data. Misalnya, gunakan psql untuk menghubungkan dan mengakses basis data.

Berikut ini adalah contoh konten krb5.conf untuk. AWS Managed Microsoft AD

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

Berikut adalah contoh konten krb5.conf untuk Microsoft Active Directory on-premise.

```
[libdefaults]
 default_realm = EXAMPLE.COM
[realms]
 EXAMPLE.COM = {
  kdc = example.com
  admin_server = example.com
 }
 ONPREM.COM = {
  kdc = onprem.com
  admin_server = onprem.com
 }
[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
 .onprem.com = ONPREM.COM
 onprem.com = ONPREM.COM
 .rds.amazonaws.com = EXAMPLE.COM
 .amazonaws.com.cn = EXAMPLE.COM
 .amazon.com = EXAMPLE.COM
```

Mengelola klaster DB di Domain

Anda dapat menggunakan konsol, CLI, atau RDS API untuk mengelola klaster DB dan hubungannya dengan Microsoft Active Directory. Misalnya, Anda dapat mengaitkan Active Directory untuk mengaktifkan autentikasi Kerberos. Anda juga dapat menghapus pengaitan untuk Active Directory guna menonaktifkan autentikasi Kerberos. Anda juga dapat memindahkan klaster DB untuk diautentikasi secara eksternal oleh satu Microsoft Active Directory ke yang lain.

Misalnya, dengan CLI, Anda dapat melakukan hal berikut:

- Untuk mencoba kembali mengaktifkan autentikasi Kerberos untuk keanggotaan yang gagal, gunakan perintah CLI modify-db-cluster. Tentukan ID direktori keanggotaan saat ini untuk opsi `--domain`.
- Untuk menonaktifkan autentikasi Kerberos pada instans DB, gunakan perintah CLI modify-db-cluster. Tentukan `none` untuk opsi `--domain`.
- Untuk memindahkan instans DB dari satu domain ke domain lain, gunakan perintah CLI modify-db-cluster. Tentukan pengidentifikasi domain dari domain baru untuk opsi `--domain`.

Memahami keanggotaan Domain

Setelah Anda membuat atau memodifikasi klaster DB, instans DB menjadi anggota domain. Anda dapat melihat status keanggotaan domain di konsol atau dengan menjalankan perintah CLI describe-db-instances. Status instans DB dapat berupa salah satu dari berikut ini:

- `kerberos-enabled` – Instans DB mengaktifkan autentikasi Kerberos.
- `enabling-kerberos` – AWS sedang dalam proses mengaktifkan autentikasi Kerberos pada instans DB ini.
- `pending-enable-kerberos` – Pengaktifan autentikasi Kerberos tertunda pada instans DB ini.
- `pending-maintenance-enable-kerberos` – AWS akan berusaha mengaktifkan autentikasi Kerberos di instans DB pada jendela pemeliharaan terjadwal berikutnya.
- `pending-disable-kerberos` – Penonaktifan autentikasi Kerberos tertunda pada instans DB ini.
- `pending-maintenance-disable-kerberos` – AWS akan berusaha menonaktifkan autentikasi Kerberos di instans DB pada jendela pemeliharaan terjadwal berikutnya.
- `enable-kerberos-failed` – Masalah konfigurasi mencegah AWS mengaktifkan autentikasi Kerberos pada instans DB. Perbaiki masalah konfigurasi sebelum menerbitkan ulang perintah untuk memodifikasi instans DB.
- `disabling-kerberos` – AWS sedang dalam proses menonaktifkan autentikasi Kerberos pada instans DB ini.

Permintaan untuk mengaktifkan autentikasi Kerberos dapat gagal karena masalah konektivitas jaringan atau peran IAM yang salah. Dalam beberapa kasus, upaya untuk mengaktifkan autentikasi Kerberos mungkin gagal saat Anda membuat atau memodifikasi klaster DB. Jika demikian, pastikan Anda menggunakan peran IAM yang benar, kemudian ubah klaster DB untuk bergabung ke domain.

Menghubungkan ke PostgreSQL dengan autentikasi Kerberos

Anda dapat terhubung ke PostgreSQL dengan autentikasi Kerberos, dengan antarmuka pgAdmin, atau dengan antarmuka baris perintah seperti psql. Untuk informasi selengkapnya tentang cara menghubungkan, lihat Menghubungkan ke klaster DB Amazon Aurora PostgreSQL.. Untuk mengetahui informasi tentang mendapatkan titik akhir, nomor port, dan detail lain yang diperlukan untuk koneksi, lihat Melihat titik akhir untuk klaster Aurora.

pgAdmin

Agar dapat menggunakan pgAdmin untuk terhubung ke PostgreSQL dengan autentikasi Kerberos, lakukan langkah berikut:

1. Luncurkan aplikasi pgAdmin di komputer klien Anda.
2. Pada tab Dasbor, pilih Tambahkan Server Baru.
3. Di kotak dialog Buat - Server, masukkan nama pada tab Umum untuk mengidentifikasi server di pgAdmin.
4. Pada tab Koneksi, masukkan informasi berikut dari basis data Aurora PostgreSQL Anda.
   - Untuk Host, masukkan titik akhir untuk instans Penulis dari klaster DB Aurora PostgreSQL Anda. Titik akhir terlihat seperti berikut ini:

     ```
     AUR-cluster-instance.111122223333.aws-region.rds.amazonaws.com
     ```

     Untuk terhubung ke Microsoft Active Directory on-premise dari klien Windows, gunakan nama domain AWS Managed Active Directory, bukan `rds.amazonaws.com` di titik akhir host. Misalnya, anggaplah nama domain untuk AWS Managed Active Directory adalah `corp.example.com`. Kemudian untuk Host, titik akhir akan ditentukan sebagai berikut:

     ```
     AUR-cluster-instance.111122223333.aws-region.corp.example.com
     ```

   - Untuk Port, masukkan port yang ditetapkan.
   - Untuk Basis data pemeliharaan, masukkan nama basis data awal yang akan dihubungkan ke klien.
   - Untuk Nama pengguna, masukkan nama pengguna yang Anda masukkan untuk autentikasi Kerberos di Langkah 7: Buat pengguna PostgreSQL untuk pengguna utama Kerberos Anda .
5. Pilih Simpan.

Psql

Agar dapat menggunakan psql untuk terhubung ke PostgreSQL dengan autentikasi Kerberos, lakukan langkah berikut:

1. Pada prompt perintah, jalankan perintah berikut.

   ```
   kinit username
   ```

   Ganti *username* dengan nama pengguna. Pada prompt, masukkan kata sandi yang disimpan dalam Microsoft Active Directory untuk pengguna.

2. Jika klaster DB PostgreSQL menggunakan VPC yang dapat diakses publik, masukkan alamat IP untuk titik akhir klaster DB di file `/etc/hosts` Anda pada klien EC2. Misalnya, perintah berikut mendapatkan alamat IP lalu memasukkannya ke dalam file `/etc/hosts`.

   ```
   % dig +short PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com
   ;; Truncated, retrying in TCP mode.
   ec2-34-210-197-118.AWS-Region.compute.amazonaws.com.
   34.210.197.118

   % echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com" >> /etc/
   hosts
   ```

   Jika Anda menggunakan Microsoft Active Directory on-premise dari klien Windows, Anda perlu terhubung menggunakan titik akhir khusus. Alih-alih menggunakan domain Amazon `rds.amazonaws.com` di titik akhir host, gunakan nama domain AWS Managed Active Directory.

   Misalnya, anggaplah nama domain untuk AWS Managed Active Directory Anda adalah `corp.example.com`. Kemudian, gunakan format *PostgreSQL-endpoint.AWS-Region.corp.example.com* untuk titik akhir dan tempatkan ke dalam file `/etc/hosts`.

   ```
   % echo " 34.210.197.118  PostgreSQL-endpoint.AWS-Region.corp.example.com" >> /etc/
   hosts
   ```

3. Gunakan perintah psql berikut untuk login ke klaster DB PostgreSQL yang terintegrasi dengan Active Directory. Gunakan titik akhir klaster atau instans.

   ```
   psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
   Region.rds.amazonaws.com postgres
   ```

Untuk login ke klaster DB PostgreSQL dari klien Windows menggunakan Active Directory on-premise, gunakan perintah psql berikut dengan nama domain dari langkah sebelumnya (`corp.example.com`):

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-endpoint.AWS-
Region.corp.example.com postgres
```

Menggunakan grup keamanan AD untuk kontrol akses Aurora PostgreSQL

Dari Aurora PostgreSQL 14.10 dan 15.5 versi, Aurora PostgreSQL Access Control dapat dikelola menggunakan Directory Service untuk grup keamanan Microsoft Active Directory (AD). AWS Versi sebelumnya dari Aurora PostgreSQL mendukung otentikasi berbasis Kerberos dengan AD hanya untuk pengguna individu. Setiap pengguna AD harus secara eksplisit disediakan ke cluster DB untuk mendapatkan akses.

Alih-alih secara eksplisit menyediakan setiap pengguna AD ke klaster DB berdasarkan kebutuhan bisnis, Anda dapat memanfaatkan grup keamanan AD seperti yang dijelaskan di bawah ini:

- Pengguna AD adalah anggota dari berbagai grup keamanan AD di Active Directory. Ini tidak ditentukan oleh administrator cluster DB, tetapi didasarkan pada persyaratan bisnis, dan ditangani oleh administrator AD.
- Administrator cluster DB membuat peran DB dalam instans DB berdasarkan persyaratan bisnis. Peran DB ini mungkin memiliki izin atau hak istimewa yang berbeda.
- Administrator cluster DB mengonfigurasi pemetaan dari grup keamanan AD ke peran DB berdasarkan per cluster DB.
- Pengguna DB dapat mengakses kluster DB menggunakan kredensyal AD mereka. Akses didasarkan pada keanggotaan grup keamanan AD. Pengguna iklan mendapatkan atau kehilangan akses secara otomatis berdasarkan keanggotaan grup iklan mereka.

Prasyarat

Pastikan Anda memiliki hal berikut sebelum menyiapkan ekstensi untuk grup Keamanan AD:

- Siapkan otentikasi Kerberos untuk cluster PostgreSQL DB. Untuk informasi selengkapnya, lihat Menyiapkan otentikasi Kerberos untuk klaster PostgreSQL DB.

  > **Note**
  >
  > Untuk grup keamanan AD, lewati Langkah 7: Buat pengguna PostgreSQL untuk prinsipal Kerberos Anda dalam prosedur penyiapan ini.

- Mengelola cluster DB dalam Domain. Untuk informasi selengkapnya, lihat Mengelola cluster DB di Domain.

Menyiapkan ekstensi pg_ad_mapping

Aurora PostgreSQL sekarang menyediakan `pg_ad_mapping` ekstensi untuk mengelola pemetaan antara grup keamanan AD dan peran DB di cluster Aurora PostgreSQL. Untuk informasi selengkapnya tentang fungsi yang disediakan oleh`pg_ad_mapping`, lihat[Menggunakan fungsi dari `pg_ad_mapping` ekstensi](#).

Untuk menyiapkan `pg_ad_mapping` ekstensi pada klaster Aurora PostgreSQL DB, Anda terlebih dahulu menambahkan `pg_ad_mapping` ke pustaka bersama pada grup parameter cluster DB kustom untuk klaster Aurora PostgreSQL DB Anda. Untuk informasi tentang membuat grup parameter cluster DB kustom, lihat[Bekerja dengan grup parameter](#). Selanjutnya, Anda menginstal `pg_ad_mapping` ekstensi. Prosedur di bagian ini menunjukkan caranya kepada Anda. Anda dapat menggunakan AWS Management Console atau AWS CLI.

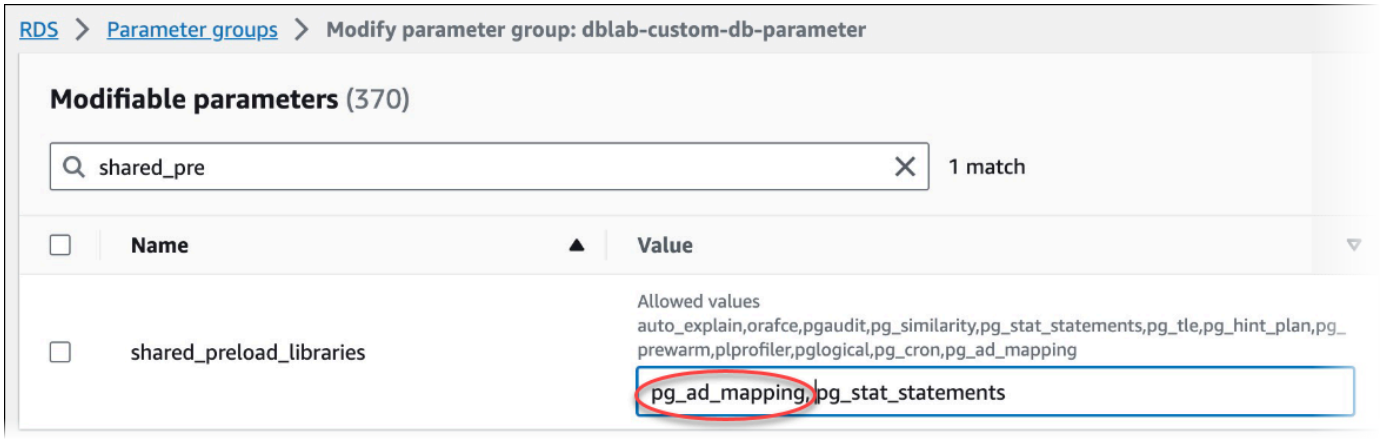
Anda harus memiliki izin sebagai peran `rds_superuser` untuk melakukan semua tugas ini.

Langkah-langkah berikut mengasumsikan bahwa cluster Aurora PostgreSQL DB Anda dikaitkan dengan grup parameter cluster DB kustom.

Konsol

Untuk mengatur **`pg_ad_mapping`** ekstensi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih instance Writer cluster PostgreSQL DB Aurora Anda.
3. Buka tab Konfigurasi untuk instance penulis cluster Aurora PostgreSQL DB Anda. Di antara detail Instans, temukan tautan Grup parameter.
4. Pilih tautan untuk membuka parameter kustom yang terkait dengan klaster DB Aurora PostgreSQL Anda.
5. Di kolom pencarian Parameter, ketik `shared_pre` untuk menemukan parameter `shared_preload_libraries`.
6. Pilih Edit parameter untuk mengakses nilai properti.
7. Tambahkan `pg_ad_mapping` ke daftar di kolom Nilai. Gunakan koma untuk memisahkan item dalam daftar nilai.



8. Reboot instance penulis cluster Aurora PostgreSQL DB Anda sehingga perubahan Anda pada parameter berlaku. `shared_preload_libraries`

9. Ketika instans tersedia, verifikasi bahwa `pg_ad_mapping` telah diinisialisasi. Gunakan `psql` untuk terhubung ke instance penulis cluster Aurora PostgreSQL DB Anda, dan kemudian jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_ad_mapping
(1 row)
```

10. Dengan `pg_ad_mapping` diinisialisasi, Anda sekarang dapat membuat ekstensi. Anda perlu membuat ekstensi setelah menginisialisasi perpustakaan untuk mulai menggunakan fungsi yang disediakan oleh ekstensi ini.

```
CREATE EXTENSION pg_ad_mapping;
```

11. Tutup sesi `psql`.

```
labdb=> \q
```

AWS CLI

Untuk mengatur pg_ad_mapping

Untuk mengatur pg_ad_mapping menggunakan AWS CLI, Anda memanggil modify-db-parameter-groupoperasi untuk menambahkan parameter ini dalam grup parameter kustom Anda, seperti yang ditunjukkan dalam prosedur berikut.

1. Gunakan AWS CLI perintah berikut `pg_ad_mapping` untuk menambah `shared_preload_libraries` parameter.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_ad_mapping,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Gunakan AWS CLI perintah berikut untuk me-reboot instance penulis cluster Aurora PostgreSQL DB Anda sehingga pg_ad_mapping diinisialisasi.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. Saat instans tersedia, verifikasikan bahwa `pg_ad_mapping` telah diinisialisasi. Gunakan `psql` untuk terhubung ke instance penulis cluster Aurora PostgreSQL DB Anda, dan kemudian jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_ad_mapping
(1 row)
```

   Dengan pg_ad_mapping diinisialisasi, Anda sekarang dapat membuat ekstensi.

```
CREATE EXTENSION pg_ad_mapping;
```

4. Tutup sesi `psql` sehingga Anda dapat menggunakan AWS CLI.

```
labdb=> \q
```

Mengambil SID Grup Direktori Aktif di PowerShell

Security identifier (SID) digunakan untuk mengidentifikasi prinsip keamanan atau kelompok keamanan secara unik. Setiap kali grup keamanan atau akun dibuat di Active Directory, SID ditugaskan untuk itu. Untuk mengambil SID grup keamanan AD dari direktori aktif, Anda dapat menggunakan cmdlet Get-AdGroup dari mesin klien windows yang bergabung dengan domain Active Directory tersebut. Parameter Identity menentukan nama grup Active Directory untuk mendapatkan SID yang sesuai.

Contoh berikut mengembalikan SID grup AD *adgroup1*.

```
C:\Users\Admin> Get-ADGroup -Identity adgroup1 | select SID

             SID
-----------------------------------------------
S-1-5-21-3168537779-1985441202-1799118680-1612
```

Memetakan peran DB dengan grup keamanan AD

Anda harus secara eksplisit menyediakan grup keamanan AD dalam database sebagai peran PostgreSQL DB. Pengguna AD, yang merupakan bagian dari setidaknya satu grup keamanan AD yang disediakan akan mendapatkan akses ke database. Anda tidak boleh memberikan `rds_ad role` peran DB berbasis keamanan grup AD. *Otentikasi Kerberos untuk grup keamanan akan dipicu dengan menggunakan akhiran nama domain seperti user1@example.com.* Peran DB ini tidak dapat menggunakan Password atau autentikasi IAM untuk mendapatkan akses ke database.

> **Note**
>
> Pengguna AD yang memiliki peran DB terkait dalam database dengan `rds_ad` peran yang diberikan kepada mereka, tidak dapat masuk sebagai bagian dari grup keamanan AD. Mereka akan mendapatkan akses melalui peran DB sebagai pengguna individu.

Misalnya, grup akun adalah grup keamanan di AD tempat Anda ingin menyediakan grup keamanan ini di Aurora PostgreSQL sebagai peran akun.

Grup Keamanan AD	Peran PosgreSQL DB
akun-grup	akun-peran

Saat memetakan peran DB dengan grup keamanan AD, Anda harus memastikan bahwa peran DB memiliki atribut LOGIN yang ditetapkan dan memiliki hak istimewa CONNECT ke database login yang diperlukan.

```
postgres => alter role accounts-role login;

ALTER ROLE
postgres => grant connect on database accounts-db to accounts-role;
```

Admin sekarang dapat melanjutkan untuk membuat pemetaan antara grup keamanan AD dan peran PostgreSQL DB.

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', <SID>, <Weight>);
```

Untuk informasi tentang mengambil SID grup keamanan AD, lihatMengambil SID Grup Direktori Aktif di PowerShell.

Mungkin ada kasus di mana pengguna AD termasuk dalam beberapa grup, dalam hal ini, pengguna AD akan mewarisi hak istimewa peran DB, yang disediakan dengan bobot tertinggi. Jika kedua peran memiliki bobot yang sama, pengguna AD akan mewarisi hak istimewa peran DB yang sesuai dengan pemetaan yang ditambahkan baru-baru ini. Rekomendasinya adalah untuk menentukan bobot yang mencerminkan izin/hak istimewa relatif dari peran DB individu. Lebih tinggi izin atau hak istimewa peran DB, lebih tinggi bobot yang harus dikaitkan dengan entri pemetaan. Ini akan menghindari ambiguitas dua pemetaan yang memiliki bobot yang sama.

Tabel berikut menunjukkan contoh pemetaan dari grup keamanan AD ke peran Aurora PostgreSQL DB.

Grup Keamanan AD	Peran PosgreSQL DB	Berat Badan
akun-grup	akun-peran	7
kelompok penjualan	peran penjualan	10
kelompok pengembang	peran pengembang	7

Dalam contoh berikut, `user1` akan mewarisi hak istimewa peran penjualan karena memiliki bobot yang lebih tinggi sementara `user2` akan mewarisi hak istimewa sebagai pemetaan untuk peran ini dibuat `dev-role` setelahnya`accounts-role`, yang memiliki bobot yang sama dengan. `accounts-role`

nama pengguna	Keanggotaan Grup Keamanan
user1	akun-kelompok penjualan-kelompok
user2	akun-grup dev-group

Perintah psql untuk membuat, membuat daftar, dan menghapus pemetaan ditunjukkan di bawah ini. Saat ini, tidak mungkin untuk memodifikasi entri pemetaan tunggal. Entri yang ada perlu dihapus dan pemetaan dibuat ulang.

```
admin=>select pgadmap_set_mapping('accounts-group', 'accounts-role', 'S-1-5-67-890',
 7);
admin=>select pgadmap_set_mapping('sales-group', 'sales-role', 'S-1-2-34-560', 10);
admin=>select pgadmap_set_mapping('dev-group', 'dev-role', 'S-1-8-43-612', 7);

admin=>select * from pgadmap_read_mapping();

ad_sid       | pg_role        | weight | ad_grp
-------------+----------------+--------+--------------
S-1-5-67-890 | accounts-role | 7      | accounts-group
S-1-2-34-560 | sales-role    | 10     | sales-group
```

```
S-1-8-43-612 | dev-role      | 7      | dev-group
(3 rows)
```

Pencatatan/audit identitas pengguna AD

Gunakan perintah berikut untuk menentukan peran database yang diwarisi oleh pengguna saat ini atau sesi:

```
postgres=>select session_user, current_user;

session_user | current_user
-------------+--------------
dev-role     | dev-role

(1 row)
```

Untuk menentukan identitas utama keamanan AD, gunakan perintah berikut:

```
postgres=>select principal from pg_stat_gssapi where pid = pg_backend_pid();

 principal
-------------------------
 user1@example.com

(1 row)
```

Saat ini, identitas pengguna AD tidak terlihat di log audit. `log_connections`Parameter dapat diaktifkan untuk mencatat pembentukan sesi DB. Untuk informasi selengkapnya, lihat log_connections. Output untuk ini termasuk identitas pengguna AD, seperti yang ditunjukkan di bawah ini. PID backend yang terkait dengan output ini kemudian dapat membantu tindakan atribut kembali ke pengguna AD yang sebenarnya.

```
pgrole1@postgres:[615]:LOG: connection authorized: user=pgrole1
 database=postgres application_name=psql GSS (authenticated=yes, encrypted=yes,
 principal=Admin@EXAMPLE.COM)
```

Batasan

- ID Microsoft Entra yang dikenal sebagai Azure Active Directory tidak didukung.

Menggunakan fungsi dari **`pg_ad_mapping`** ekstensi

`pg_ad_mapping`ekstensi memberikan dukungan untuk fungsi-fungsi berikut:

pgadmap_set_mapping

Fungsi ini menetapkan pemetaan antara grup keamanan AD dan peran Database dengan bobot terkait.

Sintaks

```
pgadmap_set_mapping(
ad_group,
db_role,
ad_group_sid,
weight)
```

Argumen

Parameter	Deskripsi
ad_group	Nama Grup AD. Nilai tidak bisa null atau string kosong.
db_role	Peran database yang akan dipetakan ke Grup AD yang ditentukan. Nilai tidak bisa null atau string kosong.
ad_group_id	Pengidentifikasi keamanan yang digunakan untuk mengident ifikasi grup AD secara unik. Nilai dimulai dengan 'S-1-' dan tidak dapat berupa string nol atau kosong. Untuk informasi selengkap nya, lihat Mengambil SID Grup Direktori Aktif di PowerShell.
berat	Berat yang terkait dengan peran database. Peran dengan bobot tertinggi diutamakan ketika pengguna adalah anggota dari beberapa grup. Nilai default berat adalah 1.

Jenis pengembalian

None

Catatan penggunaan

Fungsi ini menambahkan pemetaan baru dari grup keamanan AD ke peran database. Itu hanya dapat dieksekusi pada instance DB utama dari cluster DB oleh pengguna yang memiliki hak istimewa rds_superuser.

Contoh-contoh

```
postgres=> select pgadmap_set_mapping('accounts-group','accounts-
role','S-1-2-33-12345-67890-12345-678',10);

pgadmap_set_mapping

(1 row)
```

pgadmap_read_mapping

Fungsi ini mencantumkan pemetaan antara grup keamanan AD dan peran DB yang disetel menggunakan `pgadmap_set_mapping` fungsi.

Sintaks

```
pgadmap_read_mapping()
```

Argumen

None

Jenis pengembalian

Parameter	Deskripsi
ad_group_id	Pengidentifikasi keamanan yang digunakan untuk mengident ifikasi grup AD secara unik. Nilai dimulai dengan 'S-1-' dan tidak dapat berupa string nol atau kosong. Untuk informasi lebih

Parameter	Deskripsi
	lanjut, lihat Mengambil SID Grup Direktori Aktif di PowerShell .accounts-role@example.com
db_role	Peran database yang akan dipetakan ke Grup AD yang ditentukan. Nilai tidak bisa null atau string kosong.
berat	Berat yang terkait dengan peran database. Peran dengan bobot tertinggi diutamakan ketika pengguna adalah anggota dari beberapa grup. Nilai default berat adalah 1.
ad_group	Nama Grup AD. Nilai tidak bisa null atau string kosong.

Catatan penggunaan

Panggil fungsi ini untuk mencantumkan semua pemetaan yang tersedia antara grup keamanan AD dan peran DB.

Contoh-contoh

```
postgres=> select * from pgadmap_read_mapping();

ad_sid                              | pg_role       | weight | ad_grp
------------------------------------+---------------+--------+------------------
S-1-2-33-12345-67890-12345-678      | accounts-role | 10     | accounts-group
(1 row)

(1 row)
```

pgadmap_reset_mapping

Fungsi ini me-reset satu atau semua pemetaan yang diatur menggunakan fungsi. `pgadmap_set_mapping`

Sintaks

```
pgadmap_reset_mapping(
ad_group_sid,
db_role,
```

```
weight)
```

Argumen

Parameter	Deskripsi
ad_group_id	Pengidentifikasi keamanan yang digunakan untuk mengident ifikasi grup AD secara unik.
db_role	Peran database yang akan dipetakan ke Grup AD yang ditentukan.
berat	Berat yang terkait dengan peran database.

Jika tidak ada argumen yang diberikan, semua pemetaan peran grup AD ke DB disetel ulang. Entah semua argumen perlu disediakan atau tidak sama sekali.

Jenis pengembalian

None

Catatan penggunaan

Panggil fungsi ini untuk menghapus grup AD tertentu ke pemetaan peran DB atau untuk mengatur ulang semua pemetaan. Fungsi ini hanya dapat dijalankan pada instance DB utama dari cluster DB oleh pengguna yang memiliki `rds_superuser` hak istimewa.

Contoh-contoh

```
postgres=> select * from pgadmap_read_mapping();

   ad_sid                       | pg_role      | weight      | ad_grp
--------------------------------+--------------+-------------+-------------------
 S-1-2-33-12345-67890-12345-678 | accounts-role| 10          | accounts-group
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10          | sales-group

(2 rows)
postgres=> select pgadmap_reset_mapping('S-1-2-33-12345-67890-12345-678', 'accounts-
role', 10);
```

```
pgadmap_reset_mapping
(1 row)

postgres=> select * from pgadmap_read_mapping();

   ad_sid                        | pg_role      | weight      | ad_grp
---------------------------------+--------------+-------------+---------------
 S-1-2-33-12345-67890-12345-666 | sales-role   | 10          | sales-group

(1 row)
postgres=> select pgadmap_reset_mapping();

pgadmap_reset_mapping
(1 row)

postgres=> select * from pgadmap_read_mapping();

   ad_sid                        | pg_role      | weight      | ad_grp
---------------------------------+--------------+-------------+--------------
 (0 rows)
```

Memigrasikan data ke Amazon Aurora dengan kompatibilitas PostgreSQL

Anda memiliki beberapa opsi untuk memigrasikan data dari basis data yang ada ke klaster DB Amazon Aurora Edisi Kompatibel PostgreSQL. Opsi migrasi Anda juga bergantung pada basis data asal migrasi Anda dan ukuran data yang Anda migrasikan. Berikut adalah pilihan Anda:

Memigrasikan instans DB RDS for PostgreSQL menggunakan snapshot

Anda dapat memigrasikan data secara langsung dari snapshot DB RDS for PostgreSQL ke klaster DB Aurora PostgreSQL.

Memigrasikan instans DB RDS for PostgreSQL menggunakan replika baca Aurora

Anda juga dapat memigrasikan dari instans DB RDS for PostgreSQL dengan membuat replika baca Aurora PostgreSQL pada instans DB RDS for PostgreSQL. Jika lag replika antara instans DB RDS for PostgreSQL dan replika baca Aurora PostgreSQL adalah nol, Anda dapat menghentikan replikasi. Pada titik ini, Anda dapat membuat replika baca Aurora menjadi klaster DB Aurora PostgreSQL mandiri untuk membaca dan menulis.

[Mengimpor data dari Amazon S3 ke Aurora PostgreSQL]

Anda dapat memigrasikan data dengan mengimpornya dari Amazon S3 ke tabel milik klaster DB Aurora PostgreSQL.

Migrasi dari basis data yang tidak kompatibel dengan PostgreSQL

Anda dapat menggunakan AWS Database Migration Service (AWS DMS) untuk memigrasikan data dari database yang tidak kompatibel dengan PostgreSQL. Untuk informasi selengkapnya AWS DMS, lihat Apa itu AWS Database Migration Service? dalam AWS Database Migration Service User Guide.

> **Note**
>
> Mengaktifkan otentikasi Kerberos saat ini tidak didukung di klaster Aurora PostgreSQL DB selama migrasi dari RDS untuk PostgreSQL. Anda dapat mengaktifkan otentikasi Kerberos hanya pada cluster Aurora PostgreSQL DB mandiri.

Untuk daftar Wilayah AWS di mana Aurora tersedia, lihat Amazon Aurora di. Referensi Umum AWS

> **Important**
>
> Jika Anda berencana untuk memigrasikan instans DB RDS for PostgreSQL ke klaster DB Aurora PostgreSQL dalam waktu dekat, kami sangat menyarankan agar Anda menonaktifkan peningkatan versi minor otomatis untuk instans DB di awal tahap perencanaan migrasi. Migrasi ke Aurora PostgreSQL mungkin tertunda jika versi RDS for PostgreSQL belum didukung oleh Aurora PostgreSQL.
> Untuk informasi tentang versi Aurora PostgreSQL, lihat Versi mesin untuk Amazon Aurora PostgreSQL.

Memigrasikan snapshot instans DB RDS for PostgreSQL ke klaster DB Aurora PostgreSQL

Untuk membuat klaster DB Aurora PostgreSQL, Anda dapat memigrasikan snapshot DB dari instans DB RDS for PostgreSQL. Klaster DB Aurora PostgreSQL baru akan diisi dengan data dari instans DB

RDS for PostgreSQL asli. Untuk informasi tentang membuat snapshot DB, lihat Membuat snapshot DB.

Dalam beberapa kasus, snapshot DB mungkin tidak berada di Wilayah AWS tempat Anda ingin menemukan data Anda. Jika demikian, gunakan konsol Amazon RDS untuk menyalin snapshot DB ke Wilayah AWS tersebut. Untuk informasi tentang menyalin snapshot DB, lihat Menyalin snapshot DB.

Anda dapat memigrasikan snapshot RDS for PostgreSQL yang kompatibel dengan versi Aurora PostgreSQL yang tersedia di Wilayah AWS yang ditentukan. Misalnya, Anda dapat memigrasikan snapshot dari instans DB RDS for PostgreSQL 11.1 ke Aurora PostgreSQL versi 11.4, 11.7, 11.8, atau 11.9 di Wilayah AS Barat (California Utara). Anda dapat memigrasikan snapshot RDS for PostgreSQL 10.11 ke Aurora PostgreSQL 10.11, 10.12, 10.13, dan 10.14. Dengan kata lain, snapshot RDS for PostgreSQL harus menggunakan versi minor yang sama dengan atau lebih rendah dari Aurora PostgreSQL.

Anda juga dapat memilih klaster DB Aurora PostgreSQL baru Anda untuk dienkripsi saat diam dengan menggunakan AWS KMS key. Opsi ini hanya tersedia untuk snapshot DB yang tidak dienkripsi.

Untuk memigrasikan snapshot RDS untuk PostgreSQL DB ke cluster DB PostgreSQL Aurora, Anda dapat menggunakan, API, atau RDS. AWS Management Console AWS CLI Saat Anda menggunakan AWS Management Console, konsol mengambil tindakan yang diperlukan untuk membuat cluster DB dan instance utama.

Konsol

Untuk memigrasikan snapshot DB PostgreSQL menggunakan konsol RDS

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Snapshot.
3. Di halaman Snapshot, pilih snapshot RDS for PostgreSQL yang ingin Anda migrasikan ke klaster DB Aurora PostgreSQL.
4. Pilih Tindakan, lalu pilih Migrasikan snapshot.
5. Tetapkan nilai-nilai berikut pada halaman Migrasikan basis data:

    - Versi mesin DB: Pilih versi mesin DB yang ingin Anda gunakan untuk instans baru yang dimigrasikan.

- Pengidentifikasi instans DB: Masukkan nama untuk cluster DB yang unik untuk akun Anda dalam Wilayah AWS yang Anda pilih. Pengidentifikasi ini digunakan di alamat titik akhir untuk instans di klaster DB Anda. Anda dapat memilih untuk menambahkan beberapa kecerdasan pada nama, seperti menyertakan mesin Wilayah AWS dan DB yang Anda pilih, misalnya**`aurora-cluster1`**.

  Pengenal instans DB memiliki batasan berikut:

  - Pengidentifikasi ini harus berisi 1-63 karakter alfanumerik atau tanda hubung.
  - Karakter pertamanya harus berupa huruf.
  - Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut.
  - Pengidentifikasi ini harus unik untuk semua instans DB per akun AWS , per Wilayah AWS.
- Kelas instans DB: Pilih kelas instans DB yang memiliki penyimpanan dan kapasitas yang diperlukan untuk basis data Anda, misalnya `db.r6g.large`. Volume klaster Aurora secara otomatis bertambah seiring jumlah data dalam basis data Anda meningkat. Jadi, Anda hanya perlu memilih kelas instans DB yang memenuhi persyaratan penyimpanan Anda saat ini. Untuk informasi selengkapnya, lihat Gambaran umum penyimpanan Amazon Aurora.
- Cloud privat virtual (VPC): Jika memiliki VPC yang sudah ada, Anda dapat menggunakan VPC tersebut dengan klaster DB Aurora PostgreSQL dengan memilih pengidentifikasi VPC Anda, misalnya `vpc-a464d1c1`. Untuk informasi tentang pembuatan VPC, lihat Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4).

  Atau, Anda dapat memilih agar Amazon RDS membuat VPC untuk Anda dengan memilih Buat VPC baru.
- Grup subnet DB: Jika memiliki grup subnet yang sudah ada, Anda dapat menggunakan grup subnet tersebut dengan klaster DB Aurora PostgreSQL dengan memilih pengidentifikasi grup subnet Anda, misalnya, `gs-subnet-group1`.
- Akses publik: Pilih Tidak untuk menentukan bahwa instans di klaster DB Anda hanya dapat diakses oleh sumber daya di dalam VPC Anda. Pilih Ya untuk menentukan bahwa instans di klaster DB Anda dapat diakses oleh sumber daya di jaringan publik.

> **Note**
>
> Klaster DB produksi Anda mungkin tidak perlu berada di subnet publik karena hanya server aplikasi Anda yang memerlukan akses ke klaster DB Anda. Jika klaster DB Anda tidak perlu berada di subnet publik, tetapkan Akses publik menjadi Tidak.

- Grup keamanan VPC: Pilih grup keamanan VPC untuk mengizinkan akses ke basis data Anda.
- Ketesediaan Zona: Pilih Zona Ketersediaan guna meng-host instans primer untuk klaster DB Aurora PostgreSQL Anda. Agar Amazon RDS memilih Zona Ketersediaan untuk Anda, pilih Tidak Ada Preferensi.
- Port basis data: Masukkan port default yang akan digunakan ketika menghubungkan ke instans di klaster DB Aurora PostgreSQL. Default-nya adalah `5432`.

> **Note**
>
> Anda mungkin berada di balik firewall perusahaan yang tidak mengizinkan akses ke port default seperti port default PostgreSQL, 5432. Dalam hal ini, berikan nilai port yang diizinkan oleh firewall perusahaan Anda. Ingat nilai port tersebut nanti saat Anda terhubung ke klaster DB Aurora PostgreSQL.

- Aktifkan Enkripsi: Pilih Aktifkan Enkripsi untuk klaster DB Aurora PostgreSQL baru Anda agar dienkripsi saat diam. Pilih juga kunci KMS sebagai nilai AWS KMS key.
- Peningkatan versi minor otomatis: Pilih Aktifkan peningkatan versi minor otomatis agar klaster DB Aurora PostgreSQL Anda dapat menerima peningkatan versi minor engine DB PostgreSQL secara otomatis saat tersedia.

  Opsi Peningkatan versi minor otomatis hanya berlaku untuk peningkatan versi minor engine PostgreSQL untuk klaster DB Aurora PostgreSQL Anda. Opsi ini tidak berlaku untuk patch biasa yang diterapkan untuk menjaga stabilitas sistem.

6. Pilih Migrasikan untuk memigrasikan snapshot DB Anda.
7. Pilih Basis data untuk melihat klaster DB baru. Pilih klaster DB baru untuk memantau progres migrasi. Ketika migrasi selesai, Status untuk klaster adalah Tersedia. Pada tab Konektivitas & keamanan, Anda dapat menemukan titik akhir klaster yang digunakan untuk menghubungkan ke instans penulis primer klaster DB. Untuk informasi selengkapnya tentang koneksi ke klaster DB Aurora PostgreSQL, lihat Menghubungkan ke klaster DB Amazon Aurora.

AWS CLI

Menggunakan AWS CLI untuk memigrasikan RDS untuk PostgreSQL DB snapshot ke PostgreSQL Aurora PostgreSQL melibatkan dua perintah terpisah. AWS CLI Pertama, Anda menggunakan `restore-db-cluster-from-snapshot` AWS CLI perintah membuat cluster Aurora PostgreSQL DB baru. Anda kemudian menggunakan perintah `create-db-instance` untuk membuat instans DB primer di klaster baru untuk menyelesaikan migrasi. Prosedur berikut membuat klaster DB Aurora PostgreSQL dengan instans DB primer yang memiliki konfigurasi yang sama dengan instans DB yang digunakan untuk membuat snapshot.

Untuk memigrasikan snapshot DB RDS for PostgreSQL ke klaster DB Aurora PostgreSQL

1. Gunakan describe-db-snapshotsperintah untuk mendapatkan informasi tentang snapshot DB yang ingin Anda migrasikan. Anda dapat menentukan parameter `--db-instance-identifier` atau `--db-snapshot-identifier` dalam perintah. Jika Anda tidak menentukan salah satu parameter ini, Anda akan mendapatkan semua snapshot.

   ```
   aws rds describe-db-snapshots --db-instance-identifier <your-db-instance-name>
   ```

2. Perintah ini akan menghasilkan semua detail konfigurasi untuk setiap snapshot yang dibuat dari instans DB yang ditentukan. Pada output, temukan snapshot yang ingin Anda migrasikan dan temukan Amazon Resource Name (ARN)-nya. Untuk mempelajari selengkapnya tentang ARN Amazon RDS, lihat Amazon Relational Database Service (Amazon RDS). ARN terlihat serupa dengan output berikut.

   ```
   “DBSnapshotArn": "arn:aws:rds:aws-region:111122223333:snapshot:<snapshot_name>"
   ```

   Dalam output tersebut, Anda juga dapat menemukan detail konfigurasi untuk instans DB RDS for PostgreSQL, seperti versi mesin, penyimpanan yang dialokasikan, apakah instans DB dienkripsi atau tidak, dan sebagainya.

3. Gunakan perintah restore-db-cluster-from-snapshot untuk memulai migrasi. Tentukan parameter berikut:

   - `--db-cluster-identifier` – Nama yang ingin Anda berikan ke klaster DB Aurora PostgreSQL. Klaster Aurora DB ini adalah target untuk migrasi snapshot DB Anda.
   - `--snapshot-identifier` – Nama Amazon Resource Name (ARN) untuk snapshot DB yang akan dimigrasikan.
   - `--engine` – Tentukan `aurora-postgresql` untuk mesin klaster Aurora DB.

- `--kms-key-id` – Parameter opsional ini memungkinkan Anda membuat klaster DB Aurora PostgreSQL terenkripsi dari snapshot DB yang tidak terenkripsi. Hal ini juga memungkinkan Anda memilih kunci enkripsi untuk klaster DB yang berbeda dari kunci yang digunakan untuk snapshot DB.

> **Note**
>
> Anda tidak dapat membuat klaster DB Aurora PostgreSQL yang tidak terenkripsi dari snapshot DB terenkripsi.

Tanpa `--kms-key-id` parameter yang ditentukan seperti yang ditunjukkan berikut, AWS CLI perintah restore-db-cluster-from-snapshot membuat cluster Aurora PostgreSQL DB kosong yang dienkripsi menggunakan kunci yang sama dengan snapshot DB atau tidak dienkripsi jika snapshot DB sumber tidak dienkripsi.

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier cluster-name \
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-
snapshot-name \
    --engine aurora-postgresql
```

Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-cluster-identifier new_cluster ^
    --snapshot-identifier arn:aws:rds:aws-region:111122223333:snapshot:your-
snapshot-name ^
    --engine aurora-postgresql
```

4. Perintah ini menghasilkan detail tentang klaster DB Aurora PostgreSQL yang sedang dibuat untuk migrasi. Anda dapat memeriksa status cluster Aurora PostgreSQL DB dengan menggunakan perintah. describe-db-clusters AWS CLI

```
aws rds describe-db-clusters --db-cluster-identifier cluster-name
```

5. Ketika cluster DB menjadi “tersedia”, Anda menggunakan create-db-instanceperintah untuk mengisi klaster Aurora PostgreSQL DB dengan instans DB berdasarkan snapshot Amazon RDS DB Anda. Tentukan parameter berikut:

   - `--db-cluster-identifier` – Nama klaster DB Aurora PostgreSQL baru yang Anda buat pada langkah sebelumnya.
   - `--db-instance-identifier` – Nama yang ingin Anda berikan ke instans DB. Instans ini akan menjadi simpul primer dalam klaster DB Aurora PostgreSQL Anda.
   - `----db-instance-class` – Tentukan kelas instans DB yang akan digunakan. Pilih dari kelas instans DB yang didukung oleh versi Aurora PostgreSQL tempat Anda akan bermigrasi. Lihat informasi yang lebih lengkap di Jenis kelas instans DB dan Mesin DB yang didukung untuk kelas instans DB.
   - `--engine` – Tentukan `aurora-postgresql` untuk instans DB.

   Anda juga dapat membuat instans DB dengan konfigurasi yang berbeda dari snapshot DB sumber, dengan meneruskan opsi yang sesuai dalam `create-db-instance` AWS CLI perintah. Untuk informasi lebih lanjut, lihat create-db-instanceperintah.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-instance \
       --db-cluster-identifier cluster-name \
       --db-instance-identifier --db-instance-class db.instance.class \
       --engine aurora-postgresql
   ```

   Untuk Windows:

   ```
   aws rds create-db-instance ^
       --db-cluster-identifier cluster-name ^
       --db-instance-identifier --db-instance-class db.instance.class ^
       --engine aurora-postgresql
   ```

Ketika proses migrasi selesai, klaster Aurora PostgreSQL memiliki instans DB primer yang terisi.

Memigrasikan data dari instans DB RDS for PostgreSQL ke klaster DB Aurora PostgreSQL menggunakan replika baca Aurora

Anda dapat menggunakan instans DB RDS for PostgreSQL sebagai dasar untuk klaster DB Aurora PostgreSQL baru dengan menggunakan replika baca Aurora untuk proses migrasi. Opsi replika baca Aurora hanya tersedia untuk migrasi dalam akun Wilayah AWS dan akun yang sama, dan hanya tersedia jika Wilayah menawarkan versi Aurora PostgreSQL yang kompatibel untuk RDS Anda untuk instans PostgreSQL DB. Dengan kompatibel, yang kami maksud adalah bahwa versi Aurora PostgreSQL sama dengan versi RDS for PostgreSQL, atau versi minor yang lebih tinggi dalam kelompok versi mayor yang sama.

Misalnya, untuk menggunakan teknik ini dalam memigrasikan instans DB RDS for PostgreSQL 11.14, Wilayahnya harus menawarkan Aurora PostgreSQL versi 11.14 atau versi minor yang lebih tinggi dalam kelompok PostgreSQL versi 11.

Topik

- Gambaran umum tentang memigrasikan data menggunakan replika baca Aurora
- Bersiap untuk memigrasikan data dengan menggunakan replika baca Aurora
- Membuat replika baca Aurora
- Mempromosikan replika baca Aurora

Gambaran umum tentang memigrasikan data menggunakan replika baca Aurora

Migrasi dari instans DB RDS for PostgreSQL ke klaster DB Aurora PostgreSQL adalah prosedur multilangkah. Pertama, buat replika baca Aurora dari instans DB RDS for PostgreSQL sumber Anda. Tindakan ini adalah awal proses replikasi dari instans DB RDS for PostgreSQL Anda ke klaster DB tujuan khusus yang dikenal sebagai klaster Replika. Klaster Replika hanya terdiri dari replika baca Aurora (instans pembaca).

Setelah klaster Replika ada, Anda memantau lag antara klaster Replika ini dan instans DB RDS for PostgreSQL sumber. Ketika lag replika adalah nol (0), Anda dapat mempromosikan klaster Replika. Replikasi akan berhenti, klaster Replika akan dipromosikan ke klaster DB Aurora mandiri, dan pembaca akan dipromosikan menjadi instans penulis untuk klaster. Anda kemudian dapat menambahkan instans ke klaster DB Aurora PostgreSQL untuk mengukur klaster DB Aurora PostgreSQL Anda untuk kasus penggunaan Anda. Anda juga dapat menghapus instans DB RDS for PostgreSQL jika Anda tidak membutuhkannya lagi.

> **Note**
>
> Diperlukan beberapa jam per terabyte data agar migrasi selesai.

Anda tidak dapat membuat replika baca Aurora jika instans DB RDS for PostgreSQL Anda sudah memiliki replika baca Aurora atau jika memiliki replika baca lintas Wilayah.

Bersiap untuk memigrasikan data dengan menggunakan replika baca Aurora

Selama proses migrasi menggunakan replika baca Aurora, pembaruan yang dilakukan pada instans DB RDS for PostgreSQL sumber akan direplikasi secara asinkron ke replika baca Aurora dari klaster Replika. Proses ini menggunakan fungsionalitas replikasi aliran native PostgreSQL yang menyimpan segmen write-ahead log (WAL) pada instans sumber. Sebelum memulai proses migrasi ini, pastikan bahwa instans Anda memiliki kapasitas penyimpanan yang memadai dengan memeriksa nilai metrik yang tercantum dalam tabel.

Metrik	Deskripsi
`FreeStorageSpace`	Ruang penyimpanan yang tersedia. Unit: Byte
`OldestReplicationSlotLag`	Ukuran lag untuk data WAL di replika yang mengalami lag paling lama. Unit: Megabyte
`RDSToAuroraPostgreSQLReplicaLag`	Jumlah waktu dalam hitungan detik ketika klaster DB Aurora PostgreSQL mengalami lag di belakang instans DB RDS sumber.
`TransactionLogsDiskUsage`	Ruang disk yang digunakan oleh log transaksi. Unit: Megabyte

Untuk informasi selengkapnya tentang pemantauan instans RDS Anda, lihat Pemantauan dalam Panduan Pengguna Amazon RDS.

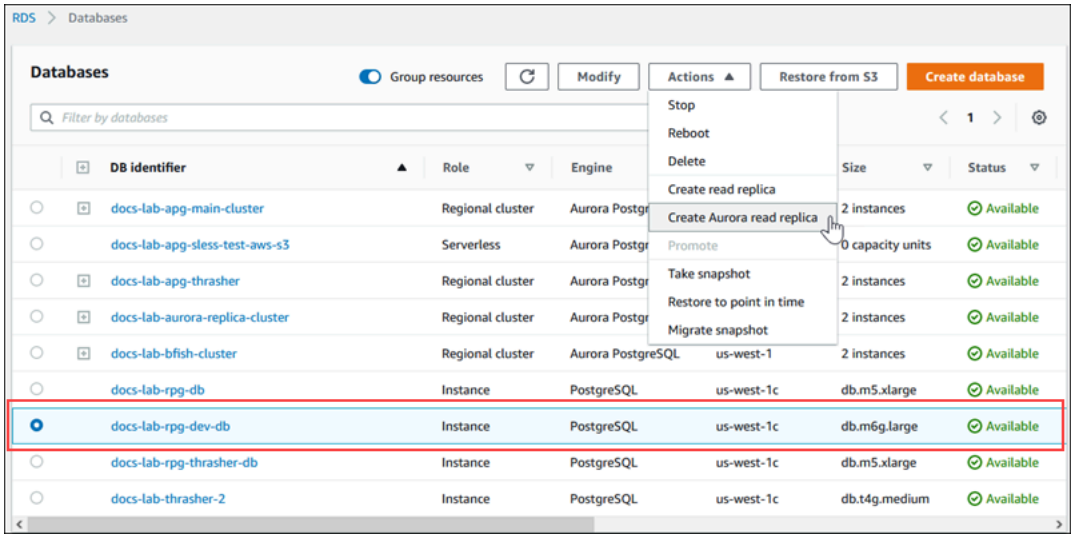
Membuat replika baca Aurora

Anda dapat membuat replika baca Aurora untuk RDS untuk instance PostgreSQL DB dengan menggunakan atau. AWS Management Console AWS CLI Opsi untuk membuat replika baca Aurora menggunakan hanya tersedia jika Wilayah AWS menawarkan versi AWS Management Console Aurora PostgreSQL yang kompatibel. Artinya, opsi ini hanya tersedia jika ada versi Aurora PostgreSQL yang sama dengan versi RDS for PostgreSQL atau versi minor yang lebih tinggi dalam kelompok versi mayor yang sama.

Konsol

Untuk membuat replika baca Aurora dari instans DB PostgreSQL sumber

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih instans DB RDS for PostgreSQL yang ingin Anda gunakan sebagai sumber untuk replika baca Aurora Anda. Untuk Tindakan, pilih Buat replika baca Aurora. Jika pilihan ini tidak ditampilkan, itu berarti versi Aurora PostgreSQL yang kompatibel tidak tersedia di Wilayah tersebut.



4. Pada halaman pengaturan "Buat replika baca Aurora", Anda perlu mengonfigurasi properti untuk klaster DB Aurora PostgreSQL seperti yang ditunjukkan pada tabel berikut. Klaster DB Replika dibuat dari snapshot instans DB sumber menggunakan nama pengguna dan kata sandi 'master' yang sama dengan sumbernya, sehingga Anda tidak dapat mengubahnya saat ini.

Opsi	Deskripsi
Kelas instans DB	Pilih kelas instans DB yang memenuhi persyaratan pemrosesan dan memori untuk instans primer dalam klaster DB. Untuk informasi selengkapnya, lihat Kelas instans DB Aurora.
Deployment Multi-AZ	Tidak tersedia selama migrasi
Pengidentifikasi instans DB	Masukkan nama yang ingin Anda berikan ke instans DB. Pengidentifikasi ini digunakan dalam alamat titik akhir untuk instans primer klaster DB baru. Pengidentifikasi instans DB memiliki batasan berikut: • Pengidentifikasi ini harus berisi 1-63 karakter alfanumerik atau tanda hubung. • Karakter pertamanya harus berupa huruf. • Pengidentifikasi ini tidak boleh diakhiri dengan tanda hubung atau mengandung dua tanda hubung berturut-turut. • Ini harus unik untuk semua instans DB untuk setiap AWS akun, untuk masing-masing Wilayah AWS.
Cloud Privat Virtual (VPC)	Pilih VPC untuk meng-host klaster DB. Pilih Buat VPC baru agar Amazon RDS membuatkan VPC untuk Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.
Grup subnet DB	Pilih grup subnet DB yang akan digunakan untuk klaster DB. Pilih Buat Grup Subnet DB baru agar Amazon RDS membuatkan grup subnet DB untuk Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.

Opsi	Deskripsi
Aksesibilitas publik	Pilih Ya untuk memberi klaster DB alamat IP publik. Atau, pilih Tidak. Instans di klaster DB Anda dapat menjadi perpaduan dari instans DB publik dan privat. Untuk informasi selengkapnya tentang menyembun yikan instans dari akses publik, lihat Menyembunyikan instans DB dalam VPC dari internet.
Zona ketersediaan	Tentukan apakah Anda ingin menentukan Zona Ketersediaan tertentu. Untuk informasi selengkapnya tentang Zona Ketersediaan, lihat Wilayah dan Zona Ketersediaan.
Grup keamanan VPC	Pilih satu atau beberapa grup keamanan VPC untuk mengamankan akses jaringan ke klaster DB. Pilih Buat grup keamanan VPC baru agar Amazon RDS membuatkan grup keamanan VPC untuk Anda. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.
Port basis data	Tentukan port untuk aplikasi dan utilitas yang akan digunakan untuk mengakses basis data. Klaster DB Aurora PostgreSQL akan ditetapkan secara default ke port default PostgreSQL, 5432. Firewall di beberapa perusahaan memblokir koneksi ke port ini. Jika firewall perusahaan Anda memblokir port default ini, pilih port lain untuk klaster DB baru.
Grup parameter DB	Pilih grup parameter DB untuk klaster DB Aurora PostgreSQL. Aurora memiliki grup parameter DB default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter DB Anda sendiri. Untuk informasi selengkapnya tentang grup parameter DB, lihat Bekerja dengan grup parameter.

Opsi	Deskripsi
Grup parameter klaster DB	Pilih grup parameter klaster DB untuk klaster DB Aurora PostgreSQL. Aurora memiliki grup parameter klaster DB default yang dapat Anda gunakan, atau Anda dapat membuat grup parameter klaster DB Anda sendiri. Untuk informasi selengkapnya tentang grup parameter klaster DB, lihat Bekerja dengan grup parameter.
Enkripsi	Pilih Aktifkan Enkripsi untuk klaster DB Aurora baru Anda agar dienkripsi saat diam. Jika Anda memilih Aktifkan enkripsi, pilih juga kunci KMS sebagai nilai AWS KMS key.
Prioritas	Pilih prioritas failover untuk klaster DB. Jika Anda tidak memilih nilai, nilai default-nya adalah tier-1. Prioritas ini akan menentukan urutan promosi Aurora Replika saat melakukan pemulihan dari kegagalan instans primer. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora.
Periode retensi cadangan	Pilih durasi waktu, dalam kisaran 1–35 hari, saat Aurora mempertahankan salinan cadangan basis data. Salinan cadangan dapat digunakan untuk point-in-time mengembalikan (PITR) database Anda hingga yang kedua.
Pemantauan yang ditingkatkan	Pilih Aktifkan pemantauan yang ditingkatkan untuk mengaktifkan metrik pengumpulan secara waktu nyata untuk sistem operasi tempat klaster DB Anda berjalan. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan.

Opsi	Deskripsi
Peran Pemantauan	Hanya tersedia jika Anda memilih Aktifkan pemantaua n yang ditingkatkan. Peran AWS Identity and Access Management (IAM) yang digunakan untuk Pemantaua n yang Ditingkatkan. Untuk informasi selengkapnya, lihat Menyiapkan dan mengaktifkan Pemantauan yang Ditingkatkan.
Granularitas	Hanya tersedia jika Anda memilih Aktifkan pemantauan yang ditingkatkan. Atur interval, dalam detik, di antara waktu pengumpulan metrik untuk klaster DB Anda.
Peningkatan versi minor otomatis	Pilih Ya agar klaster DB Aurora PostgreSQL Anda dapat menerima peningkatan versi minor engine DB PostgreSQL secara otomatis saat tersedia. Opsi Peningkatan versi minor otomatis hanya berlaku untuk peningkatan versi minor engine PostgreSQL untuk klaster DB Aurora PostgreSQL Anda. Opsi ini tidak berlaku untuk patch biasa yang diterapkan untuk menjaga stabilitas sistem.
Jendela pemeliharaan	Pilih rentang waktu mingguan untuk melakukan pemeliharaan sistem.

5. Pilih Buat replika baca.

AWS CLI

Untuk membuat replika baca Aurora dari sumber RDS untuk instance PostgreSQL DB dengan menggunakan AWS CLI, pertama-tama Anda menggunakan perintah CLI create-db-clusteruntuk membuat cluster DB Aurora kosong. Setelah klaster DB ada, Anda kemudian menggunakan perintah create-db-instance untuk membuat instans primer untuk klaster DB Anda. Instans primer adalah instans pertama yang dibuat di klaster DB Aurora. Dalam hal ini, instans tersebut awalnya dibuat sebagai replika baca Aurora pada instans DB RDS for PostgreSQL Anda. Saat prosesnya selesai, instans DB RDS for PostgreSQL Anda telah dimigrasikan secara efektif ke klaster DB Aurora PostgreSQL.

Anda tidak perlu menentukan akun pengguna primer (biasanya, `postgres`), kata sandinya, atau nama basis data. Replika baca Aurora memperolehnya secara otomatis dari sumber RDS untuk instance PostgreSQL DB yang Anda identifikasi saat Anda menjalankan perintah. AWS CLI

Anda perlu menentukan versi mesin yang akan digunakan untuk klaster DB dan instans DB Aurora PostgreSQL. Versi yang Anda tentukan harus cocok dengan instans DB RDS for PostgreSQL sumber. Jika instans DB RDS for PostgreSQL sumber dienkripsi, Anda juga perlu menentukan enkripsi untuk instans primer klaster DB Aurora PostgreSQL. Migrasi instans terenkripsi ke klaster DB Aurora yang tidak terenkripsi tidak didukung.

Contoh berikut membuat klaster DB Aurora PostgreSQL bernama `my-new-aurora-cluster` yang akan menggunakan instans DB RDS sumber yang tidak terenkripsi. Pertama-tama, Anda membuat klaster DB Aurora PostgreSQL dengan memanggil perintah CLI `create-db-cluster`. Contoh ini menunjukkan cara menggunakan parameter `--storage-encrypted` opsional untuk menentukan bahwa klaster DB harus dienkripsi. Karena sumber DB tidak dienkripsi, `--kms-key-id` akan digunakan untuk menentukan kunci yang akan digunakan. Untuk informasi selengkapnya tentang parameter wajib dan opsional, lihat contoh berikut.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster  \
    --db-cluster-identifier my-new-aurora-cluster \
    --db-subnet-group-name my-db-subnet
    --vpc-security-group-ids sg-11111111
    --engine aurora-postgresql \
    --engine-version same-as-your-rds-instance-version \
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db \
    --storage-encrypted \
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

Untuk Windows:

```
aws rds create-db-cluster  ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-subnet-group-name my-db-subnet ^
    --vpc-security-group-ids sg-11111111 ^
    --engine aurora-postgresql ^
    --engine-version same-as-your-rds-instance-version ^
```

```
    --replication-source-identifier arn:aws:rds:aws-region:111122223333:db/rpg-source-
db ^
    --storage-encrypted ^
    --kms-key-id arn:aws:kms:aws-
region:111122223333:key/11111111-2222-3333-444444444444
```

Dalam daftar berikut, Anda dapat menemukan informasi selengkapnya tentang beberapa opsi yang ditampilkan dalam contoh. Kecuali jika ditentukan lain, parameter ini wajib ada.

- `--db-cluster-identifier` – Anda harus memberikan nama untuk klaster DB Aurora PostgreSQL baru Anda.
- `--db-subnet-group-name` – Buat klaster DB Aurora PostgreSQL Anda di subnet DB yang sama dengan instans DB sumber.
- `--vpc-security-group-ids` – Tentukan grup keamanan untuk klaster DB Aurora PostgreSQL Anda.
- `--engine-version` – Tentukan versi yang akan digunakan untuk klaster DB Aurora PostgreSQL. Versi ini harus sama dengan versi yang digunakan oleh instans DB RDS for PostgreSQL sumber Anda.
- `--replication-source-identifier` – Identifikasi instans DB RDS for PostgreSQL menggunakan Amazon Resource Name (ARN). Untuk informasi selengkapnya tentang ARN Amazon RDS, lihat Amazon Relational Database Service (Amazon RDS) dalam Referensi Umum AWS untuk klaster DB Anda.
- `--storage-encrypted` – Opsional. Gunakan hanya jika diperlukan untuk menentukan enkripsi sebagai berikut:
  - Gunakan parameter ini ketika instans DB sumber memiliki penyimpanan terenkripsi. Panggilan ke `create-db-cluster` akan gagal jika Anda tidak menggunakan parameter ini dengan instans DB sumber yang memiliki penyimpanan terenkripsi. Jika Anda ingin menggunakan kunci untuk klaster DB Aurora PostgreSQL yang berbeda dari kunci yang digunakan oleh instans DB sumber, Anda juga perlu menentukan `--kms-key-id`.
  - Gunakan jika penyimpanan instans DB sumber tidak terenkripsi, tetapi Anda ingin klaster DB Aurora PostgreSQL menggunakan enkripsi. Jika demikian, Anda juga perlu mengidentifikasi kunci enkripsi yang akan digunakan dengan parameter `--kms-key-id`.
- `--kms-key-id` – Opsional. Saat digunakan, Anda dapat menentukan kunci yang akan digunakan untuk enkripsi penyimpanan (`--storage-encrypted`) dengan menggunakan ARN kunci, ID, ARN alias, atau nama aliasnya. Parameter ini diperlukan hanya untuk situasi-situasi berikut:

- Untuk memilih kunci untuk klaster DB Aurora PostgreSQL yang berbeda dari yang digunakan oleh instans DB sumber.
- Untuk membuat klaster terenkripsi dari sumber yang tidak terenkripsi. Dalam hal ini, Anda perlu menentukan kunci yang harus digunakan Aurora PostgreSQL untuk enkripsi.

Setelah membuat klaster DB Aurora PostgreSQL, Anda kemudian membuat instans primer dengan menggunakan perintah CLI `create-db-instance`, seperti yang ditunjukkan berikut ini:

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
    --db-cluster-identifier my-new-aurora-cluster \
    --db-instance-class db.x2g.16xlarge \
    --db-instance-identifier rpg-for-migration \
    --engine aurora-postgresql
```

Untuk Windows:

```
aws rds create-db-instance ^
    --db-cluster-identifier my-new-aurora-cluster ^
    --db-instance-class db.x2g.16xlarge ^
    --db-instance-identifier rpg-for-migration ^
    --engine aurora-postgresql
```

Dalam daftar berikut, Anda dapat menemukan informasi selengkapnya tentang beberapa opsi yang ditampilkan dalam contoh.

- `--db-cluster-identifier` – Tentukan nama klaster DB Aurora PostgreSQL yang Anda buat dengan perintah `create-db-instance` di langkah sebelumnya.
- `--db-instance-class` – Nama kelas instans DB yang akan digunakan untuk instans primer Anda, seperti `db.r4.xlarge`, `db.t4g.medium`, `db.x2g.16xlarge`, dan sebagainya. Untuk daftar kelas instans DB yang tersedia, lihat Jenis kelas instans DB.
- `--db-instance-identifier` – Tentukan nama yang akan diberikan untuk instans primer Anda.
- `--engine aurora-postgresql` – Tentukan `aurora-postgresql` untuk mesin.

API RDS

Untuk membuat replika baca Aurora dari instans DB RDS for PostgreSQL sumber, pertama-tama gunakan operasi API RDS `CreateDBCluster` untuk membuat klaster DB Aurora baru untuk replika baca Aurora yang dibuat dari instans DB RDS for PostgreSQL sumber Anda. Ketika klaster DB Aurora PostgreSQL tersedia, Anda kemudian menggunakan `CreateDBInstance` untuk membuat instans primer untuk klaster DB Aurora.

Anda tidak perlu menentukan akun pengguna primer (biasanya, `postgres`), kata sandinya, atau nama basis data. Replika baca Aurora akan memperolehnya secara otomatis dari instans DB RDS for PostgreSQL sumber yang ditentukan dengan `ReplicationSourceIdentifier`.

Anda perlu menentukan versi mesin yang akan digunakan untuk klaster DB dan instans DB Aurora PostgreSQL. Versi yang Anda tentukan harus cocok dengan instans DB RDS for PostgreSQL sumber. Jika instans DB RDS for PostgreSQL sumber dienkripsi, Anda juga perlu menentukan enkripsi untuk instans primer klaster DB Aurora PostgreSQL. Migrasi instans terenkripsi ke klaster DB Aurora yang tidak terenkripsi tidak didukung.

Untuk membuat klaster DB Aurora untuk replika baca Aurora, gunakan operasi API RDS `CreateDBCluster` dengan parameter berikut:

- `DBClusterIdentifier` – Nama klaster DB yang akan dibuat.
- `DBSubnetGroupName` – Nama grup subnet DB yang akan dikaitkan dengan klaster DB ini.
- `Engine=aurora-postgresql` – Nama mesin yang akan digunakan.
- `ReplicationSourceIdentifier` – Amazon Resource Name (ARN) untuk instans DB PostgreSQL sumber. Untuk informasi selengkapnya tentang ARN Amazon RDS, lihat Amazon Relational Database Service (Amazon RDS) dalam Referensi Umum Amazon Web Services. Jika `ReplicationSourceIdentifier` mengidentifikasi sumber terenkripsi, Amazon RDS akan menggunakan kunci KMS default Anda kecuali jika Anda menentukan kunci yang berbeda menggunakan opsi `KmsKeyId`.
- `VpcSecurityGroupIds` – Daftar grup keamanan VPC Amazon EC2 yang akan dikaitkan dengan klaster DB ini.
- `StorageEncrypted` – Menunjukkan bahwa klaster DB dienkripsi. Jika Anda menggunakan parameter ini tanpa menentukan `ReplicationSourceIdentifier`, Amazon RDS akan menggunakan kunci KMS default Anda.

- `KmsKeyId` – Kunci untuk klaster terenkripsi. Saat digunakan, Anda dapat menentukan kunci yang akan digunakan untuk enkripsi penyimpanan dengan menggunakan ARN kunci, ID, ARN alias, atau nama aliasnya.

Untuk informasi selengkapnya, lihat `CreateDBCluster` dalam Referensi API Amazon RDS.

Setelah klaster DB Aurora tersedia, Anda dapat membuat instans primer untuk klaster tersebut dengan menggunakan operasi API RDS `CreateDBInstance` dengan parameter berikut:

- `DBClusterIdentifier` – Nama klaster DB Anda.
- `DBInstanceClass` – Nama kelas instans DB yang akan digunakan untuk instans primer Anda.
- `DBInstanceIdentifier` – Nama instans primer Anda.
- `Engine=aurora-postgresql` – Nama mesin yang akan digunakan.

Untuk informasi selengkapnya, lihat `CreateDBInstance` dalam Referensi API Amazon RDS.

Mempromosikan replika baca Aurora

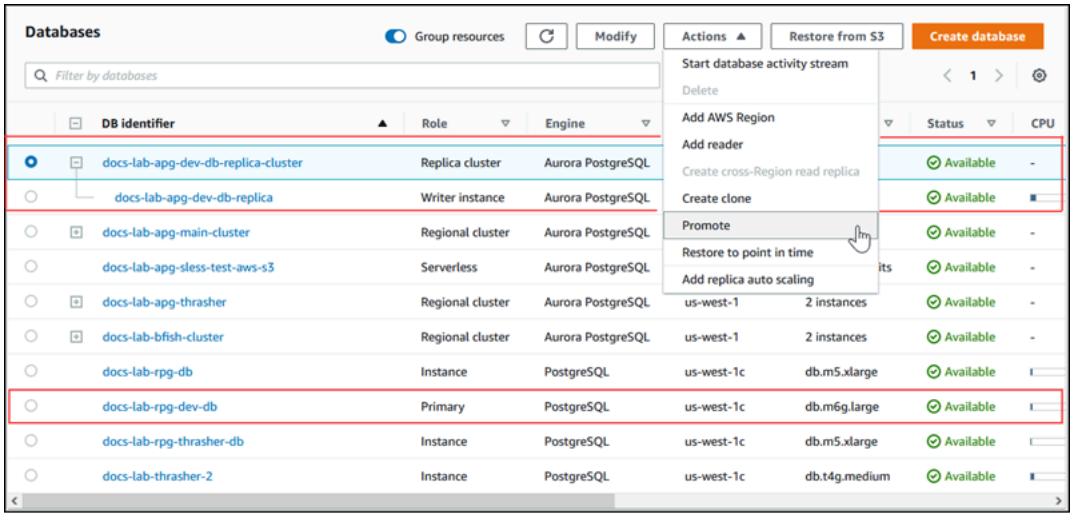
Migrasi ke Aurora PostgreSQL belum selesai sampai Anda mempromosikan klaster Replika, jadi jangan hapus instans DB RDS for PostgreSQL sumber.

Sebelum mempromosikan klaster Replika, pastikan bahwa instans DB RDS for PostgreSQL tidak memiliki transaksi dalam proses atau aktivitas lain yang menulis ke basis data. Saat lag replika pada replika baca Aurora mencapai nol (0), Anda dapat mempromosikan klaster Replika. Untuk informasi selengkapnya tentang pemantauan lag replika, lihat Memantau replikasi Aurora PostgreSQL dan Metrik tingkat instans untuk Amazon Aurora.

Konsol

Untuk mempromosikan replika baca Aurora ke klaster DB Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster Replika.



4. Untuk Tindakan, pilih Promosikan. Hal ini mungkin memerlukan waktu beberapa menit dan dapat menyebabkan waktu henti.

Ketika prosesnya selesai, klaster Aurora Replika adalah klaster DB Aurora PostgreSQL Regional, dengan instans Penulis yang berisi data dari instans DB RDS for PostgreSQL.

AWS CLI

Untuk mempromosikan replika baca Aurora ke cluster DB yang berdiri sendiri, gunakan perintah. promote-read-replica-db-cluster AWS CLI

Example

Untuk Linux, macOS, atau Unix:

```
aws rds promote-read-replica-db-cluster \
    --db-cluster-identifier myreadreplicacluster
```

Untuk Windows:

```
aws rds promote-read-replica-db-cluster ^
    --db-cluster-identifier myreadreplicacluster
```

API RDS

Untuk mempromosikan replika baca Aurora ke cluster DB yang berdiri sendiri, gunakan operasi RDS API DBCluster. PromoteReadReplica

Setelah mempromosikan klaster Replika, Anda dapat mengonfirmasi bahwa promosinya telah selesai dengan memeriksa log peristiwa sebagai berikut.

Untuk mengonfirmasi bahwa klaster Replika Aurora telah dipromosikan

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Peristiwa.
3. Pada halaman Peristiwa, temukan nama klaster Anda di daftar Sumber. Setiap peristiwa memiliki sumber, jenis, waktu, dan pesan. Anda dapat melihat semua peristiwa yang terjadi di Wilayah AWS untuk akun Anda. Promosi yang sukses menghasilkan pesan berikut.

```
Promoted Read Replica cluster to a stand-alone database cluster.
```

Setelah promosi selesai, instans DB RDS for PostgreSQL sumber dan klaster DB Aurora PostgreSQL akan menjadi tidak tertaut. Anda dapat mengarahkan aplikasi klien Anda ke titik akhir untuk replika baca Aurora. Untuk informasi selengkapnya tentang titik akhir Aurora, lihat Manajemen koneksi Amazon Aurora. Pada tahap ini, Anda dapat menghapus instans DB dengan aman.

Meningkatkan performa kueri untuk Aurora PostgreSQL dengan Aurora Optimized Reads

Anda dapat mencapai pemrosesan kueri yang lebih cepat untuk Aurora PostgreSQL dengan Aurora Optimized Reads. Instans DB Aurora PostgreSQL yang menggunakan Aurora Optimized Reads memberikan latensi kueri hingga 8x lebih baik dan penghematan biaya hingga 30% untuk aplikasi dengan set data besar, yang melebihi kapasitas memori instans DB.

Topik

- Gambaran umum Aurora Optimized Reads di PostgreSQL
- Menggunakan Aurora Optimized Reads
- Kasus penggunaan untuk Aurora Optimized Reads
- Memantau instans DB yang menggunakan Aurora Optimized Reads
- Praktik terbaik untuk Aurora Optimized Reads

Gambaran umum Aurora Optimized Reads di PostgreSQL

Aurora Optimized Reads tersedia secara default saat Anda membuat klaster DB yang menggunakan instans R6gd berbasis Graviton dan R6id berbasis Intel dengan penyimpanan non-volatile memory express (NVMe). Fitur ini tersedia dari versi PostgreSQL berikut:

- 16.1 dan semua versi yang lebih tinggi
- 15.4 dan versi yang lebih tinggi
- 14.9 dan versi yang lebih tinggi

Aurora Optimized Reads mendukung dua kemampuan: cache berjenjang dan objek sementara.

Cache berjenjang yang didukung Optimized Reads - Menggunakan cache berjenjang, Anda dapat memperluas kapasitas caching instans DB hingga 5x memori instans. Kemampuan ini secara otomatis mempertahankan cache untuk menyimpan data terbaru yang konsisten secara transaksional, sehingga membebaskan aplikasi dari overhead dalam mengelola keterkinian data dalam solusi caching berbasis hasil set eksternal. Kemampuan ini menawarkan latensi hingga 8x lebih baik untuk kueri yang sebelumnya mengambil data dari penyimpanan Aurora.

Di Aurora, nilai untuk `shared_buffers` dalam kelompok parameter default biasanya diatur ke sekitar 75% dari memori yang tersedia. Namun, untuk jenis instans r6gd dan r6id, Aurora akan mengurangi `shared_buffers` ruang sebesar 4,5% untuk meng-host metadata untuk cache Bacaan yang Dioptimalkan.

Objek sementara yang didukung Optimized Reads - Menggunakan objek sementara, Anda dapat mencapai pemrosesan kueri yang lebih cepat dengan menempatkan file sementara yang dihasilkan oleh PostgreSQL pada penyimpanan NVMe lokal. Kemampuan ini mengurangi lalu lintas ke Elastic Block Storage (EBS) melalui jaringan. Ini menawarkan latensi dan throughput hingga 2x lebih baik untuk kueri lanjutan yang mengurutkan, menggabungkan, atau menggabungkan volume besar data yang tidak sesuai dengan kapasitas memori yang tersedia pada instans DB.

Pada klaster Aurora I/O Dioptimalkan, Optimized Reads menggunakan cache berjenjang dan objek sementara pada penyimpanan NVMe. Dengan kemampuan cache berjenjang yang didukung Optimized Reads, Aurora mengalokasikan 2x memori instans untuk objek sementara, sekitar 10% penyimpanan untuk operasi internal, dan penyimpanan yang tersisa sebagai cache berjenjang. Pada klaster Standar Aurora, Optimized Reads hanya menggunakan objek sementara.

Mesin	Konfigurasi penyimpanan klaster	Objek sementara yang didukung Optimized Reads	Cache berjenjang yang didukung Optimized Reads	Versi yang didukung
yang kompatibel dengan Aurora PostgreSQL	Standar	Ya	Tidak	Aurora PostgreSQL versi 16.1 dan semua versi yang lebih tinggi, 15.4 dan lebih tinggi, versi 14.9 dan lebih tinggi
	I/O Dioptimalkan	Ya	Ya	

> **Note**
>
> Peralihan antara klaster I/O Dioptimalkan dan Standar pada kelas instans DB berbasis NVMe akan menyebabkan pengaktifan ulang mesin basis data langsung.

Di Aurora PostgreSQL, gunakan `temp_tablespaces` parameter untuk mengkonfigurasi ruang tabel tempat objek sementara disimpan.

Untuk memeriksa apakah objek sementara dikonfigurasi, gunakan perintah berikut:

```
postgres=> show temp_tablespaces;
temp_tablespaces
---------------------
aurora_temp_tablespace
(1 row)
```

`aurora_temp_tablespace` adalah ruang tabel yang dikonfigurasi oleh Aurora yang menunjuk ke penyimpanan lokal NVMe. Anda tidak dapat mengubah parameter ini atau beralih kembali ke penyimpanan Amazon EBS.

Untuk memeriksa apakah cache baca yang dioptimalkan diaktifkan, gunakan perintah berikut:

```
postgres=> show shared_preload_libraries;
```

```
                shared_preload_libraries
-----------------------------------------------------
rdsutils,pg_stat_statements,aurora_optimized_reads_cache
```

Menggunakan Aurora Optimized Reads

Saat Anda menyediakan instans DB PostgreSQL Aurora dengan salah satu instans DB berbasis NVME, instans DB secara otomatis menggunakan Aurora Optimized Reads.

Untuk mengaktifkan Aurora Optimized Reads, lakukan salah satu hal berikut ini:

- Buat klaster DB Aurora PostgreSQL menggunakan salah satu kelas instans DB berbasis NVMe. Untuk informasi selengkapnya, lihat Membuat klaster DB Amazon Aurora.
- Ubah klaster DB Aurora PostgreSQL yang ada untuk menggunakan salah satu kelas instans DB berbasis NVMe. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora.

Bacaan yang Dioptimalkan Aurora tersedia di semua Wilayah AWS tempat di mana satu atau lebih kelas instans DB dengan penyimpanan SSD NVMe lokal didukung. Untuk informasi selengkapnya, lihat Kelas instans DB Aurora.

Untuk beralih kembali ke instance Aurora pembacaan yang tidak dioptimalkan, ubah kelas instans DB instance Aurora Anda ke kelas instance serupa tanpa penyimpanan sementara NVMe untuk beban kerja database Anda. Misalnya, jika kelas instans DB saat ini adalah db.r6gd.4xlarge, pilih db.r6g.4xlarge untuk beralih kembali. Untuk informasi selengkapnya, lihat Memodifikasi instans Aurora DB.

Kasus penggunaan untuk Aurora Optimized Reads

Cache berjenjang yang didukung Optimized Reads

Berikut ini adalah beberapa kasus penggunaan yang dapat memperoleh manfaat dari Optimized Reads dengan cache berjenjang:

- Aplikasi skala internet seperti pemrosesan pembayaran, penagihan, e-commerce dengan SLA performa yang ketat.
- Dasbor pelaporan waktu nyata yang menjalankan ratusan kueri titik untuk pengumpulan metrik/data.
- Aplikasi AI generatif dengan ekstensi pgvector untuk mencari neighbor yang tepat atau terdekat dalam jutaan penyematan vektor.

Objek sementara yang didukung Optimized Reads

Berikut ini adalah beberapa kasus penggunaan yang dapat memperoleh manfaat dari Optimized Reads dengan objek sementara:

- Kueri analitis yang mencakup Ekspresi Tabel Umum (CTE), tabel turunan, dan operasi pengelompokan.
- Replika baca yang menangani kueri yang tidak dioptimalkan untuk aplikasi.
- Kueri pelaporan sesuai permintaan atau dinamis dengan operasi kompleks seperti GROUP BY dan ORDER BY yang tidak selalu dapat menggunakan indeks yang sesuai.
- `CREATE INDEX`atau `REINDEX` operasi untuk menyortir.
- Beban kerja lain yang menggunakan tabel sementara internal.

Memantau instans DB yang menggunakan Aurora Optimized Reads

Anda dapat memantau kueri yang menggunakan cache berjenjang yang didukung Optimized Reads dengan perintah EXPLAIN seperti yang ditunjukkan pada contoh berikut:

```
Postgres=> EXPLAIN (ANALYZE, BUFFERS) SELECT c FROM sbtest15 WHERE id=100000000


QUERY PLAN
------------------------------------------------------------------------------------------
 Index Scan using sbtest15_pkey on sbtest15  (cost=0.57..8.59 rows=1 width=121) (actual
 time=0.287..0.288 rows=1 loops=1)
   Index Cond: (id = 100000000)
   Buffers: shared hit=3 read=2 aurora_orcache_hit=2
   I/O Timings: shared/local read=0.264
 Planning:
   Buffers: shared hit=33 read=6 aurora_orcache_hit=6
   I/O Timings: shared/local read=0.607
 Planning Time: 0.929 ms
 Execution Time: 0.303 ms
(9 rows)
Time: 2.028 ms
```

> **Note**
>
> `aurora_orcache_hit`dan `aurora_storage_read` bidang di `Buffers` bagian rencana penjelasan hanya ditampilkan ketika Pembacaan yang Dioptimalkan diaktifkan dan nilainya lebih besar dari nol. Bidang baca adalah total `aurora_storage_read` bidang `aurora_orcache_hit` dan.

Anda dapat memantau instans DB yang menggunakan Bacaan yang Dioptimalkan Aurora menggunakan metrik berikut CloudWatch:

- `AuroraOptimizedReadsCacheHitRatio`
- `FreeEphemeralStorage`
- `ReadIOPSEphemeralStorage`
- `ReadLatencyEphemeralStorage`
- `ReadThroughputEphemeralStorage`
- `WriteIOPSEphemeralStorage`
- `WriteLatencyEphemeralStorage`
- `WriteThroughputEphemeralStorage`

Metrik ini menyediakan data tentang penyimpanan instans, IOPS, dan throughput yang tersedia. Untuk informasi selengkapnya tentang metrik ini, lihat Metrik tingkat instans untuk Amazon Aurora.

Anda juga dapat menggunakan ekstensi `pg_proctab` untuk memantau penyimpanan NVMe.

```
postgres=>select * from pg_diskusage();

major | minor |       devname       | reads_completed | reads_merged | sectors_read |
 readtime | writes_completed | writes_merged | sectors_written | writetime | current_io
 | iotime  | totaliotime
------+-------+---------------------+-----------------+--------------+--------------
+----------+------------------+---------------+-----------------+-----------
+------------+---------+-------------
      |       | rdstemp             |           23264 |            0 |       191450 |
  11670 |          1750892 |             0 |        24540576 |    819350 |          0 |
 3847580 |      831020
```

```
      |       | rdsephemeralstorage |           23271 |            0 |       193098 |
   2620 |           114961 |             0 |        13845120 |    130770 |          0 |
  215010 |      133410
(2 rows)
```

Praktik terbaik untuk Aurora Optimized Reads

Gunakan praktik terbaik berikut untuk Aurora Optimized Reads:

- Pantau ruang penyimpanan yang tersedia di penyimpanan instans dengan CloudWatch metrik`FreeEphemeralStorage`. Jika penyimpanan instance mencapai batasnya karena beban kerja pada instans DB, atur konkurensi dan kueri yang banyak menggunakan objek sementara atau memodifikasinya untuk menggunakan kelas instans DB yang lebih besar.
- Pantau CloudWatch metrik untuk hit rate cache Optimized Reads. Operasi seperti VACUUM akan memodifikasi sejumlah besar blok dengan sangat cepat. Hal ini dapat menyebabkan penurunan sementara dalam rasio hit. Ekstensi `pg_prewarm` dapat digunakan untuk memuat data ke dalam cache buffer yang memungkinkan Aurora secara proaktif menulis beberapa blok tersebut ke cache Optimized Reads.
- Anda dapat mengaktifkan manajemen cache klaster (CCM) untuk melakukan "warm up" cache buffer dan cache berjenjang pada pembaca tingkat 0, yang akan digunakan sebagai target failover. Ketika CCM diaktifkan, cache buffer dipindai secara berkala untuk menulis halaman yang memenuhi syarat untuk pengosongan dalam cache berjenjang. Untuk informasi selengkapnya tentang CCM, lihat Pemulihan cepat setelah failover dengan manajemen cache klaster untuk Aurora PostgreSQL.

Menggunakan Babelfish for Aurora PostgreSQL

Babelfish for Aurora PostgreSQL memperluas klaster DB Aurora PostgreSQL Anda dengan kemampuan untuk menerima koneksi basis data dari klien SQL Server. Dengan Babelfish, aplikasi yang awalnya dibuat untuk SQL Server dapat bekerja secara langsung dengan Aurora PostgreSQL dengan sedikit perubahan kode dibandingkan dengan migrasi tradisional dan tanpa mengubah driver basis data. Untuk informasi selengkapnya tentang bermigrasi, lihat Memigrasi basis data SQL Server ke Babelfish for Aurora PostgreSQL.

Babelfish menyediakan titik akhir tambahan untuk klaster basis data Aurora PostgreSQL yang memungkinkannya untuk memahami protokol tingkat kabel SQL Server dan pernyataan SQL Server yang umum digunakan. Aplikasi klien yang menggunakan protokol kabel Tabular Data Stream (TDS) dapat terhubung secara native ke port pendengar TDS di Aurora PostgreSQL. Untuk mempelajari lebih lanjut tentang TDS, lihat [MS-TDS]: Tabular Data Stream Protocol di situs web Microsoft.

> **Note**
>
> Babelfish for Aurora PostgreSQL mendukung TDS versi 7.1 hingga 7.4.

Babelfish juga menyediakan akses ke data menggunakan koneksi PostgreSQL. Secara default, kedua dialek SQL yang didukung oleh Babelfish tersedia melalui protokol kabel aslinya di port berikut:

- Dialek SQL Server (T-SQL), klien terhubung ke port 1433.
- Dialek PostgreSQL (PL/pgSQL), klien terhubung ke port 5432.

Babelfish menjalankan bahasa Transact-SQL (T-SQL) dengan beberapa perbedaan. Untuk informasi selengkapnya, lihat Perbedaan antara Babelfish for Aurora PostgreSQL dan SQL Server.

Pada bagian berikut, Anda dapat menemukan informasi tentang pengaturan dan penggunaan klaster DB Babelfish for Aurora PostgreSQL.

Topik

- Batasan Babelfish
- Memahami konfigurasi dan arsitektur Babelfish
- Membuat klaster DB Babelfish for Aurora PostgreSQL
- Memigrasi basis data SQL Server ke Babelfish for Aurora PostgreSQL

- Autentikasi basis data dengan Babelfish for Aurora PostgreSQL
- Menghubungkan ke klaster DB Babelfish
- Menggunakan Babelfish
- Memecahkan Masalah Babelfish
- Menonaktifkan Babelfish
- Pembaruan versi Babelfish
- Referensi Babelfish for Aurora PostgreSQL

Batasan Babelfish

Batasan berikut saat ini berlaku untuk Babelfish for Aurora PostgreSQL:

- Babelfish saat ini tidak mendukung fitur Aurora berikut ini:
  - Deployment Blue/Green Amazon RDS
  - AWS Identity and Access Management
  - Stream Aktivitas Basis Data (DAS)
  - Replikasi logis PostgreSQL
  - RDS Data API dengan Aurora PostgreSQL Serverless v2 dan disediakan
  - Proksi RDS dengan RDS for SQL Server
  - Salted Challenge Response Authentication Mechanism (SCRAM)
  - Editor kueri
- Babelfish saat ini tidak mendukung otentikasi berbasis Kerberos untuk grup Active Directory.
- Babelfish tidak menyediakan dukungan API driver klien berikut ini:
  - Permintaan API dengan atribut koneksi yang terkait dengan Microsoft Distributed Transaction Coordinator (MSDTC) tidak didukung. Ini termasuk panggilan XA oleh kelas SQLServerXAResource di Driver JDBC SQL Server.
  - Babelfish mendukung penyatuan koneksi dengan driver yang menggunakan versi terbaru dari protokol TDS. Dengan driver yang lebih lama, permintaan API dengan atribut koneksi dan metode yang terkait dengan penyatuan koneksi tidak didukung.
- Babelfish saat ini tidak mendukung ekstensi Aurora PostgreSQL berikut:
  - `bloom`
  - `btree_gin`
  - `btree_gist`
  - `citext`
  - `cube`
  - `hstore`
  - `hypopg`
  - Replikasi logis menggunakan `pglogical`
  - `ltree`
  - `pgcrypto`

- Manajemen rencana kueri menggunakan `apg_plan_mgmt`

  Untuk mempelajari lebih lanjut tentang ekstensi PostgreSQL, lihat. Menangani ekstensi dan pembungkus data asing

- Driver jTDS sumber terbuka yang dirancang sebagai alternatif driver JDBC Microsoft tidak didukung.

Memahami konfigurasi dan arsitektur Babelfish

Anda mengelola klaster DB Edisi yang Kompatibel dengan Aurora PostgreSQL yang menjalankan Babelfish seperti halnya klaster DB Aurora. Artinya, Anda mendapat manfaat dari skalabilitas, ketersediaan tinggi dengan dukungan failover, dan replikasi default yang disediakan oleh klaster DB Aurora. Untuk mempelajari lebih lanjut tentang kapabilitas ini, lihat Mengelola performa dan penskalaan untuk klaster DB Aurora, Ketersediaan yang tinggi untuk Amazon Aurora, dan Replikasi dengan Amazon Aurora. Anda juga memiliki akses ke banyak AWS alat dan utilitas lain, termasuk yang berikut ini:

- Amazon CloudWatch adalah layanan pemantauan dan observabilitas yang memberi Anda data dan wawasan yang dapat ditindaklanjuti. Untuk informasi selengkapnya, lihat Memantau metrik Amazon Aurora dengan Amazon CloudWatch.
- Wawasan Performa adalah fitur penyetelan dan pemantauan performa basis data yang membantu Anda menilai beban basis data dengan cepat. Untuk mempelajari selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.
- Basis data global Aurora mencakup beberapa Wilayah AWS, memungkinkan pembacaan global latensi rendah dan memberikan pemulihan cepat dari pemadaman langka yang mungkin memengaruhi keseluruhan. Wilayah AWS Untuk informasi selengkapnya, lihat Menggunakan basis data global Amazon Aurora.
- Penambalan perangkat lunak otomatis menjaga database Anda up-to-date dengan keamanan terbaru dan tambalan fitur saat tersedia.
- Peristiwa Amazon RDS memberi tahu Anda melalui email atau pesan SMS tentang peristiwa basis data penting, seperti failover otomatis. Untuk informasi selengkapnya, lihat Memantau peristiwa Amazon Aurora.

Berikut ini, Anda dapat mempelajari tentang arsitektur Babelfish dan bagaimana basis data SQL Server yang Anda migrasi ditangani oleh Babelfish. Saat membuat klaster DB Babelfish, Anda perlu

membuat beberapa keputusan lebih awal tentang satu basis data atau beberapa basis data, kolasi, dan detail lainnya.

Topik

- Arsitektur Babelfish
- Pengaturan grup parameter klaster DB untuk Babelfish
- Kolasi yang didukung oleh Babelfish
- Mengelola penanganan kesalahan Babelfish dengan escape hatch

Arsitektur Babelfish

Saat Anda membuat klaster Aurora PostgreSQL dengan Babelfish diaktifkan, Aurora menyediakan klaster dengan basis data PostgreSQL bernama `babelfish_db`. Basis data ini adalah tempat semua objek dan struktur SQL Server yang dimigrasi berada.

> **Note**
>
> Dalam klaster Aurora PostgreSQL, nama basis data `babelfish_db` dicadangkan untuk Babelfish. Membuat basis data "babelfish_db" Anda sendiri pada klaster Babelfish DB mencegah Aurora berhasil menyediakan Babelfish.

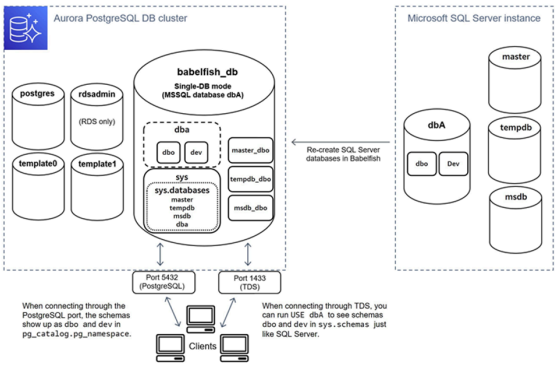
Saat Anda terhubung ke port TDS, sesi ditempatkan pada basis data `babelfish_db`. Dari T-SQL, strukturnya terlihat mirip dengan kondisi saat terhubung ke instans SQL Server. Anda dapat melihat basis data `master`,`msdb`, dan `tempdb`, serta katalog `sys.databases`. Anda dapat membuat basis data pengguna tambahan dan beralih antar-basis data dengan pernyataan USE. Ketika Anda membuat basis data pengguna SQL Server, basis data tersebut akan diratakan ke dalam basis data PostgreSQL `babelfish_db`. Basis data Anda mempertahankan sintaks lintas-basis data dan semantik yang setara atau mirip dengan yang disediakan oleh SQL Server.

Menggunakan Babelfish dengan satu atau beberapa basis data

Saat Anda membuat klaster Aurora PostgreSQL untuk digunakan dengan Babelfish, Anda memilih antara menggunakan satu basis data SQL Server atau beberapa basis data SQL Server sekaligus. Pilihan Anda memengaruhi bagaimana nama skema SQL Server di dalam basis data `babelfish_db` muncul dari Aurora PostgreSQL. Mode migrasi disimpan dalam parameter

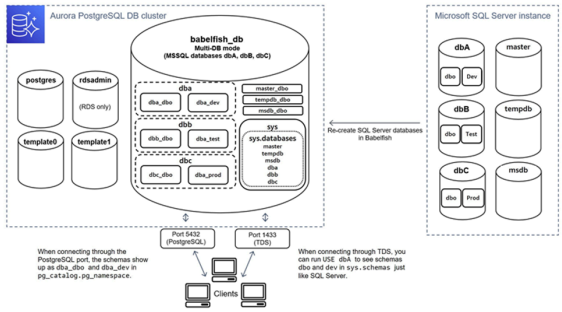
`migration_mode`. Anda tidak boleh mengubah parameter ini setelah membuat klaster karena Anda bisa kehilangan akses ke semua objek SQL yang telah dibuat sebelumnya.

Dalam mode basis data tunggal, nama skema basis data SQL Server tetap sama dalam basis data `babelfish_db` PostgreSQL. Jika Anda memilih untuk hanya memigrasikan basis data tunggal, nama skema basis data pengguna yang dimigrasi dapat dirujuk di PostgreSQL dengan nama yang sama seperti di SQL Server. Misalnya, skema dbo dan `smith` berada dalam basis data dbA .



Saat menghubungkan melalui TDS, Anda dapat menjalankan `USE dbA` untuk melihat skema dbo dan `smith` dari T-SQL, seperti yang Anda lakukan di SQL Server. Nama skema yang tidak berubah juga terlihat dari PostgreSQL.

Dalam modus multi-basis data, nama skema basis data pengguna menjadi `dbname_schemaname` ketika diakses dari PostgreSQL. Nama skema tetap sama ketika diakses dari T-SQL.



Seperti yang ditunjukkan pada gambar, mode multi-basis data dan mode basis data tunggal sama dengan SQL Server saat menghubungkan melalui port TDS dan menggunakan T-SQL. Misalnya, `USE dbA` mencantumkan skema dbo dan `smith` seperti halnya di SQL Server. Nama skema yang dipetakan, seperti `dbA_dbo` dan `dbA_smith`, terlihat dari PostgreSQL.

Setiap basis data masih berisi skema Anda. Nama setiap basis data ditambahkan ke nama skema SQL Server, menggunakan garis bawah sebagai pembatas, misalnya:

- dbA berisi `dbA_dbo` dan `dbA_smith`.

- dbB berisi dbB_dbo dan dbB_jones.
- dbC berisi dbC_dbo dan dbC_miller.

Di dalam basis data babelfish_db, pengguna T-SQL masih perlu menjalankan USE dbname untuk mengubah konteks basis data, sehingga tampilan dan nuansanya tetap mirip dengan SQL Server.

Memilih mode migrasi

Setiap mode migrasi memiliki kelebihan dan kekurangan. Pilih mode migrasi berdasarkan jumlah basis data pengguna milik Anda, dan rencana migrasi Anda. Setelah Anda membuat klaster untuk digunakan dengan Babelfish, Anda tidak boleh mengubah mode migrasi karena Anda mungkin akan kehilangan akses ke semua objek SQL yang dibuat sebelumnya. Saat memilih mode migrasi, pertimbangkan persyaratan basis data dan klien pengguna Anda.

Saat Anda membuat klaster untuk digunakan dengan Babelfish, Aurora PostgreSQL membuat basis data sistem, master dan tempdb. Jika Anda membuat atau memodifikasi objek dalam basis data sistem (master atau tempdb), pastikan untuk membuat ulang objek tersebut di klaster baru. Tidak seperti SQL Server, Babelfish tidak menginisialisasi ulang tempdb setelah klaster melakukan reboot.

Gunakan mode migrasi basis data tunggal dalam kasus berikut:

- Jika Anda memigrasi satu basis data SQL Server. Dalam mode basis data tunggal, nama skema yang dimigrasi saat diakses dari PostgreSQL identik dengan nama skema SQL Server asli. Hal ini akan mengurangi perubahan kode pada kueri SQL yang ada jika Anda ingin mengoptimalkannya untuk dijalankan dengan koneksi PostgreSQL.
- Jika tujuan akhir Anda adalah migrasi lengkap ke Aurora PostgreSQL asli. Sebelum bermigrasi, konsolidasikan skema Anda ke dalam satu skema (dbo) dan kemudian migrasikan ke dalam satu klaster untuk mengurangi perubahan yang diperlukan.

Gunakan beberapa mode migrasi basis data dalam kasus berikut:

- Jika Anda menginginkan pengalaman SQL Server default dengan beberapa basis data pengguna dalam instans yang sama.
- Jika beberapa basis data pengguna perlu dimigrasikan bersama.

Pengaturan grup parameter klaster DB untuk Babelfish

Saat Anda membuat klaster DB Aurora PostgreSQL dan memilih Aktifkan Babelfish, grup parameter klaster DB akan dibuat untuk Anda secara otomatis jika Anda memilih Buat baru. Grup parameter klaster DB ini didasarkan pada grup parameter klaster Aurora PostgreSQL DB untuk versi Aurora PostgreSQL yang dipilih untuk instalasi, misalnya, Aurora PostgreSQL versi 14. Grup ini dinamai menggunakan pola umum berikut:

```
custom-aurora-postgresql14-babelfish-compat-3
```

Anda dapat mengubah pengaturan berikut selama proses pembuatan klaster tetapi beberapa di antaranya tidak dapat diubah setelah disimpan di grup parameter kustom, jadi pilih dengan cermat:

- Basis data tunggal atau Multibasis data
- Lokal kolasi default
- Nama kolasi
- Grup parameter DB

Untuk menggunakan klaster Aurora PostgreSQL DB versi 13 atau grup parameter yang lebih tinggi, edit grup dan atur parameter `babelfish_status` ke on. Tentukan opsi Babelfish sebelum membuat klaster Aurora PostgreSQL Anda. Untuk mempelajari selengkapnya, lihat Bekerja dengan grup parameter.

Parameter berikut mengontrol preferensi Babelfish. Kecuali dinyatakan lain dalam Deskripsi, parameter dapat dimodifikasi. Nilai default disertakan dalam deskripsi. Untuk melihat nilai yang diizinkan untuk parameter apa pun, lakukan hal berikut:

> **Note**
>
> Ketika Anda mengaitkan grup parameter DB baru dengan instans DB, parameter statis dan dinamis yang dimodifikasi diterapkan hanya setelah instans DB di-reboot. Namun, jika Anda memodifikasi parameter dinamis dalam grup parameter DB setelah Anda mengaitkannya dengan instans DB, perubahan ini diterapkan segera tanpa reboot.

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

2. Pilih Grup parameter dari menu navigasi.
3. Pilih grup parameter klaster DB `default.aurora-postgresql14` dari daftar.
4. Masukkan nama parameter di kolom pencarian. Misalnya, masukkan `babelfishpg_tsql.default_locale` di kolom pencarian untuk menampilkan parameter ini dan nilai defaultnya serta pengaturan yang diizinkan.

Parameter	Deskripsi	Tipe Penerapan	Dapat Dimodifikasi
babelfishpg_tds.tds_default_numeric_scale	Menetapkan skala default tipe numerik yang akan dikirim dalam metadata kolom TDS jika mesin tidak menentukannya. (Default: 8) (Diizinkan: 0–38)	dinamis	true
babelfishpg_tds.tds_default_numeric_precision	Integer yang menetapkan presisi default tipe numerik yang akan dikirim dalam metadata kolom TDS jika mesin tidak menentukannya. (Default: 38) (Diizinkan: 1–38)	dinamis	true
babelfishpg_tds.tds_default_packet_size	Integer yang menetapkan ukuran paket default untuk menghubungkan klien SQL Server. (Default: 4096) (Diizinkan: 512–32767)	dinamis	true

Parameter	Deskripsi	Tipe Penerapan	Dapat Dimodifikasi
babelfishpg_tds.tds_default_protocol_version	Integer yang menetapkan versi protokol TDS default untuk menghubungkan klien. (Default: DEFAULT) (Diizinkan: TDSv7.0, TDSv7.1, TDSv7.1.1, TDSv7.2, TDSv7.3a, TDSv7.3b, TDSv7.4, DEFAULT)	dinamis	true
babelfishpg_tds.default_server_name	String yang mengidentifikasi nama default server Babelfish. (Default: Microsoft SQL Server) (Diizinkan: null)	dinamis	true
babelfishpg_tds.enable_tds_debug_log_level	Integer yang menetapkan tingkat logging di TDS; 0 mematikan logging. (Default: 1) (Diizinkan: 0, 1, 2, 3)	dinamis	true

Parameter	Deskripsi	Tipe Penerapan	Dapat Dimodifikasi
babelfishpg_tds.listen_addresses	String yang menetapkan nama host atau alamat IP atau alamat untuk mendengarkan TDS aktif. Parameter ini tidak dapat dimodifikasi setelah klaster DB Babelfish dibuat. (Default: *) (Diizinkan: null)	–	false
babelfishpg_tds.port	Integer yang menentukan port TCP yang digunakan untuk permintaan dalam sintaks SQL Server. (Default: 1433) (Diizinkan: 1–65535)	statis	true
babelfishpg_tds.tds_ssl_encrypt	Boolean yang mengaktifkan (0) atau menonaktifkan (1) enkripsi untuk data yang melintasi port pendengar TDS. Untuk informasi detail cara menggunakan SSL untuk koneksi klien, lihat Pengaturan SSL Babelfish dan koneksi klien. (Default: 0) (Diizinkan: 0, 1)	dinamis	true

Parameter	Deskripsi	Tipe Penerapan	Dapat Dimodifikasi
babelfishpg_tds.tds_ssl_max_protocol_version	String yang menentukan versi protokol SSL/TLS tertinggi yang digunakan untuk sesi TDS. (Default: 'TLSv1.2') (Diizinkan: 'TLSv1', 'TLSv1.1', 'TLSv1.2')	dinamis	true
babelfishpg_tds.tds_ssl_min_protocol_version	String yang menentukan SSL/TLS minimum yang akan digunakan untuk sesi TDS. (Default: 'TLSv1') (Diizinkan: 'TLSv1', 'TLSv1.1', 'TLSv1.2')	dinamis	true
babelfishpg_tds.unix_socket_directories	String yang mengidentifikasi direktori soket Unix TDS. Parameter ini tidak dapat dimodifikasi setelah klaster DB Babelfish dibuat. (Default: /tmp) (Diizinkan: null)	–	false

Parameter	Deskripsi	Tipe Penerapan	Dapat Dimodifikasi
babelfishpg_tds.unix_socket_group	String yang mengidentifikasi grup soket Unix server TDS. Parameter ini tidak dapat dimodifikasi setelah klaster DB Babelfish dibuat. (Default: rdsdb) (Diizinkan: null)	–	false
babelfishpg_tsql.default_locale	Sebuah string yang menentukan lokal default yang digunakan untuk koleksi Babelfish. Lokal default hanya lokal dan tidak menyertakan kualifikasi apa pun. Tetapkan parameter ini saat Anda menyediakan klaster DB Babelfish. Setelah klaster DB disediakan, perubahan pada parameter ini diabaikan. (Default: en_US) (Diizinkan: Lihat tabel)	statis	true

Parameter	Deskripsi	Tipe Penerapan	Dapat Dimodifikasi
babelfishpg_tsql.migration_mode	Daftar yang tidak dapat dimodifikasi yang menentukan dukungan untuk basis data satu atau beberapa pengguna. Tetapkan parameter ini saat Anda menyediakan klaster DB Babelfish. Setelah klaster DB disediakan, Anda tidak dapat mengubah nilai parameter ini. (Default: multi-db dari Aurora PostgreSQL versi 16, single-db untuk versi yang lebih lama dari Aurora PostgreSQL versi 16) (Diizinkan: single-db, multi-db, null)	statis	true

Parameter	Deskripsi	Tipe Penerapan	Dapat Dimodifikasi
babelfishpg_tsql.server_collation_name	String yang menentukan nama pemeriksaan yang digunakan untuk tindakan tingkat server. Tetapkan parameter ini saat Anda menyediakan klaster DB Babelfish. Setelah klaster DB disediakan, jangan ubah nilai parameter ini. (Default: bbf_unicode_general_ci_as) (Diizinkan: Lihat tabel)	statis	true
babelfishpg_tsql.version	Sebuah string yang menetapkan output dari variabel @@VERSION. Jangan ubah nilai ini untuk klaster DB Aurora PostgreSQL. (Default: null) (Diizinkan: default)	dinamis	true

Parameter	Deskripsi	Tipe Penerapan	Dapat Dimodifikasi
rds.babelfish_status	String yang menetapkan status fungsionalitas Babelfish. Ketika parameter ini diatur ke `datatypesonly` , Babelfish dinonaktifkan tetapi tipe data SQL Server masih tersedia. (Default: nonaktif) (Diizinkan: aktif, nonaktif, khusus tipe data)	statis	true
unix_socket_permissions	Integer yang menetapkan izin soket Unix server TDS. Parameter ini tidak dapat dimodifikasi setelah klaster DB Babelfish dibuat. (Default: 0700) (Diizinkan: 0–511)	–	false

Pengaturan SSL Babelfish dan koneksi klien

Ketika klien terhubung ke port TDS (default `1433`), Babelfish membandingkan pengaturan Secure Sockets Layer (SSL) yang dikirim selama jabat tangan klien ke pengaturan parameter Babelfish SSL (`tds_ssl_encrypt`). Babelfish kemudian menentukan apakah koneksi diizinkan. Jika koneksi diizinkan, perilaku enkripsi diberlakukan atau tidak, tergantung pada pengaturan parameter Anda dan dukungan untuk enkripsi yang ditawarkan oleh klien.

Tabel berikut menunjukkan bagaimana Babelfish berperilaku untuk setiap kombinasi.

Pengaturan SSL Klien	Pengaturan SSL Babelfish	Koneksi diizinkan?	Nilai dikembalikan ke klien
ENCRYPT_OFF	tds_ssl_encrypt=0	Diizinkan, paket login dienkripsi	ENCRYPT_OFF
ENCRYPT_OFF	tds_ssl_encrypt=1	Diizinkan, seluruh koneksi dienkripsi	ENCRYPT_REQ
ENCRYPT_ON	tds_ssl_encrypt=0	Diizinkan, seluruh koneksi dienkripsi	ENCRYPT_ON
ENCRYPT_ON	tds_ssl_encrypt=1	Diizinkan, seluruh koneksi dienkripsi	ENCRYPT_ON
ENCRYPT_NOT_SUP	tds_ssl_encrypt=0	Ya	ENCRYPT_NOT_SUP
ENCRYPT_NOT_SUP	tds_ssl_encrypt=1	Tidak, koneksi ditutup	ENCRYPT_REQ

Pengaturan SSL Klien	Pengaturan SSL Babelfish	Koneksi diizinkan?	Nilai dikembalikan ke klien
ENCRYPT_REQ	tds_ssl_encrypt=0	Diizinkan, seluruh koneksi dienkripsi	ENCRYPT_ON
ENCRYPT_REQ	tds_ssl_encrypt=1	Diizinkan, seluruh koneksi dienkripsi	ENCRYPT_ON
ENCRYPT_CLIENT_CERT	tds_ssl_encrypt=0	Tidak, koneksi ditutup	Tidak didukung
ENCRYPT_CLIENT_CERT	tds_ssl_encrypt=1	Tidak, koneksi ditutup	Tidak didukung

Kolasi yang didukung oleh Babelfish

Saat Anda membuat klaster DB Aurora PostgreSQL dengan Babelfish, Anda memilih kolasi untuk data Anda. Sebuah kolasi menentukan urutan dan pola bit yang menghasilkan teks atau karakter dalam bahasa manusia tertulis tertentu. Kolasi mencakup aturan yang membandingkan data untuk set pola bit tertentu. Kolasi terkait dengan lokalisasi. Lokal yang berbeda memengaruhi pemetaan karakter, urutan, dan sejenisnya. Atribut kolasi tercermin dalam nama berbagai kolasi. Untuk mengetahui informasi atribut, lihat Babelfish collation attributes table.

Babelfish memetakan kolasi SQL Server ke kolasi sebanding yang disediakan oleh Babelfish. Babelfish menentukan kolasi Unicode dengan urutan dan perbandingan string yang sensitif secara budaya. Babelfish juga menyediakan cara untuk menerjemahkan kolasi di SQL Server DB Anda ke kolasi Babelfish yang paling cocok. Kolasi khusus lokal disediakan untuk berbagai bahasa dan wilayah.

Beberapa kolasi menentukan halaman kode yang sesuai dengan pengodean sisi klien. Babelfish secara otomatis menerjemahkan dari pengodean server ke pengodean klien bergantung pada kolasi setiap kolom output.

Babelfish mendukung kolasi yang tercantum dalam Babelfish supported collations table. Babelfish memetakan kolasi SQL Server ke kolasi sebanding yang disediakan oleh Babelfish.

Babelfish menggunakan pustaka kolasi International Components for Unicode (ICU) versi 153.80. Untuk informasi lebih lanjut tentang kolasi ICU, lihat Kolasi dalam dokumentasi ICU. Untuk mempelajari lebih lanjut tentang PostgreSQL dan kolasi, lihat Dukungan Kolasi dalam dokumentasi PostgreSQL.

Topik

- Parameter klaster DB yang mengontrol kolasi dan lokal
- Kolasi deterministik dan nondeterministik serta Babelfish
- Kolasi yang didukung oleh Babelfish
- Kolasi Default di Babelfish
- Mengelola kolasi
- Batasan kolasi dan perbedaan perilaku

Parameter klaster DB yang mengontrol kolasi dan lokal

Parameter berikut memengaruhi perilaku kolasi.

babelfishpg_tsql.default_locale

Parameter ini menentukan lokal default yang digunakan oleh kolasi. Parameter ini digunakan dalam kombinasi dengan atribut yang tercantum dalam Babelfish collation attributes table untuk menyesuaikan kolasi untuk bahasa dan wilayah tertentu. Nilai default untuk parameter ini adalah `en-US`.

Lokal default berlaku untuk semua nama kolasi Babelfish yang dimulai dengan "BBF" dan untuk semua kolasi SQL Server yang dipetakan ke kolasi Babelfish. Mengubah pengaturan untuk parameter ini pada klaster DB Babelfish yang ada tidak memengaruhi lokal kolasi yang ada. Untuk daftar kolasi, lihat Babelfish supported collations table.

babelfishpg_tsql.server_collation_name

Parameter ini menentukan kolasi default untuk server (instans klaster DB Aurora PostgreSQL) dan basis data. Nilai default-nya adalah `sql_latin1_general_cp1_ci_as`. `server_collation_name` harus berupa kolasi `CI_AS` karena di T-SQL, kolasi server akan menentukan perbandingan pengidentifikasi.

Saat Anda membuat klaster DB Babelfish, Anda memilih Nama kolasi dari daftar yang dapat dipilih. Ini termasuk susunan yang tercantum dalam Babelfish supported collations table. Jangan memodifikasi `server_collation_name` setelah basis data Babelfish dibuat.

Pengaturan yang Anda pilih saat membuat klaster DB Babelfish for Aurora PostgreSQL disimpan dalam grup parameter klaster DB yang terkait dengan klaster untuk parameter ini dan mengatur perilaku kolasinya.

Kolasi deterministik dan nondeterministik serta Babelfish

Babelfish mendukung kolasi deterministik dan nondeterministik:

- Kolasi deterministik mengevaluasi karakter yang memiliki urutan byte identik sebagai sama. Itu berarti itu `x` dan `X` tidak sama dalam kolasi deterministik. Kolasi deterministik dapat berupa peka huruf besar/kecil (CS) dan peka karakter beraksen (AS).
- Kolasi nondeterministik tidak membutuhkan kecocokan yang identik. Kolasi nondeterministik mengevaluasi `x` dan `X` sebagai nilai yang setara. Kolasi nondeterministik bersifat tidak peka huruf besar/kecil (CI) dan tidak peka karakter beraksen (AI).

Pada tabel berikut, Anda dapat menemukan beberapa perbedaan perilaku antara Babelfish dan PostgreSQL saat menggunakan kolasi nondeterministik.

Babelfish	PostgreSQL
Mendukung klausa LIKE untuk kolasi CI_AS.	Tidak mendukung klausa LIKE pada kolasi nondeterministik.
Tidak mendukung klausa `LIKE` pada kolasi AI.	
Jangan mendukung operasi pencocokan pola pada kolasi nondeterministik.	

Untuk melihat daftar batasan dan perbedaan perilaku lainnya untuk Babelfish dibandingkan dengan SQL Server dan PostgreSQL, lihat Batasan kolasi dan perbedaan perilaku.

Babelfish dan SQL Server mengikuti konvensi penamaan untuk kolasi yang menjelaskan atribut kolasi, seperti yang ditunjukkan pada tabel berikut.

Atribut	Deskripsi
AI	Tidak peka karakter beraksen.
AS	Peka karakter beraksen.
BIN2	BIN2 meminta data untuk diurutkan dalam urutan titik kode. Urutan titik kode Unicode adalah urutan karakter yang sama untuk pengodean UTF-8, UTF-16, dan UCS-2. Urutan titik kode adalah kolasi deterministik cepat.
CI	Tidak peka huruf besar/kecil.
CS	Peka huruf besar/kecil.
PREF	Untuk mengurutkan huruf besar sebelum huruf kecil, gunakan kolasi PREF. Jika perbandingan tidak peka huruf besar/kecil, versi huruf besar akan diurutkan sebelum versi huruf kecil, jika tidak ada perbedaan lain. Pustaka ICU mendukung preferensi huruf besar dengan `colCaseFirst=upper` , tetapi tidak untuk kolasi CI_AS. PREF hanya dapat diterapkan pada kolasi deterministik `CS_AS`.

Kolasi yang didukung oleh Babelfish

Gunakan kolasi berikut sebagai kolasi server atau kolasi objek.

ID Kolasi	Catatan
bbf_unicode_general_ci_as	Mendukung perbandingan tidak peka huruf besar/kecil dan operator LIKE.
bbf_unicode_cp1_ci_as	Kolasi nondeterministik juga dikenal sebagai CP1252.
bbf_unicode_CP1250_ci_as	Kolasi nondeterministik digunakan untuk mewakili teks dalam bahasa Eropa Tengah dan Eropa Timur yang menggunakan aksara Latin.
bbf_unicode_CP1251_ci_as	Kolasi nondeterministik untuk bahasa yang menggunakan skrip Sirilik.
bbf_unicode_cp1253_ci_as	Kolasi nondeterministik digunakan untuk mewakili bahasa Yunani modern.
bbf_unicode_cp1254_ci_as	Kolasi nondeterministik yang mendukung bahasa Turki.
bbf_unicode_cp1255_ci_as	Kolasi nondeterministik yang mendukung bahasa Ibrani.
bbf_unicode_cp1256_ci_as	Kolasi nondeterministik digunakan untuk menulis bahasa yang menggunakan aksara Arab.
bbf_unicode_cp1257_ci_as	Kolasi nondeterministik digunakan untuk mendukung bahasa Estonia, Latvia, dan Lituania.
bbf_unicode_cp1258_ci_as	Kolasi nondeterministik digunakan untuk menulis karakter Vietnam.

ID Kolasi	Catatan
bbf_unicode_cp874_ci_as	Kolasi nondeterministik digunakan untuk menulis karakter Thailand.
sql_latin1_general_cp1250_ci_as	Pengodean karakter byte tunggal nondeterministik digunakan untuk mewakili karakter Latin.
sql_latin1_general_cp1251_ci_as	Kolasi nondeterministik yang mendukung karakter Latin.
sql_latin1_general_cp1_ci_as	Kolasi nondeterministik yang mendukung karakter Latin.
sql_latin1_general_cp1253_ci_as	Kolasi nondeterministik yang mendukung karakter Latin.
sql_latin1_general_cp1254_ci_as	Kolasi nondeterministik yang mendukung karakter Latin.
sql_latin1_general_cp1255_ci_as	Kolasi nondeterministik yang mendukung karakter Latin.
sql_latin1_general_cp1256_ci_as	Kolasi nondeterministik yang mendukung karakter Latin.
sql_latin1_general_cp1257_ci_as	Kolasi nondeterministik yang mendukung karakter Latin.
sql_latin1_general_cp1258_ci_as	Kolasi nondeterministik yang mendukung karakter Latin.

ID Kolasi	Catatan
chinese_prc_ci_as	Kolasi nondeterministik yang mendukung bahasa Mandarin (RRC).
cyrillic_general_ci_as	Kolasi nondeterministik yang mendukung Sirilik.
finnish_swedish_ci_as	Kolasi nondeterministik yang mendukung bahasa Finlandia.
french_ci_as	Kolasi nondeterministik yang mendukung bahasa Prancis.
japanese_ci_as	Kolasi nondeterministik yang mendukung bahasa Jepang. Didukung di Babelfish 2.1.0 dan rilis yang lebih baru.
korean_wansung_ci_as	Kolasi nondeterministik yang mendukung bahasa Korea (dengan jenis kamus).
latin1_general_ci_as	Kolasi nondeterministik yang mendukung karakter Latin.
modern_spanish_ci_as	Kolasi nondeterministik yang mendukung bahasa Spanyol Modern.
polish_ci_as	Kolasi nondeterministik yang mendukung bahasa Polandia.
thai_ci_as	Kolasi nondeterministik yang mendukung bahasa Thailand.
tradisional_spanish_ci_as	Kolasi nondeterministik yang mendukung bahasa Spanyol (jenis tradisional).
turkish_ci_as	Kolasi nondeterministik yang mendukung bahasa Turki.
ukrainian_ci_as	Kolasi nondeterministik yang mendukung Ukraina.

ID Kolasi	Catatan
vietnamese_ci_as	Kolasi nondeterministik yang mendukung bahasa Vietnam.

Anda dapat menggunakan kolasi berikut sebagai kolasi objek.

Dialek	Opsi deterministik	Opsi nondeterministik
Arab	Arabic_CS_AS	Arabic_CI_AS, Arabic_CI_AI
Mandarin	Chinese_CS_AS	Chinese_CI_AS, Chinese_CI_AI
Cyrillic_Umum	Cyrillic_General_CS_AS	Cyrillic_General_CI_AS, Cyrillic_General_CI_AI
Estonia	Estonian_CS_AS	Estonian_CI_AS, Estonian_CI_AI
Finlandia _Swedia	Finnish_Swedish_CS_AS	Finnish_Swedish_CI_AS, Finnish_Swedish_CI_AI
Prancis	French_CS_AS	French_CI_AS, French_CI_AI
Yunani	Greek_CS_AS	Greek_CI_AS, Greek_CI_AI
Ibrani	Hebrew_CS_AS	Hebrew_CI_AS, Hebrew_CI_AI
Jepang (Babelfish 2.1.0 dan lebih tinggi)	Japanese_CS_AS	Japanese_CI_AI, Japanese_CI_AS
Korea_Wamsung	Korean_Wamsung_CS_AS	

Dialek	Opsi deterministik	Opsi nondeterministik
		Korean_Wamsung_CI_AS, Korean_Wamsung_CI_AI
Modern_Spanyol	Modern_Spanish_CS_AS	Modern_Spanish_CI_AS, Modern_Spanish_CI_AI
Mongolia	Mongolian_CS_AS	Mongolian_CI_AS, Mongolian_CI_AI
Polandia	Polish_CS_AS	Polish_CI_AS, Polish_CI_AI
Thai	Thai_CS_AS	Thai_CI_AS, Thai_CI_AI
Tradisional_Spanyol	Traditional_Spanish_CS_AS	Traditional_Spanish_CI_AS, Traditional_Spanish_CI_AI
Turki	Turkish_CS_AS	Turkish_CI_AS, Turkish_CI_AI
Ukrania	Ukranian_CS_AS	Ukranian_CI_AS, Ukranian_CI_AI
Vietnam	Vietnamese_CS_AS	Vietnamese_CI_AS, Vietnamese_CI_AI

Kolasi Default di Babelfish

Sebelumnya, kolasi default dari tipe data yang dapat dikolasi adalah `pg_catalog.default`. Tipe data dan objek yang bergantung pada tipe data ini mengikuti kolasi peka huruf besar/kecil. Kondisi ini berpotensi memengaruhi objek T-SQL dari kumpulan data dengan kolasi tidak peka huruf besar/kecil. Dimulai dengan Babelfish 2.3.0, kolasi default untuk tipe data yang dapat dikolasi (kecuali TEXT dan NTEXT) sama dengan kolasi dalam parameter `babelfishpg_tsql.server_collation_name`. Saat Anda meningkatkan ke Babelfish 2.3.0, kolasi default dipilih secara otomatis pada saat pembuatan klaster DB, yang tidak menciptakan dampak yang terlihat.

Mengelola kolasi

Pustaka ICU menyediakan pelacakan versi pemeriksaan untuk memastikan bahwa indeks yang bergantung pada kolasi dapat diindeks ulang ketika versi baru ICU tersedia. Untuk melihat apakah basis data Anda saat ini memiliki kolasi yang perlu disegarkan, Anda dapat menggunakan kueri berikut setelah menghubungkan menggunakan `psql` atau `pgAdmin`:

```
SELECT pg_describe_object(refclassid, refobjid,
    refobjsubid) AS "Collation",
    pg_describe_object(classid, objid, objsubid) AS "Object"
    FROM pg_depend d JOIN pg_collation c ON refclassid = 'pg_collation'::regclass
    AND refobjid = c.oid WHERE c.collversion <> pg_collation_actual_version(c.oid)
    ORDER BY 1, 2;
```

Kueri ini menghasilkan output seperti berikut ini:

```
 Collation | Object
-----------+--------
(0 rows)
```

Dalam contoh ini, tidak ada kolasi yang perlu diperbarui.

Untuk mendapatkan daftar kolasi yang telah ditentukan dalam basis data Babelfish Anda, Anda dapat menggunakan `psql` atau `pgAdmin` dengan kueri berikut:

```
SELECT * FROM pg_collation;
```

Kolasi yang telah ditentukan disimpan dalam tabel `sys.fn_helpcollations`. Anda dapat menggunakan perintah berikut untuk menampilkan informasi tentang kolasi (seperti lcid, style, dan bendera kolasi). Untuk mendapatkan daftar semua kolasi dengan menggunakan `sqlcmd`, hubungkan ke port T-SQL (1433, secara default) dan jalankan kueri berikut:

```
1> :setvar SQLCMDMAXVARTYPEWIDTH 40
2> :setvar SQLCMDMAXFIXEDTYPEWIDTH 40
3> SELECT * FROM fn_helpcollations()
4> GO
name                                  description
------------------------------------- ----------------------------------------
arabic_cs_as                          Arabic, case-sensitive, accent-sensitive
arabic_ci_ai                          Arabic, case-insensitive, accent-insensi
```

```
arabic_ci_as                          Arabic, case-insensitive, accent-sensiti
bbf_unicode_bin2                      Unicode-General, case-sensitive, accent-
bbf_unicode_cp1250_ci_ai              Default locale, code page 1250, case-ins
bbf_unicode_cp1250_ci_as              Default locale, code page 1250, case-ins
bbf_unicode_cp1250_cs_ai              Default locale, code page 1250, case-sen
bbf_unicode_cp1250_cs_as              Default locale, code page 1250, case-sen
bbf_unicode_pref_cp1250_cs_as         Default locale, code page 1250, case-sen
bbf_unicode_cp1251_ci_ai              Default locale, code page 1251, case-ins
bbf_unicode_cp1251_ci_as              Default locale, code page 1251, case-ins
bbf_unicode_cp1254_ci_ai              Default locale, code page 1254, case-ins
...
(124 rows affected)
```

Baris 1 dan 2 yang ditunjukkan pada contoh mempersempit output hanya untuk tujuan keterbacaan dokumentasi.

```
1> SELECT SERVERPROPERTY('COLLATION')
2> GO
serverproperty
------------------------------------------------------------------------
sql_latin1_general_cp1_ci_as

(1 rows affected)
1>
```

Batasan kolasi dan perbedaan perilaku

Babelfish menggunakan pustaka ICU untuk dukungan kolasi. PostgreSQL dibuat dengan versi ICU tertentu dan dapat mencocokkan paling banyak satu versi kolasi. Variasi antarversi tidak dapat dihindari, begitu juga variasi kecil sepanjang waktu seiring berkembangnya bahasa. Dalam daftar berikut, Anda dapat menemukan batasan dan variasi perilaku yang diketahui dari kolasi Babelfish:

- Ketergantungan tipe indeks dan pemeriksaan – Indeks pada tipe yang ditentukan pengguna yang bergantung pada pustaka kolasi International Components for Unicode (ICU) (pustaka yang digunakan oleh Babelfish) valid saat versi pustaka berubah.
- Fungsi COLLATIONPROPERTY – Properti kolasi diimplementasikan hanya untuk kolasi BBF Babelfish yang didukung. Untuk informasi selengkapnya, lihat Babelfish supported collations table.
- Perbedaan aturan pengurutan Unicode – Kolasi SQL untuk SQL Server mengurutkan data yang berenkode Unicode (`nchar` dan `nvarchar`) secara berbeda dibandingkan data yang tidak berenkode Unicode (`char` dan `varchar`). Basis data Babelfish selalu berenkode UTF-8 dan selalu

menerapkan aturan pengurutan Unicode secara konsisten, terlepas dari tipe data, jadi urutan untuk `char` atau `varchar` sama dengan urutan untuk `nchar` atau `nvarchar`.

- Perilaku pengurutan dan kolasi setara sekunder – Kolasi ICU Unicode setara sekunder default (`CI_AS`) mengurutkan tanda baca dan karakter nonalfanumerik lainnya sebelum karakter numerik, dan karakter numerik sebelum karakter abjad. Namun, urutan tanda baca dan karakter khusus lainnya berbeda.
- Kolasi tersier, solusi untuk ORDER BY – Kolasi SQL, seperti `SQL_Latin1_General_Pref_CP1_CI_AS`, mendukung fungsi `TERTIARY_WEIGHTS` dan kemampuan untuk mengurutkan string yang membandingkan secara merata dalam kolasi `CI_AS` yang akan diurutkan berdasarkan huruf besar terlebih dahulu: `ABC`, `ABc`, `AbC`, `Abc`, `aBC`, `aBc`, `abC`, dan terakhir `abc`. Dengan demikian, fungsi analitik `DENSE_RANK OVER (ORDER BY column)` menilai string-string ini sebagai memiliki peringkat yang sama tetapi mengurutkan string berdasarkan huruf besar terlebih dahulu dalam sebuah partisi.

  Anda bisa mendapatkan hasil yang serupa menggunakan Babelfish dengan menambahkan klausa `COLLATE` ke klausa `ORDER BY` yang menentukan kolasi `CS_AS` tersier yang menentukan `@colCaseFirst=upper`. Namun, pengubah `colCaseFirst` hanya berlaku untuk string setara tersier (bukan setara sekunder seperti kolasi `CI_AS`). Jadi, Anda tidak dapat mengemulasi kolasi SQL tersier menggunakan satu kolasi ICU.

  Sebagai solusinya, kami merekomendasikan agar Anda mengubah aplikasi yang menggunakan kolasi `SQL_Latin1_General_Pref_CP1_CI_AS` untuk menggunakan kolasi `BBF_SQL_Latin1_General_CP1_CI_AS` terlebih dahulu. Kemudian, tambahkan `COLLATE BBF_SQL_Latin1_General_Pref_CP1_CS_AS` ke klausa `ORDER BY` apa pun untuk kolom ini.
- Ekspansi karakter – Ekspansi karakter memperlakukan satu karakter sama dengan urutan karakter di tingkat primer. Kolasi `CI_AS` default SQL Server mendukung ekspansi karakter. Kolasi ICU mendukung ekspansi karakter hanya untuk kolasi yang tidak peka terhadap aksen.

  Ketika ekspansi karakter diperlukan, gunakan kolasi `AI` untuk perbandingan. Namun, kolasi tersebut saat ini tidak didukung oleh operator LIKE.
- enkode char dan varchar – Ketika kolasi SQL digunakan untuk tipe data `char` atau `varchar`, urutan karakter ASCII 127 sebelumnya ditentukan oleh halaman kode spesifik untuk kolasi SQL tersebut. Untuk kolasi SQL, string yang dideklarasikan sebagai `char` atau `varchar` mungkin diurutkan secara berbeda dibandingkan string yang dideklarasikan sebagai `nchar` atau `nvarchar`.

PostgreSQL mengenkode semua string dengan enkode basis data, sehingga semua karakter dikonversi ke UTF-8 dan diurutkan menggunakan aturan Unicode.

Karena kolasi SQL mengurutkan tipe data nchar dan nvarchar menggunakan aturan Unicode, Babelfish mengenkode semua string di server menggunakan UTF-8. Babelfish mengurutkan string nchar dan nvarchar dengan cara yang sama seperti mengurutkan string char dan varchar, yaitu menggunakan aturan Unicode.

- Karakter tambahan – Fungsi SQL Server `NCHAR`, `UNICODE`, dan `LEN` mendukung karakter untuk titik kode di luar Unicode Basic Multilingual Plane (BMP). Sebaliknya, kolasi non-SC menggunakan karakter pasangan pengganti untuk menangani karakter tambahan. Untuk tipe data Unicode, SQL Server dapat mewakili hingga 65.535 karakter menggunakan UCS-2, atau rentang Unicode penuh (1.114.114 karakter) jika karakter tambahan digunakan.
- Kolasi yang peka terhadap Kana (KS) – Kolasi yang peka terhadap Kana (KS) adalah salah satu kolasi yang memperlakukan karakter Kana Jepang `Hiragana` dan `Katakana` secara berbeda. ICU mendukung standar kolasi Jepang `JIS X 4061`. Pengubah lokal `colhiraganaQ [on | off]` yang kini tidak digunakan lagi mungkin menyediakan fungsi yang sama dengan kolasi KS. Namun, kolasi KS untuk nama yang sama dengan SQL Server saat ini tidak didukung oleh Babelfish.
- Kolasi yang peka terhadap lebar (WS) – Ketika karakter byte tunggal (lebar setengah) dan karakter sama yang direpresentasikan sebagai karakter byte ganda (lebar penuh) diperlakukan secara berbeda, kolasi ini disebut sebagai peka terhadap lebar (WS). Kolasi WS untuk nama yang sama dengan SQL Server saat ini tidak didukung oleh Babelfish.
- Kolasi yang peka terhadap pemilih variasi (VSS) – Kolasi yang peka terhadap pemilih variasi (VSS) membedakan antara pemilih variasi ideografis dalam kolasi Jepang `Japanese_Bushu_Kakusu_140` dan `Japanese_XJIS_140`. Urutan variasi terdiri dari karakter dasar ditambah pemilih variasi tambahan. Jika Anda tidak memilih opsi `_VSS`, pemilih variasi tidak dipertimbangkan dalam perbandingan.

  Kolasi VSS saat ini tidak didukung oleh Babelfish.
- Kolasi BIN dan BIN2 – Kolasi BIN2 mengurutkan karakter menurut urutan titik kode. Urutan biner byte-per-byte UTF-8 mempertahankan urutan titik kode Unicode, sehingga kolasi ini juga cenderung menjadi kolasi dengan performa terbaik. Jika urutan titik kode Unicode berfungsi untuk aplikasi, pertimbangkan untuk menggunakan kolasi BIN2. Namun, penggunaan kolasi BIN2 dapat mengakibatkan data ditampilkan kepada klien dalam urutan yang tidak terduga secara budaya. Pemetaan baru untuk karakter huruf kecil ditambahkan ke Unicode seiring berjalannya waktu,

sehingga fungsi `LOWER` mungkin memiliki performa yang berbeda pada versi ICU yang berbeda. Kasus ini bersifat khusus untuk masalah versioning kolasi yang lebih umum daripada sebagai satu kasus spesifik untuk kolasi BIN2.

Babelfish menyediakan kolasi `BBF_Latin1_General_BIN2` dengan distribusi Babelfish untuk membuat kolasi dalam urutan titik kode Unicode. Dalam kolasi BIN, hanya karakter pertama yang diurutkan sebagai wchar. Karakter yang tersisa diurutkan byte-per-byte, secara efektif dalam urutan titik kode sesuai dengan enkodenya. Pendekatan ini tidak mengikuti aturan kolasi Unicode dan tidak didukung oleh Babelfish.

- Kolasi nondeterministik dan batasan CHARINDEX – Untuk rilis Babelfish yang lebih lama dari versi 2.1.0, Anda tidak dapat menggunakan CHARINDEX dengan kolasi nondeterministik. Secara default, Babelfish menggunakan kolasi yang tidak peka terhadap huruf besar/kecil (nondeterministik). Menggunakan CHARINDEX untuk versi Babelfish yang lebih lama akan menimbulkan kesalahan runtime berikut:

```
nondeterministic collations are not supported for substring searches
```

> **Note**
>
> Batasan dan solusi ini hanya berlaku untuk Babelfish versi 1.x (versi Aurora PostgreSQL 13.x). Babelfish 2.1.0 dan rilis yang lebih tinggi tidak memiliki masalah ini.

Anda dapat mengatasi masalah ini dengan salah satu cara berikut:

- Ubah ekspresi secara eksplisit menjadi kolasi yang peka terhadap huruf besar/kecil dan seragamkan huruf besar/kecil pada kedua argumen dengan menerapkan LOWER atau UPPER. Misalnya, `SELECT charindex('x', a) FROM t1` akan menjadi berikut:

```
SELECT charindex(LOWER('x'), LOWER(a COLLATE sql_latin1_general_cp1_cs_as)) FROM t1
```

- Buat fungsi SQL f_charindex, lalu ganti panggilan CHARINDEX dengan panggilan ke fungsi berikut:

```
CREATE function f_charindex(@s1 varchar(max), @s2 varchar(max)) RETURNS int
AS
BEGIN
declare @i int = 1
WHILE len(@s2) >= len(@s1)
```

```
BEGIN
  if LOWER(@s1) = LOWER(substring(@s2,1,len(@s1))) return @i
  set @i += 1
  set @s2 = substring(@s2,2,999999999)
END
return 0
END
go
```

Mengelola penanganan kesalahan Babelfish dengan escape hatch

Babelfish meniru perilaku SQL untuk aliran kontrol dan status transaksi bila memungkinkan. Ketika Babelfish menemukan kesalahan, layanan ini menampilkan kode kesalahan yang mirip dengan kode kesalahan SQL Server. Jika Babelfish tidak dapat memetakan kesalahan menjadi kode SQL Server, layanan ini menampilkan kode kesalahan tetap (`33557097`) dan mengambil tindakan spesifik berdasarkan jenis kesalahan, sebagai berikut:

- Untuk kesalahan waktu kompilasi, Babelfish akan melakukan rollback pada transaksi.
- Untuk kesalahan runtime, Babelfish mengakhiri batch dan melakukan rollback pada transaksi.
- Untuk kesalahan protokol antara klien dan server, tidak akan ada rollback pada transaksi.

Jika kode kesalahan tidak dapat dipetakan menjadi kode yang setara dan kode untuk kesalahan serupa tersedia, kode kesalahan akan dipetakan menjadi kode alternatif. Misalnya, perilaku yang menyebabkan kode SQL Server `8143` dan `8144` dipetakan menjadi `8143`.

Kesalahan yang tidak dapat dipetakan tidak mematuhi konsep `TRY... CATCH`.

Anda dapat menggunakan `@@ERROR` untuk menampilkan kode kesalahan SQL Server, atau fungsi `@@PGERROR` untuk menampilkan kode kesalahan PostgreSQL. Anda juga dapat menggunakan fungsi `fn_mapped_system_error_list` untuk menampilkan daftar kode kesalahan yang dipetakan. Untuk informasi tentang kode kesalahan PostgreSQL, lihat situs web PostgreSQL.

Memodifikasi pengaturan escape hatch Babelfish

Untuk menangani pernyataan yang mungkin gagal, Babelfish mendefinisikan opsi tertentu yang disebut escape hatch. Escape hatch adalah opsi yang menentukan perilaku Babelfish ketika menemukan fitur atau sintaks yang tidak didukung.

Anda dapat menggunakan prosedur tersimpan `sp_babelfish_configure` untuk mengontrol pengaturan escape hatch. Gunakan skrip untuk mengatur escape hatch ke `ignore` atau `strict`. Jika disetel ke `strict`, Babelfish menampilkan kesalahan yang perlu Anda perbaiki sebelum melanjutkan.

Untuk menerapkan perubahan pada sesi saat ini dan pada tingkat klaster, sertakan kata kunci `server`.

Penggunaannya dapat dilihat sebagai berikut:

- Untuk mencantumkan semua escape hatch dan statusnya, serta informasi penggunaan, jalankan `sp_babelfish_configure`.
- Untuk mencantmukan escape hatch bernama beserta nilainya, untuk sesi saat ini atau tingkat klaster, jalankan perintah `sp_babelfish_configure 'hatch_name'` di mana *`hatch_name`* adalah pengidentifikasi satu escape hatch atau lebih. *`hatch_name`* dapat menggunakan wildcard SQL, seperti '%'.
- Untuk mengatur satu escape hatch atau lebih ke nilai yang ditentukan, jalankan `sp_babelfish_configure ['hatch_name' [, 'strict'|'ignore' [, 'server']]]`. Untuk membuat pengaturan menjadi permanen di tingkat klaster, sertakan kata kunci `server`, seperti yang ditunjukkan sebagai berikut:

  ```
  EXECUTE sp_babelfish_configure 'escape_hatch_unique_constraint', 'ignore', 'server'
  ```

  Untuk mengaturnya hanya untuk sesi ini, jangan gunakan `server`.
- Untuk mengatur ulang semua escape hatch ke nilai default-nya, jalankan `sp_babelfish_configure 'default'` (Babelfish versi 1.2.0 dan yang lebih tinggi).

String yang mengidentifikasi hatch dapat mencakup wildcard SQL. Misalnya, yang berikut ini menetapkan semua escape hatch sintaks ke `ignore` untuk klaster Aurora PostgreSQL.

```
EXECUTE sp_babelfish_configure '%', 'ignore', 'server'
```

Dalam tabel berikut, Anda dapat menemukan deskripsi dan nilai default untuk escape hatch Babelfish yang telah ditentukan sebelumnya.

Escape hatch	Deskripsi	Default
`escape_hatch_checkpoint`	Memungkinkan penggunaan pernyataan CHECKPOINT dalam kode prosedural, tetapi pernyataan CHECKPOINT saat ini tidak diterapkan.	abaikan
`escape_hatch_constraint_name_for_default`		abaikan

Escape hatch	Deskripsi	Default
	Mengontrol perilaku Babelfish yang terkait dengan nama batasan default.	
`escape_hatch_database_misc_options`	Mengontrol perilaku Babelfish terkait dengan opsi berikut pada CREATE atau ALTER DATABASE: CONTAINMENT, DB_CHAINING, TRUSTWORTHY, PERSISTENT_LOG_BUFFER.	abaikan
`escape_hatch_for_replication`	Mengontrol perilaku Babelfish yang terkait dengan klausa [NOT] FOR REPLICATION saat membuat atau mengubah tabel.	ketat
`escape_hatch_fulltext`	Mengontrol perilaku Babelfish yang terkait dengan fitur FULLTEXT, seperti DEFAULT_FULLTEXT_LANGUAGE dalam CREATE/ALTER DATABASE, CREATE FULLTEXT INDEX, atau sp_fulltext_database.	abaikan
`escape_hatch_ignore_dup_key`	Mengontrol perilaku Babelfish yang terkait dengan CREATE/ALTER TABLE dan CREATE INDEX. Ketika IGNORE_DUP_KEY=ON, memunculkan kesalahan saat disetel ke `strict` (default) atau mengabaikan kesalahan saat disetel ke `ignore` (Babelfish versi 1.2.0 dan lebih tinggi).	ketat

Escape hatch	Deskripsi	Default
`escape_hatch_index_clustering`	Mengontrol perilaku Babelfish yang terkait dengan kata kunci CLUSTERED atau NONCLUSTERED untuk indeks dan kendala PRIMARY KEY atau UNIQUE. Ketika CLUSTERED diabaikan, indeks atau kendala masih dibuat seolah-olah NONCLUSTERED ditentukan.	abaikan
`escape_hatch_index_columnstore`	Mengontrol perilaku Babelfish yang terkait dengan klausa COLUMNSTORE. Jika Anda menentukan `ignore`, Babelfish membuat indeks B-tree biasa.	ketat
`escape_hatch_join_hints`	Mengontrol perilaku kata kunci dalam operator JOIN: LOOP, HASH, MERGE, REMOTE, REDUCE, REDISTRIBUTE, REPLICATE.	abaikan

Escape hatch	Deskripsi	Default
escape_hatch_language_non_english	Mengontrol perilaku Babelfish yang terkait dengan bahasa selain bahasa Inggris untuk pesan di layar. Babelfish saat ini hanya mendukung us_english untuk pesan di layar. SET LANGUAGE mungkin menggunakan variabel yang berisi nama bahasa, sehingga bahasa sebenarnya yang disetel hanya dapat dideteksi pada runtime.	ketat
escape_hatch_login_hashed_password	Ketika diabaikan, menekan kesalahan untuk kata kunci HASHED untuk CREATE LOGIN dan ALTER LOGIN.	ketat
escape_hatch_login_misc_options	Ketika diabaikan, menekan kesalahan untuk kata kunci lain selain HASHED, MUST_CHANGE , OLD_PASSWORD , dan UNLOCK untuk CREATE LOGIN dan ALTER LOGIN.	ketat
escape_hatch_login_old_password	Ketika diabaikan, menekan kesalahan untuk kata kunci OLD_PASSWORD untuk CREATE LOGIN dan ALTER LOGIN.	ketat

Escape hatch	Deskripsi	Default
`escape_hatch_login_password_must_change`	Ketika diabaikan, menekan kesalahan untuk kata kunci `MUST_CHANGE` untuk `CREATE LOGIN` dan `ALTER LOGIN`.	ketat
`escape_hatch_login_password_unlock`	Ketika diabaikan, menekan kesalahan untuk kata kunci `UNLOCK` untuk `CREATE LOGIN` dan `ALTER LOGIN`.	ketat
`escape_hatch_nocheck_add_constraint`	Mengontrol perilaku Babelfish yang terkait dengan klausa WITH CHECK atau NOCHECK untuk kendala.	ketat
`escape_hatch_nocheck_existing_constraint`	Mengontrol perilaku Babelfish yang terkait dengan klausa FOREIGN KEY atau CHECK untuk kendala.	ketat
`escape_hatch_query_hints`	Mengontrol perilaku Babelfish yang terkait dengan petunjuk kueri. Ketika opsi ini diatur untuk mengabaikan, server mengabaikan petunjuk yang menggunakan klausa OPTION (...) untuk menentukan aspek pemrosesan kueri. Contohnya termasuk SELECT FROM ... OPTION(MERGE JOIN HASH, MAXRECURSION 10)).	abaikan

Escape hatch	Deskripsi	Default
`escape_hatch_rowversion`	Mengontrol perilaku tipe data ROWVERSION dan TIMESTAMP . Untuk informasi penggunaan, lihat Menggunakan fitur Babelfish dengan implementasi terbatas.	ketat
`escape_hatch_schemabinding_function`	Mengontrol perilaku Babelfish terkait dengan klausul WITH SCHEMABINDING. Secara default, klausa WITH SCHEMABINDING diabaikan ketika ditentukan dengan perintah CREATE atau ALTER FUNCTION.	abaikan
`escape_hatch_schemabinding_procedure`	Mengontrol perilaku Babelfish terkait dengan klausul WITH SCHEMABINDING. Secara default, klausa WITH SCHEMABINDING diabaikan ketika ditentukan dengan perintah CREATE atau ALTER PROCEDURE.	abaikan
`escape_hatch_rowguidcol_column`	Mengontrol perilaku Babelfish yang terkait dengan klausa ROWGUIDCOL saat membuat atau mengubah tabel.	ketat

Escape hatch	Deskripsi	Default
escape_hatch_schemabinding_trigger	Mengontrol perilaku Babelfish terkait dengan klausul WITH SCHEMABINDING. Secara default, klausa WITH SCHEMABINDING diabaikan ketika ditentukan dengan perintah CREATE atau ALTER TRIGGER.	abaikan
escape_hatch_schemabinding_view	Mengontrol perilaku Babelfish terkait dengan klausul WITH SCHEMABINDING. Secara default, klausa WITH SCHEMABINDING diabaikan ketika ditentukan dengan perintah CREATE atau ALTER VIEW.	abaikan
escape_hatch_session_settings	Mengontrol perilaku Babelfish terhadap pernyataan SET tingkat sesi yang tidak didukung.	abaikan
escape_hatch_showplan_all	Mengontrol perilaku Babelfish yang terkait dengan SET SHOWPLAN_ALL dan SET STATISTICS PROFILE. Ketika diatur untuk mengabaikan, perilakunya seperti SET BABELFISH_SHOWPLAN_ALL dan SET BABELFISH_STATISTICS PROFILE; ketika disetel ke ketat, pengabaiannya dilakukan secara diam-diam.	ketat

Escape hatch	Deskripsi	Default
`escape_hatch_storage_on_partition`	Mengontrol perilaku Babelfish yang terkait dengan klausa `ON partition_scheme column` saat mendefinisikan partisi. Babelfish saat ini tidak menerapkan partisi.	ketat
`escape_hatch_storage_options`	Escape hatch pada opsi penyimpanan apa pun yang digunakan dalam CREATE, ALTER DATABASE, TABLE, INDEX. Ini termasuk klausa (LOG) ON, TEXTIMAGE_ON, FILESTREAM_ON yang menentukan lokasi penyimpanan (partisi, grup file) untuk tabel, indeks, dan kendala, dan juga untuk basis data. Pengaturan escape hatch ini berlaku untuk semua klausa ini (termasuk ON [PRIMARY] dan ON "DEFAULT"). Pengecualiannya adalah ketika partisi ditentukan untuk tabel atau indeks dengan ON partition_scheme (kolom).	abaikan
`escape_hatch_table_hints`	Mengontrol perilaku petunjuk tabel yang ditentukan menggunakan klausa WITH (...).	abaikan

Escape hatch	Deskripsi	Default
`escape_hatch_unique_constraint`	Ketika diatur ke ketat, perbedaan semantik yang tidak jelas antara SQL Server dan PostgreSQL dalam menangani nilai NULL pada kolom yang diindeks dapat menimbulkan kesalahan . Perbedaan semantik hanya muncul dalam kasus penggunaa n yang tidak realistis, sehingga Anda dapat mengatur escape hatch ini menjadi 'abaikan' untuk menghindari melihat kesalahan.	ketat

Membuat klaster DB Babelfish for Aurora PostgreSQL

Babelfish for Aurora PostgreSQL didukung pada Aurora PostgreSQL versi 13.4 dan lebih baru.

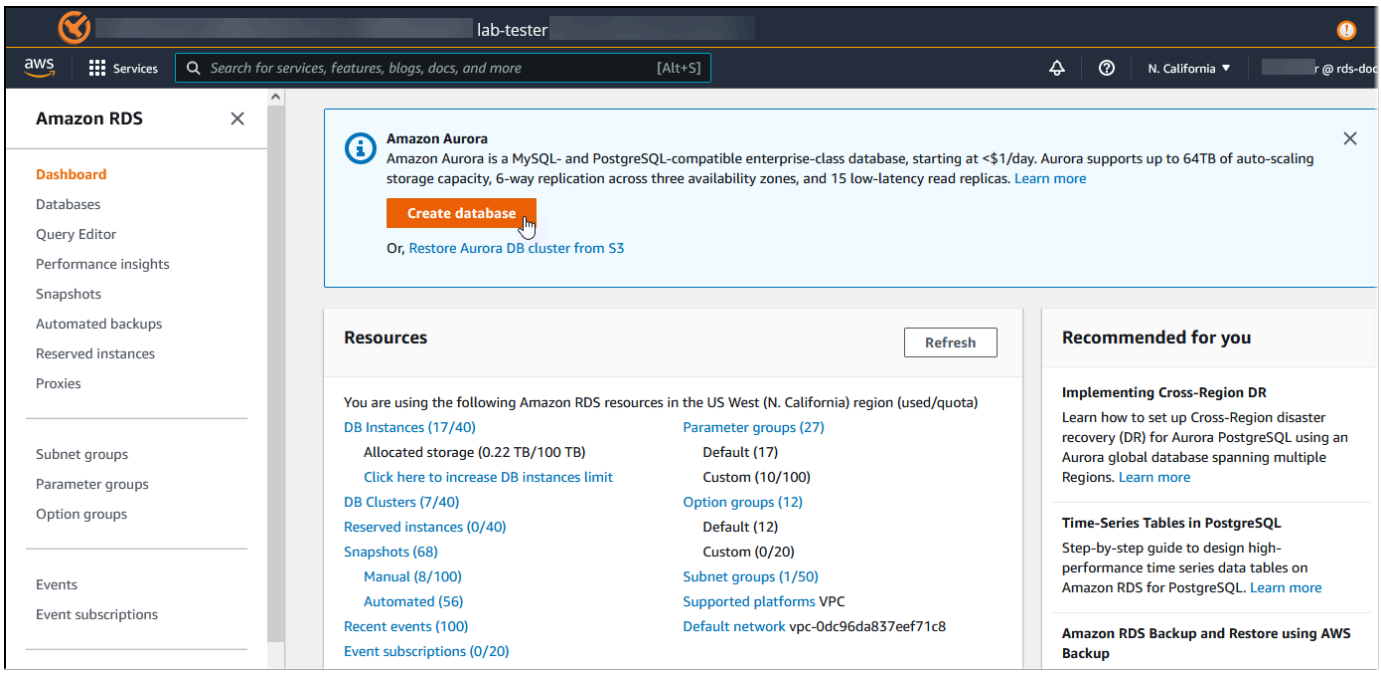
Anda dapat menggunakan AWS Management Console atau AWS CLI untuk membuat klaster Aurora PostgreSQL dengan Babelfish.

> **Note**
>
> Dalam klaster Aurora PostgreSQL, nama basis data `babelfish_db` dicadangkan untuk Babelfish. Membuat basis data "babelfish_db" Anda sendiri pada Babelfish for Aurora PostgreSQL mencegah Aurora berhasil menyediakan Babelfish.

Konsol

Untuk membuat klaster dengan Babelfish menjalankan dengan AWS Management Console

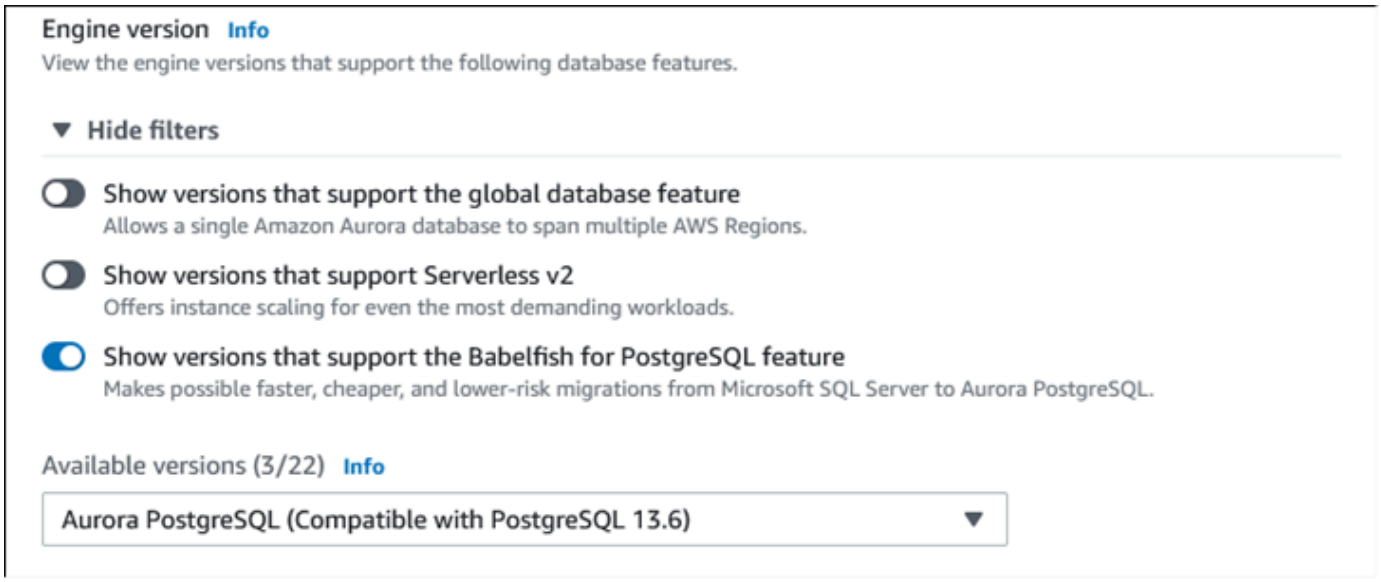
1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/, dan pilih Buat basis data.



2. Untuk Pilih metode pembuatan basis data, lakukan salah satu hal berikut:

   - Untuk menentukan opsi mesin terperinci, pilih Buat standar.
   - Untuk menggunakan opsi yang telah dikonfigurasi sebelumnya yang mendukung praktik terbaik untuk klaster Aurora, pilih Buat mudah.

3. Untuk Jenis mesin, pilih Aurora (Kompatibel dengan PostgreSQL).
4. Pilih Tampilkan filter, lalu pilih Tampilkan versi yang mendukung fitur Babelfish for PostgreSQL untuk mencantumkan jenis mesin yang mendukung Babelfish. Babelfish saat ini didukung pada Aurora PostgreSQL 13.4 dan versi yang lebih baru.
5. Untuk Versi yang tersedia, pilih versi Aurora PostgreSQL. Untuk mendapatkan fitur Babelfish terbaru, pilih versi utama Aurora PostgreSQL tertinggi.



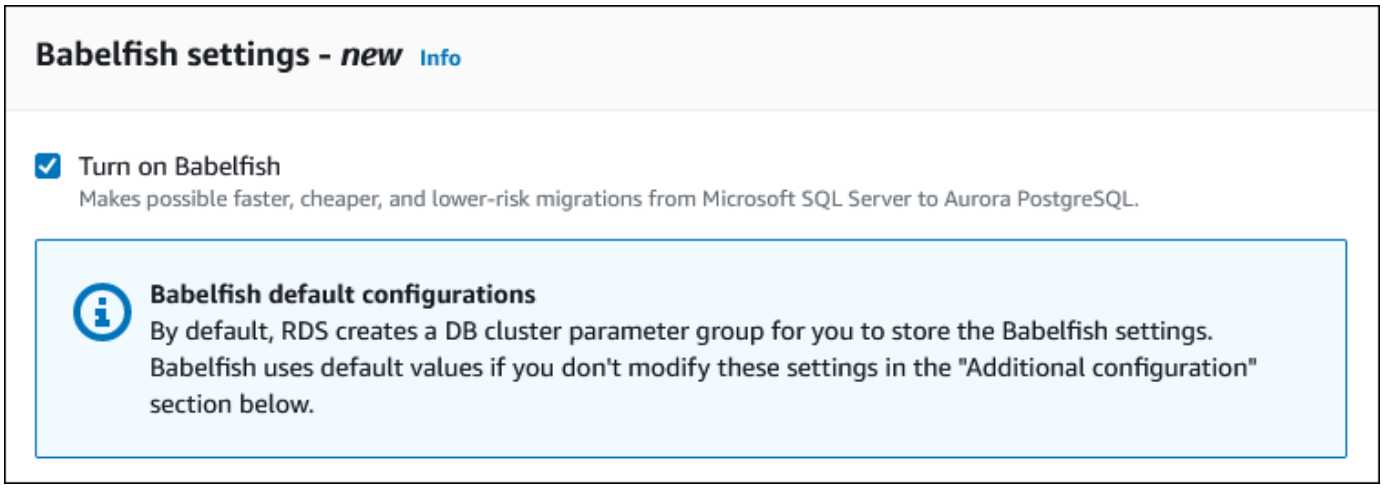
6. Untuk Templat, pilih templat yang sesuai dengan kasus penggunaan Anda.
7. Untuk Pengidentifikasi klaster DB, masukkan nama yang dapat Anda temukan dengan mudah nanti di daftar klaster DB.
8. Untuk Nama pengguna utama, masukkan nama pengguna administrator. Nilai default untuk Aurora PostgreSQL adalah `postgres`. Anda dapat menerima default atau memilih nama yang berbeda. Misalnya, untuk mengikuti konvensi penamaan yang digunakan pada basis data SQL Server Anda, Anda dapat memasukkan `sa` (administrator sistem) untuk Nama pengguna utama.

   Jika Anda tidak membuat nama pengguna `sa` saat ini, Anda dapat membuatnya nanti dengan klien pilihan Anda. Setelah membuat pengguna, gunakan perintah `ALTER SERVER ROLE` untuk menambahkannya ke grup (peran) `sysadmin` untuk klaster.

   > **Warning**
   >
   > Nama pengguna utama harus selalu menggunakan karakter huruf kecil yang gagal yang klaster DB tidak dapat terhubung ke Babelfish melalui port TDS.

9. Untuk Kata sandi utama, buat kata sandi yang kuat dan konfirmasi kata sandi.

10. Untuk opsi yang mengikuti, hingga bagian Pengaturan Babelfish, tentukan pengaturan klaster DB Anda. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.
11. Untuk membuat fungsionalitas Babelfish tersedia, pilih kotak Aktifkan Babelfish.



12. Untuk grup parameter klaster DB, lakukan salah satu hal berikut:

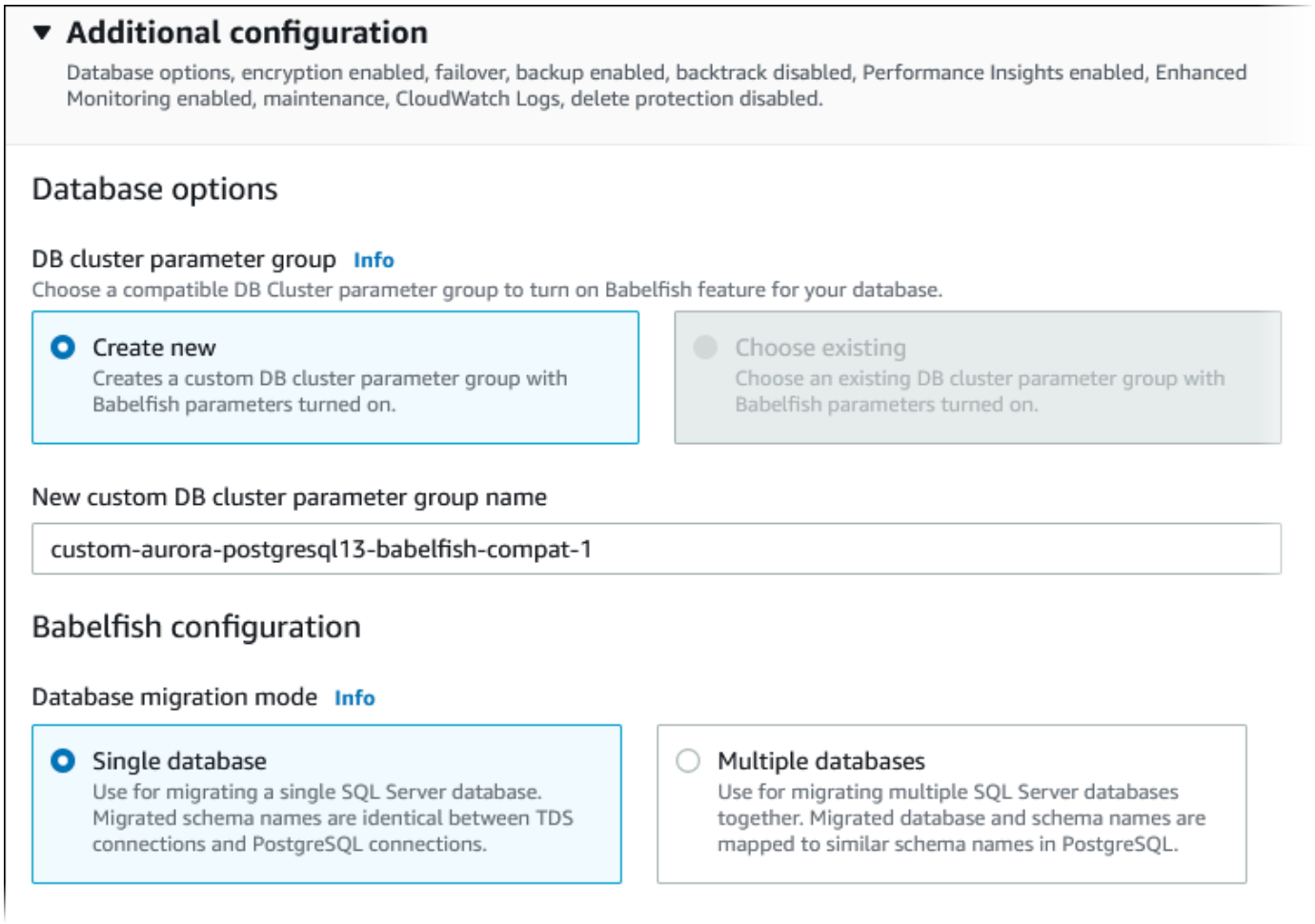
    - Pilih Buat baru untuk membuat grup parameter baru dengan Babelfish diaktifkan.
    - Klik Pilih yang ada untuk menggunakan grup parameter yang ada. Jika Anda menggunakan grup yang ada, pastikan untuk memodifikasi grup sebelum membuat klaster dan menambahkan nilai untuk parameter Babelfish. Untuk informasi tentang parameter Babelfish, lihat Pengaturan grup parameter klaster DB untuk Babelfish.

      Jika Anda menggunakan grup yang ada, berikan nama grup di kotak berikut.
13. Untuk Mode migrasi basis data, pilih salah satu langkah berikut:

    - Basis data tunggal untuk memigrasi basis data SQL Server tunggal.

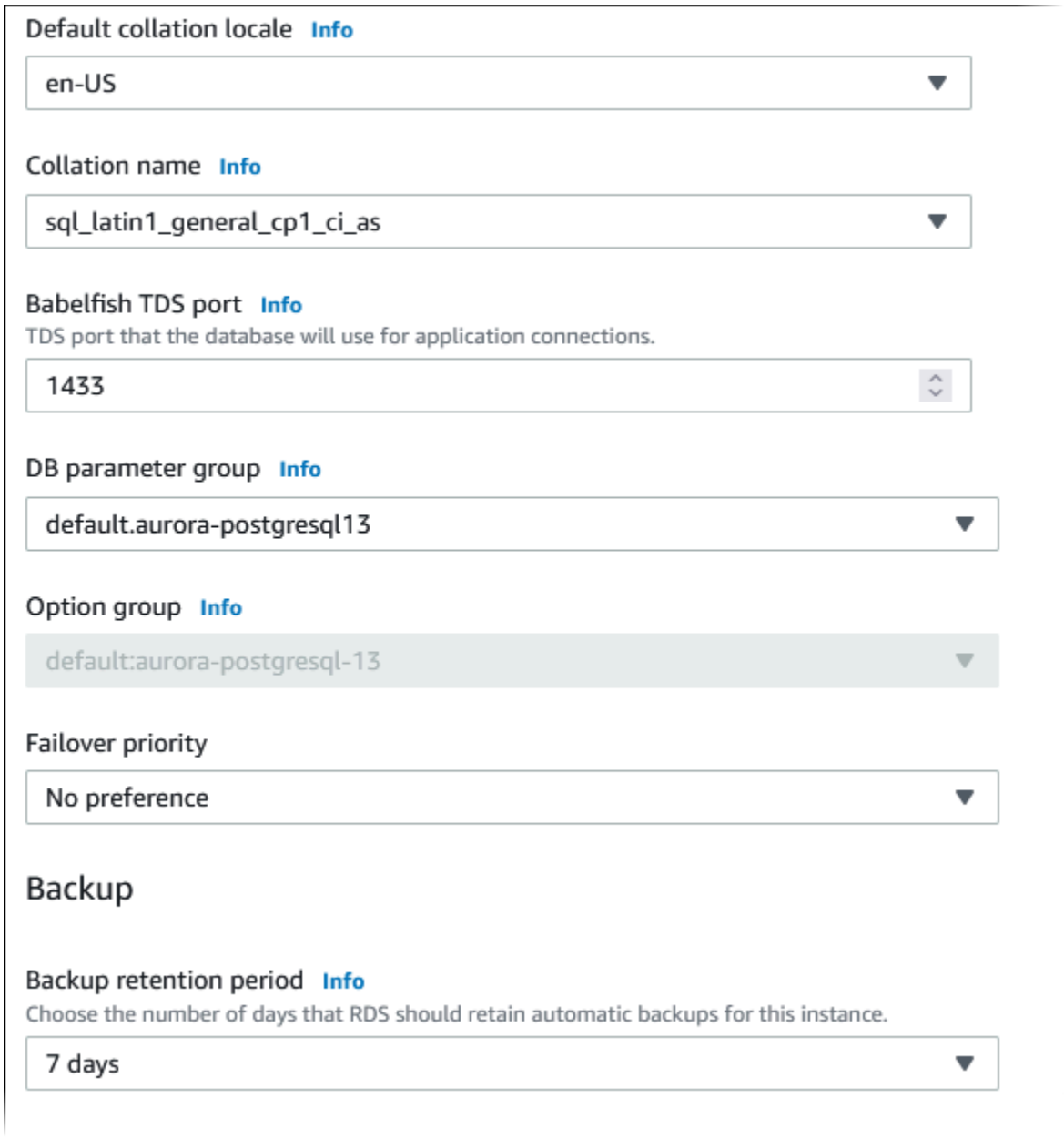
      Dalam beberapa kasus, Anda dapat memigrasikan beberapa basis data pengguna bersama-sama, dengan tujuan akhir Anda migrasi lengkap ke Aurora PostgreSQL asli tanpa Babelfish. Jika aplikasi akhir memerlukan skema konsolidasi (skema dbo tunggal), pastikan untuk terlebih dahulu mengkonsolidasikan basis data SQL Server Anda ke dalam basis data SQL server tunggal. Kemudian bermigrasi ke Babelfish menggunakan mode Basis data tunggal.
    - Beberapa basis data untuk memigrasi beberapa basis data SQL Server (berasal dari instalasi SQL Server tunggal). Beberapa mode basis data tidak mengkonsolidasikan beberapa basis data yang tidak berasal dari instalasi SQL Server tunggal. Untuk informasi tentang memigrasi beberapa basis data, lihat Menggunakan Babelfish dengan satu atau beberapa basis data.

> **Note**
>
> Dari versi Aurora PostgreSQL 16, Beberapa database dipilih secara default sebagai mode migrasi Database.



14. Untuk lokal kolasi Default, masukkan lokal server Anda. Default-nya adalah `en-US`. Untuk informasi mendetail tentang penampungan, lihat Kolasi yang didukung oleh Babelfish.
15. Untuk kolom Nama kolasi, masukkan kolasi default Anda. Default-nya adalah `sql_latin1_general_cp1_ci_as`. Untuk detail informasi, lihat Kolasi yang didukung oleh Babelfish.
16. Untuk Port TDS Babelfish, masukkan port default `1433`. Saat ini, Babelfish hanya mendukung port `1433` untuk klaster DB Anda.

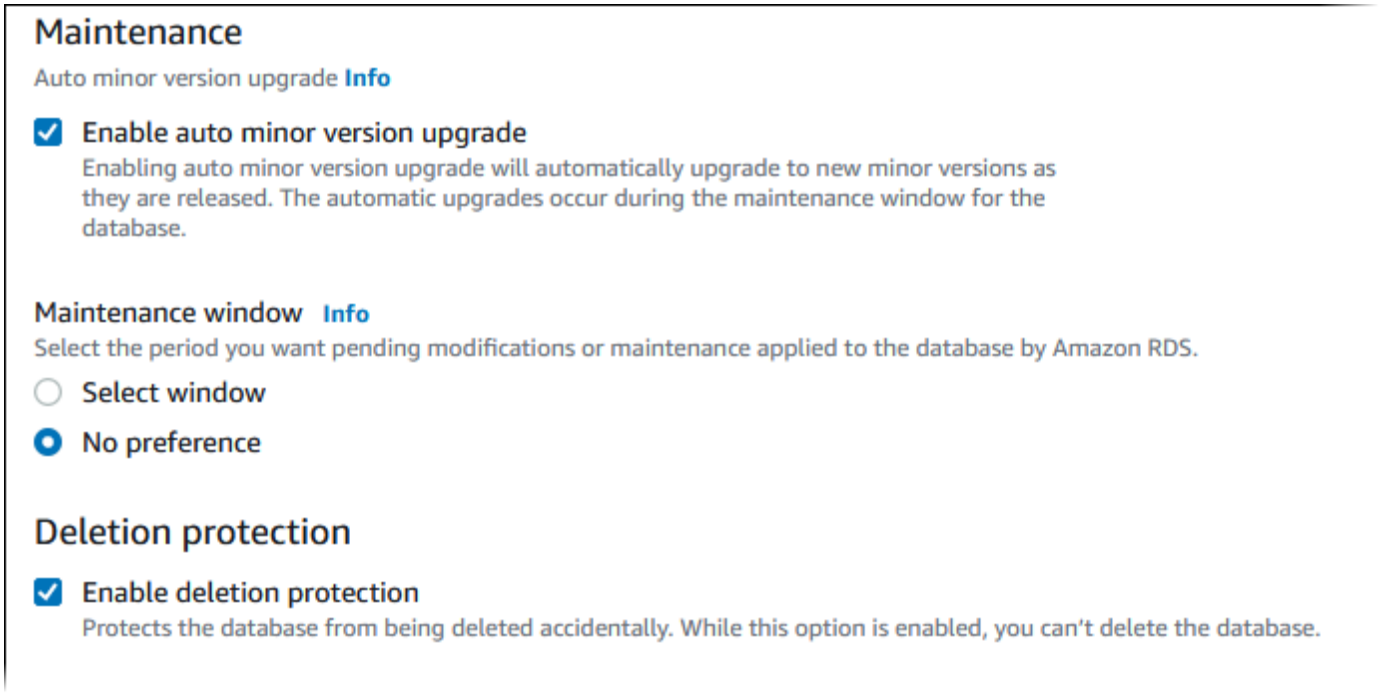
17. Untuk grup parameter DB, pilih grup parameter atau minta Aurora membuat grup baru untuk Anda dengan pengaturan default.
18. Untuk Prioritas failover, pilih prioritas failover untuk instans. Jika Anda tidak memilih suatu nilai, maka nilai defaultnya adalah `tier-1`. Prioritas ini menentukan urutan di mana replika dipromosikan saat pulih dari kegagalan instans utama. Untuk informasi selengkapnya, lihat Toleransi kesalahan untuk klaster DB Aurora.
19. Untuk Periode retensi cadangan, pilih durasi waktu (1–35 hari) Aurora mempertahankan salinan cadangan basis data. Anda dapat menggunakan salinan cadangan untuk point-in-time mengembalikan (PITR) database Anda hingga yang kedua. Periode penyimpanan default adalah tujuh hari.



20. Pilih Salin tag ke snapshot untuk menyalin tag instans DB apa pun ke snapshot DB saat Anda membuat snapshot.
21. Pilih Aktifkan enkripsi untuk mengaktifkan enkripsi saat istirahat (enkripsi penyimpanan Aurora) untuk klaster DB ini.
22. Pilih Aktifkan Wawasan Performa untuk mengaktifkan Wawasan Performa Amazon RDS.
23. Pilih Aktifkan Pemantauan lanjutan untuk mulai mengumpulkan metrik secara waktu nyata untuk sistem operasi tempat klaster DB Anda dijalankan.
24. Pilih log PostgreSQL untuk mempublikasikan file log ke Amazon Logs. CloudWatch
25. Pilih Aktifkan peningkatan versi minor otomatis untuk memperbarui klaster DB Aurora Anda secara otomatis saat peningkatan versi minor tersedia.
26. Untuk Jendela pemeliharaan, lakukan hal berikut:

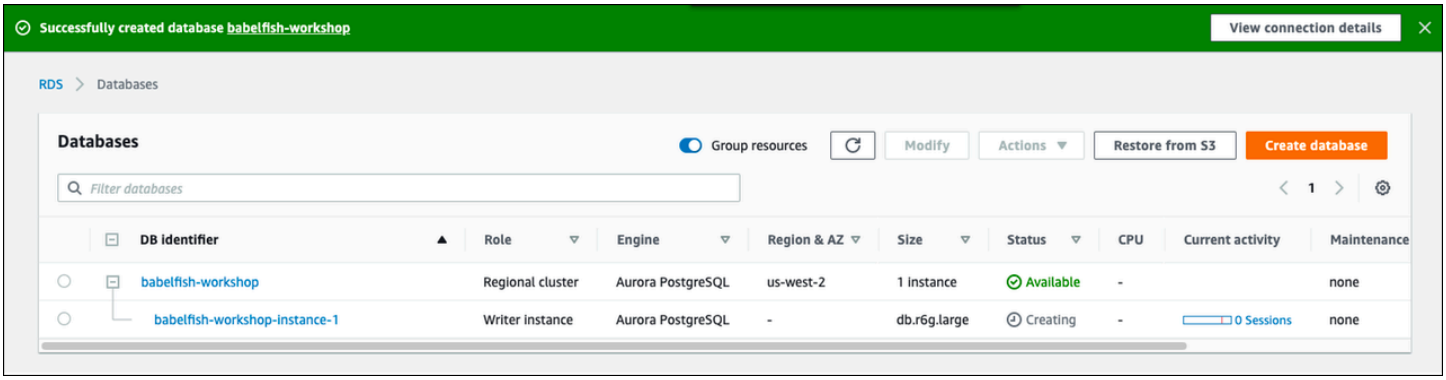
    - Untuk memilih waktu bagi Amazon RDS untuk melakukan modifikasi atau melakukan pemeliharaan, klik Pilih jendela.
    - Untuk melakukan pemeliharaan Amazon RDS pada waktu yang tidak terjadwal, pilih Tidak ada preferensi.
27. Pilih kotak Aktifkan perlindungan penghapusan untuk melindungi basis data Anda agar tidak terhapus secara tidak sengaja.

    Jika Anda mengaktifkan fitur ini, Anda tidak dapat langsung menghapus basis data. Namun, Anda perlu memodifikasi klaster basis data dan mematikan fitur ini sebelum menghapus basis data.



28. Pilih Buat basis data.

Anda dapat menemukan basis data baru yang disiapkan untuk Babelfish di daftar Basis data. Kolom Status ditampilkan Tersedia saat deployment selesai.



AWS CLI

Saat Anda membuat Babelfish for Aurora PostgreSQL; menggunakan AWS CLI, Anda harus meneruskan perintah nama grup parameter klaster DB yang akan digunakan untuk klaster. Untuk informasi selengkapnya, lihat Prasyarat klaster DB.

Sebelum Anda dapat menggunakan AWS CLI untuk membuat klaster Aurora PostgreSQL dengan Babelfish, lakukan hal berikut:

- Pilih URL titik akhir Anda dari daftar layanan di titik akhir dan kuota Amazon Aurora.

- Buat grup parameter untuk klaster. Untuk informasi selengkapnya tentang grup parameter, lihat Bekerja dengan grup parameter.
- Ubah grup parameter, tambahkan parameter yang mengaktifkan Babelfish.

Untuk membuat klaster DB Aurora PostgreSQL dengan Babelfish menggunakan AWS CLI

Contoh yang mengikuti menggunakan Nama pengguna utama default, `postgres`. Ganti sesuai kebutuhan dengan nama pengguna yang Anda buat untuk klaster DB Anda, seperti `sa` atau nama pengguna apa pun yang Anda pilih jika Anda tidak menerima default.

1. Buat grup parameter.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-cluster-parameter-group \
   --endpoint-url endpoint-url \
   --db-cluster-parameter-group-name parameter-group \
   --db-parameter-group-family aurora-postgresql14 \
   --description "description"
   ```

   Untuk Windows:

   ```
   aws rds create-db-cluster-parameter-group ^
   --endpoint-url endpoint-URL ^
   --db-cluster-parameter-group-name parameter-group ^
   --db-parameter-group-family aurora-postgresql14 ^
   --description "description"
   ```

2. Ubah grup parameter Anda untuk mengaktifkan Babelfish.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds modify-db-cluster-parameter-group \
   --endpoint-url endpoint-url \
   --db-cluster-parameter-group-name parameter-group \
   --parameters
    "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
   ```

   Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
--endpoint-url endpoint-url ^
--db-cluster-parameter-group-name paramater-group ^
--parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot"
```

3. Identifikasi grup subnet DB Anda dan ID grup keamanan virtual private cloud (VPC) untuk cluster DB baru Anda, lalu panggil create-db-clusterperintahnya.

   Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
--db-cluster-identifier  cluster-name\
--master-username postgres \
--manage-master-user-password \
--engine aurora-postgresql \
--engine-version 14.3            \
--vpc-security-group-ids security-group \
--db-subnet-group-name subnet-group-name \
--db-cluster-parameter-group-name parameter-group
```

   Untuk Windows:

```
aws rds create-db-cluster ^
--db-cluster-identifier cluster-name ^
--master-username postgres ^
--manage-master-user-password ^
--engine aurora-postgresql ^
--engine-version 14.3 ^
--vpc-security-group-ids security-group ^
--db-subnet-group-name subnet-group ^
--db-cluster-parameter-group-name parameter-group
```

   Contoh ini menentukan `--manage-master-user-password` opsi untuk menghasilkan kata sandi pengguna utama dan mengelolanya di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. Atau, Anda dapat menggunakan opsi `--master-password` untuk menentukan dan mengelola kata sandi sendiri.

4. Buat instans utama untuk klaster DB Anda. Gunakan nama cluster yang Anda buat di langkah 3 untuk `--db-cluster-identifier` argumen saat Anda memanggil create-db-instanceperintah, seperti yang ditunjukkan berikut.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-instance \
   --db-instance-identifier instance-name \
   --db-instance-class db.r6g \
   --db-subnet-group-name subnet-group \
   --db-cluster-identifier cluster-name \
   --engine aurora-postgresql
   ```

   Untuk Windows:

   ```
   aws rds create-db-instance ^
   --db-instance-identifier instance-name ^
   --db-instance-class db.r6g ^
   --db-subnet-group-name subnet-group ^
   --db-cluster-identifier cluster-name ^
   --engine aurora-postgresql
   ```

Memigrasi basis data SQL Server ke Babelfish for Aurora PostgreSQL

Anda dapat menggunakan Babelfish for Aurora PostgreSQL untuk memigrasi dari basis data SQL Server ke klaster DB Amazon Aurora PostgreSQL. Sebelum bermigrasi, tinjau Menggunakan Babelfish dengan satu atau beberapa basis data.

Topik

- Ikhtisar proses migrasi
- Mengevaluasi dan menangani perbedaan antara SQL Server dan Babelfish
- Alat impor/ekspor untuk bermigrasi dari SQL Server ke Babelfish

Ikhtisar proses migrasi

Ringkasan berikut mencantumkan langkah-langkah yang diperlukan untuk menyukseskan proses migrasi aplikasi SQL Server Anda dan membuatnya bekerja dengan Babelfish. Untuk informasi tentang alat yang dapat Anda gunakan untuk proses ekspor dan impor dan untuk detail selengkapnya, lihat Alat impor/ekspor untuk bermigrasi dari SQL Server ke Babelfish.

1. Buat klaster DB Aurora PostgreSQL baru dengan kondisi Babelfish diaktifkan. Untuk mempelajari caranya, lihat Membuat klaster DB Babelfish for Aurora PostgreSQL.

   Untuk mengimpor berbagai artefak SQL yang diekspor dari basis data SQL Server Anda, sambungkan ke klaster Babelfish menggunakan alat SQL Server seperti sqlcmd. Untuk informasi selengkapnya, lihat Menggunakan klien SQL Server untuk terhubung ke klaster DB Anda.
2. Pada basis data SQL Server yang ingin Anda migrasikan, ekspor bahasa definisi data (DDL). DDL adalah kode SQL yang menggambarkan objek basis data yang berisi data pengguna (seperti tabel, indeks, dan tampilan) dan kode basis data yang ditulis pengguna (seperti prosedur tersimpan, fungsi yang ditentukan pengguna, dan pemicu).

   Untuk informasi selengkapnya, lihat Menggunakan SQL Server Management Studio (SSMS) untuk memigrasi ke Babelfish.
3. Jalankan alat penilaian untuk mengevaluasi ruang lingkup perubahan apa pun yang mungkin perlu Anda buat sehingga Babelfish dapat secara efektif mendukung aplikasi yang berjalan di SQL Server. Untuk informasi selengkapnya, lihat Mengevaluasi dan menangani perbedaan antara SQL Server dan Babelfish.
4. Untuk memuat data, sebaiknya gunakan AWS DMS dengan Babelfish atau Aurora PostgreSQL sebagai titik akhir target berdasarkan persyaratan migrasi Anda. Pastikan untuk memodifikasi

kolom dengan tipe data Babelfish yang direkomendasikan. Untuk melakukannya, lihat Prasyarat untuk menggunakan Babelfish sebagai target untuk AWS DMS.

5. Pada klaster DB Babelfish baru Anda, jalankan DDL dalam basis data T-SQL yang Anda tentukan untuk membuat hanya skema, tipe data yang ditentukan pengguna, dan tabel dengan batasan kunci primernya.
6. Gunakan AWS DMS untuk memigrasikan data Anda dari SQL Server ke tabel Babelfish. Untuk replikasi berkelanjutan menggunakan SQL Server Change Data Capture atau SQL Replication, gunakan Aurora PostgreSQL alih-alih Babelfish sebagai titik akhir. Untuk melakukannya, lihat Menggunakan Babelfish for Aurora PostgreSQL sebagai target untuk AWS Database Migration Service.
7. Saat pemuatan data selesai, buat semua objek T-SQL yang tersisa yang mendukung aplikasi di klaster Babelfish Anda.
8. Konfigurasikan ulang aplikasi klien Anda untuk terhubung ke titik akhir Babelfish alih-alih basis data SQL Server Anda. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster DB Babelfish.
9. Ubah aplikasi Anda sesuai kebutuhan dan uji ulang. Untuk informasi selengkapnya, lihat Perbedaan antara Babelfish for Aurora PostgreSQL dan SQL Server.

Anda masih perlu menilai kueri SQL sisi klien Anda. Skema yang dihasilkan dari instans SQL Server Anda hanya mengonversi kode SQL sisi server. Sebaiknya Anda melakukan tindakan berikut:

- Tangkap kueri sisi klien dengan menggunakan SQL Server Profiler dengan templat yang telah ditentukan TSQL_Replay. Templat ini menangkap informasi pernyataan T-SQL yang kemudian dapat Anda putar ulang untuk penyesuaian dan pengujian berulang. Anda dapat memulai profiler dalam SQL Server Management Studio dari menu Alat. Pilih SQL Server Profiler untuk membuka profiler dan memilih templat TSQL_Replay.

  Untuk penggunaan pada migrasi Babelfish Anda, mulai jejak dan kemudian jalankan aplikasi Anda menggunakan pengujian fungsional Anda. Profiler menangkap pernyataan T-SQL. Saat Anda selesai menguji, hentikan jejaknya. Simpan hasilnya ke file XML dengan kueri sisi klien Anda (File > Save as > Trace XMLFile for Replay).

  Untuk informasi selengkapnya, lihat SQL Server Profiler di dokumentasi Microsoft. Untuk informasi selengkapnya tentang templat TSQL_Replay, lihat Templat SQL Server Profiler.
- Untuk aplikasi dengan kueri SQL sisi klien yang kompleks, kami menyarankan Anda menggunakan Babelfish Compass untuk menganalisis kueri ini untuk kompatibilitas Babelfish. Jika analisis

menunjukkan bahwa pernyataan SQL sisi klien berisi fitur SQL yang tidak didukung, tinjau aspek SQL dalam aplikasi klien dan modifikasi sesuai kebutuhan.

- Anda juga dapat menangkap kueri SQL sebagai peristiwa yang diperpanjang (format .xel). Untuk melakukannya, gunakan SSMS XEvent Profiler. Setelah membuat file .xel, ekstrak pernyataan SQL ke dalam file .xml yang kemudian dapat diproses oleh Compass. Untuk informasi selengkapnya, lihat Menggunakan SSMS XEvent Profler di dokumentasi Microsoft.

Ketika Anda puas dengan semua pengujian, analisis, dan modifikasi apa pun yang diperlukan untuk aplikasi yang dimigrasi, Anda dapat mulai menggunakan basis data Babelfish untuk produksi. Untuk melakukannya, hentikan basis data asli dan alihkan aplikasi klien langsung untuk menggunakan port TDS Babelfish.

Mengevaluasi dan menangani perbedaan antara SQL Server dan Babelfish

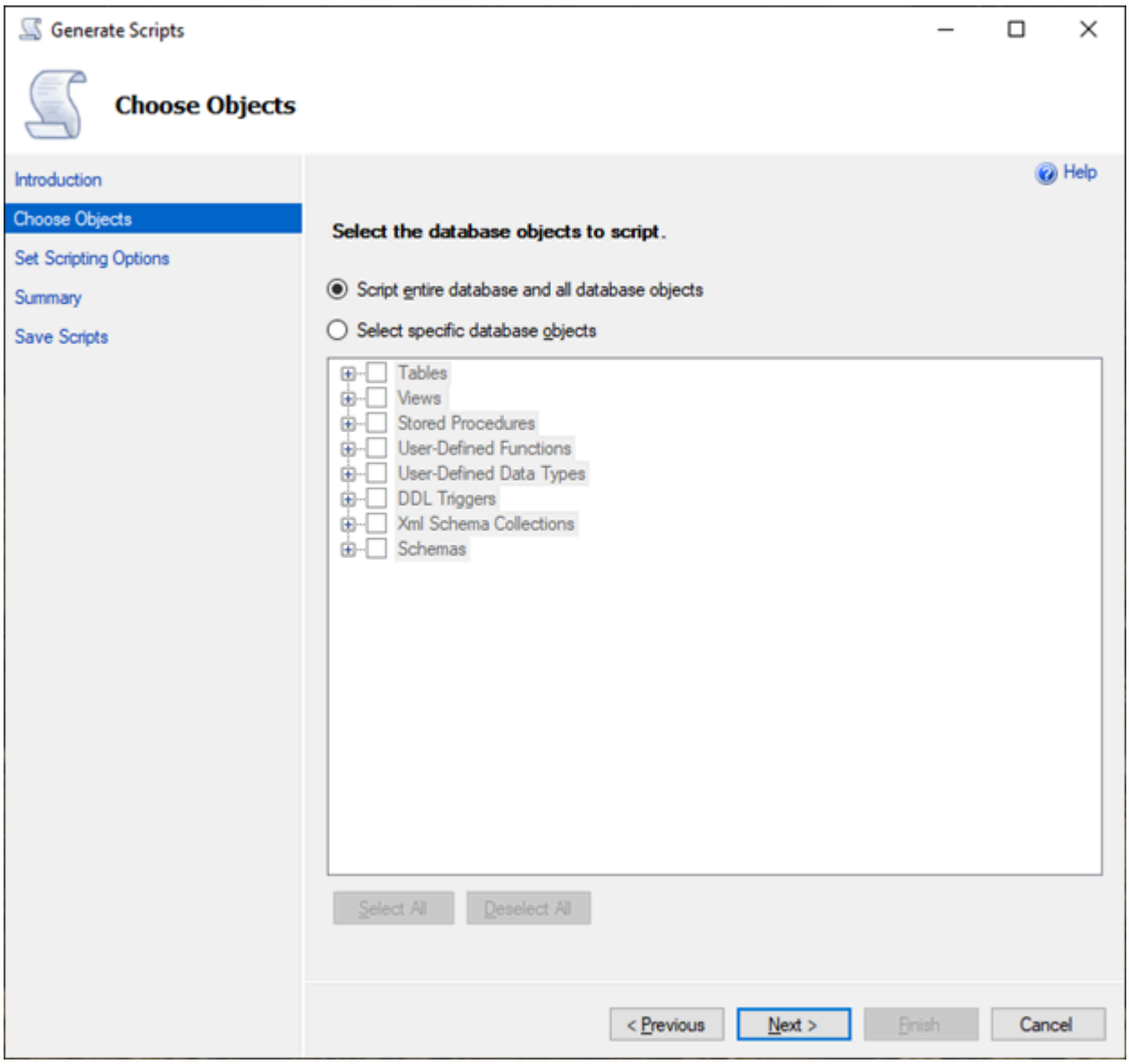
Untuk hasil terbaik, kami sarankan Anda mengevaluasi DDL/DML dan kode kueri klien yang dihasilkan sebelum benar-benar memigrasikan aplikasi basis data SQL Server Anda ke Babelfish. Bergantung pada versi Babelfish dan fitur spesifik SQL Server yang diterapkan aplikasi Anda, Anda mungkin perlu memfaktor ulang aplikasi Anda atau menggunakan alternatif untuk fungsionalitas yang belum sepenuhnya didukung di Babelfish.

- Untuk menilai kode aplikasi SQL Server Anda, gunakan Babelfish Compass pada DDL yang dihasilkan untuk menentukan berapa banyak kode T-SQL yang didukung oleh Babelfish. Identifikasi kode T-SQL yang mungkin memerlukan modifikasi sebelum berjalan di Babelfish. Untuk informasi lebih lanjut tentang alat ini, lihat Alat Kompas Babelfish di. GitHub

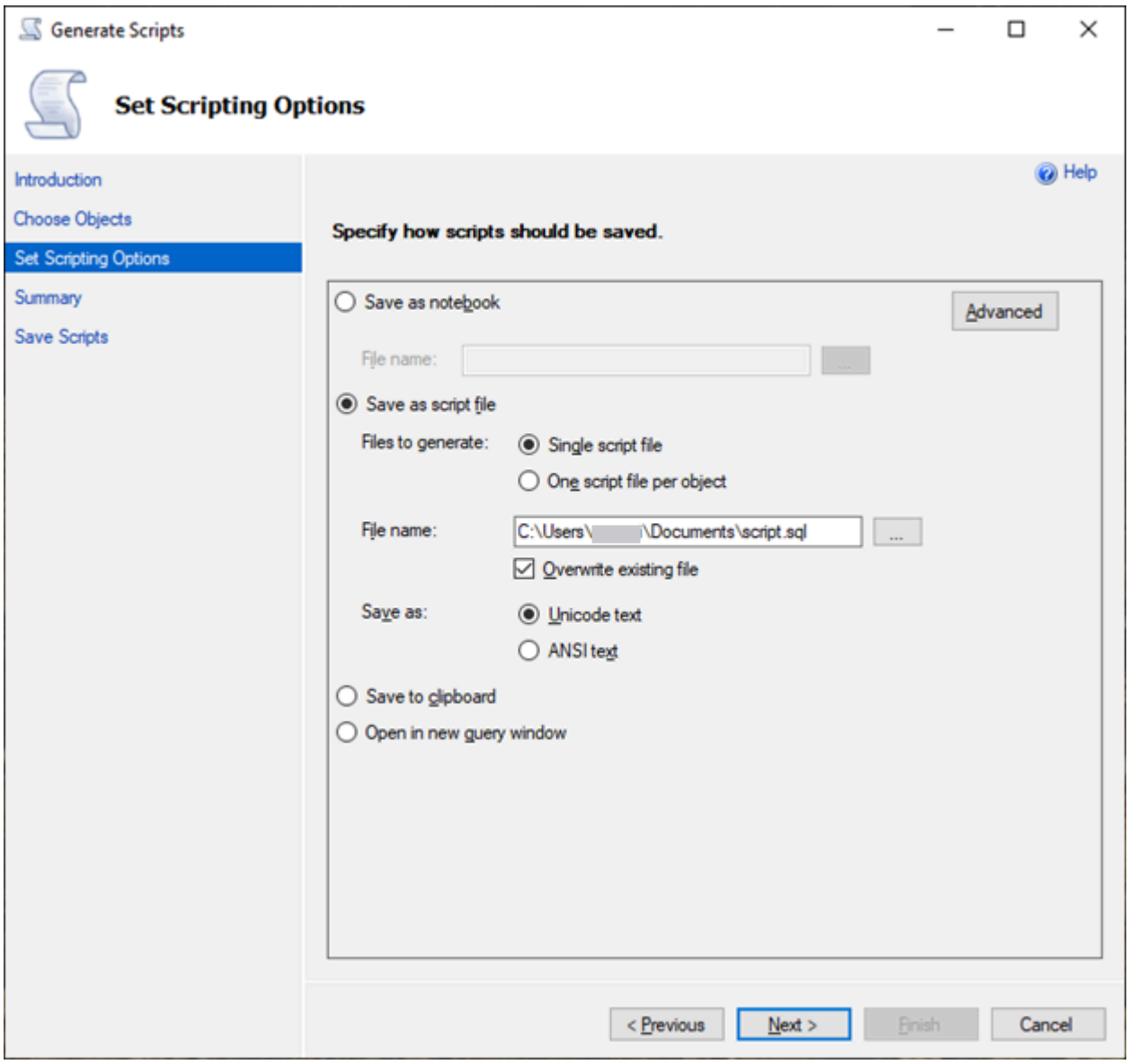
> **Note**
>
> Babelfish Compass adalah alat sumber terbuka. Laporkan masalah apa pun dengan Babelfish Compass melalui GitHub alih-alih melalui Support. AWS

Anda dapat menggunakan Wizard Generate Script dengan SQL Server Management Studio (SSMS) untuk menghasilkan file SQL yang dinilai oleh Babelfish Compass atau CLI AWS Schema Conversion Tool. Kami merekomendasikan langkah-langkah berikut untuk mengefisienkan penilaian.

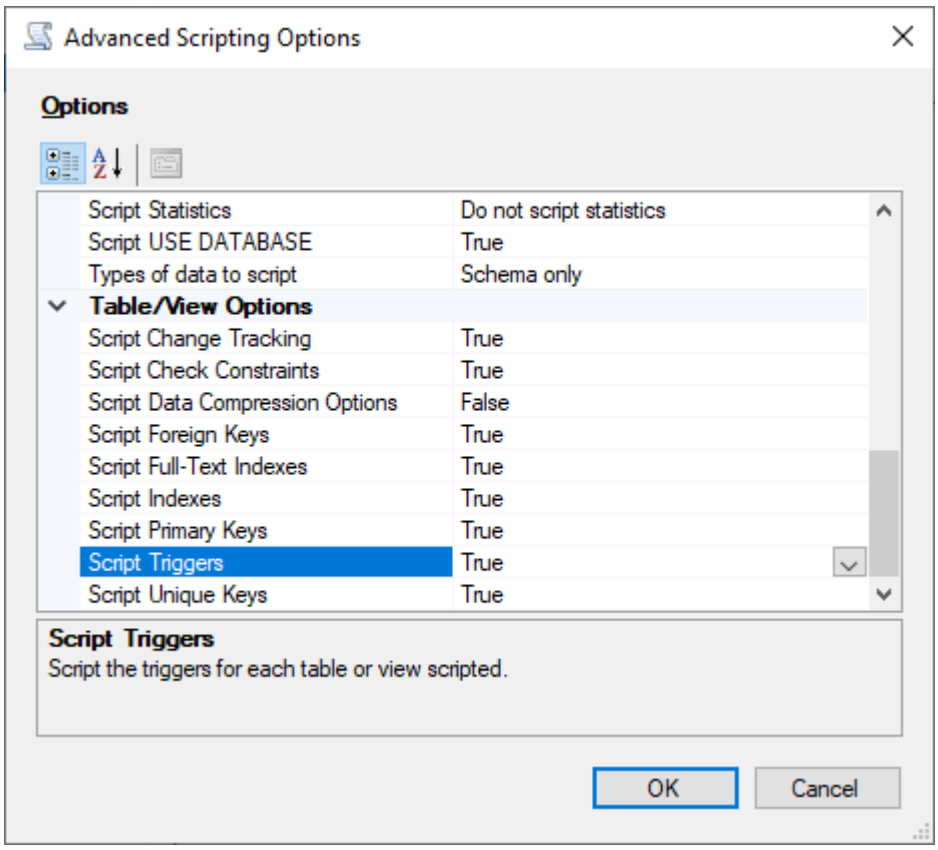
1. Pada halaman Choose Objects, pilih Script entire database and all database objects.



2. Untuk Set Scripting Options, pilih Save as script file sebagai Single script file.



3. Pilih Advanced untuk mengubah opsi pembuatan skrip default untuk mengidentifikasi fitur yang biasanya disetel ke false untuk penilaian lengkap:
   - Script Change Tracking ke True
   - Script Full-Text Indexes ke True
   - Script Triggers ke True
   - Script Logins ke True
   - Script Owner ke True
   - Script Object-Level Permissions ke True
   - Script Collations ke True



4. Lakukan langkah-langkah selanjutnya di wizard untuk membuat file.

Alat impor/ekspor untuk bermigrasi dari SQL Server ke Babelfish

Sebaiknya gunakan AWS DMS sebagai alat utama untuk bermigrasi dari SQL Server ke Babelfish. Namun, Babelfish mendukung beberapa cara lain untuk memigrasikan data menggunakan alat SQL Server yang mencakup yang berikut ini.

- SQL Server Integration Services (SSIS) untuk semua versi Babelfish. Untuk informasi selengkapnya, lihat Migrasi dari SQL Server ke Aurora PostgreSQL menggunakan SSIS dan Babelfish.
- Gunakan Wizard Impor/Ekspor SSMS untuk Babelfish versi 2.1.0 dan yang lebih baru. Alat ini tersedia melalui SSMS, tetapi juga tersedia sebagai alat mandiri. Untuk informasi selengkapnya, lihat Selamat Datang di Wizard Impor dan Ekspor SQL Server di dokumentasi Microsoft.
- Utilitas program salinan data massal (bcp) Microsoft memungkinkan Anda menyalin data dari instans Microsoft SQL Server ke file data dalam format yang Anda tentukan. Untuk informasi selengkapnya, lihat Utilitas bcp di dokumentasi Microsoft. Dukungan untuk migrasi data dengan menggunakan klien BCP dan utilitas bcp kini mendukung bendera `-E` (untuk kolom identitas) dan

bendera -b (untuk sisipan batch). Opsi bcp tertentu tidak didukung, termasuk `-C`, `-T`, `-G`, `-K`, `-R`, `-V`, dan `-h`.

Menggunakan SQL Server Management Studio (SSMS) untuk memigrasi ke Babelfish

Sebaiknya buat file terpisah untuk masing-masing jenis objek tertentu. Anda dapat menggunakan wizard Generate Scripts di SSMS untuk setiap set pernyataan DDL terlebih dahulu, dan kemudian memodifikasi objek sebagai grup untuk memperbaiki masalah yang ditemukan selama penilaian.

Lakukan langkah-langkah ini untuk memigrasikan data menggunakan AWS DMS atau metode migrasi data lainnya. Jalankan jenis skrip create ini terlebih dahulu untuk pendekatan yang lebih baik dan lebih cepat untuk memuat data pada tabel Babelfish di Aurora PostgreSQL.

1. Jalankan pernyataan `CREATE SCHEMA`.
2. Jalankan pernyataan `CREATE TYPE` untuk membuat tipe data yang ditentukan pengguna.
3. Jalankan pernyataan `CREATE TABLE` dasar dengan kunci primer atau kendala unik.

Lakukan pemuatan data menggunakan alat impor/ekspor yang disarankan. Jalankan skrip yang dimodifikasi untuk langkah-langkah berikut untuk menambahkan objek basis data yang tersisa. Anda memerlukan pernyataan create table untuk menjalankan skrip ini untuk kendala, pemicu, dan indeks. Setelah skrip dibuat, hapus pernyataan create table.

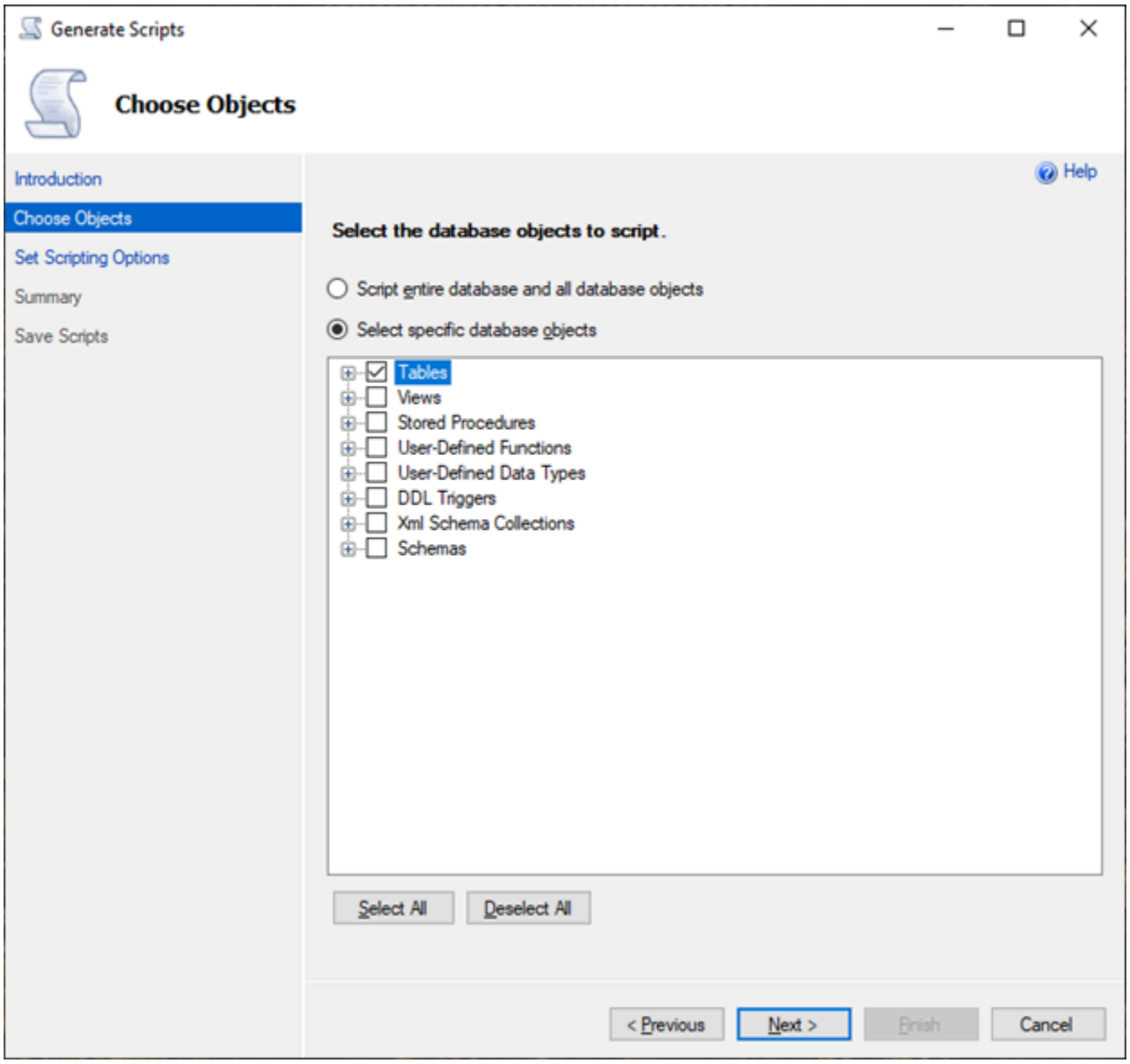
1. Jalankan pernyataan `ALTER TABLE` untuk kendala check, kendala foreign key, kendala default.
2. Jalankan pernyataan `CREATE TRIGGER`.
3. Jalankan pernyataan `CREATE INDEX`.
4. Jalankan pernyataan `CREATE VIEW`.
5. Jalankan pernyataan `CREATE STORED PROCEDURE`.

Untuk membuat skrip untuk setiap jenis objek

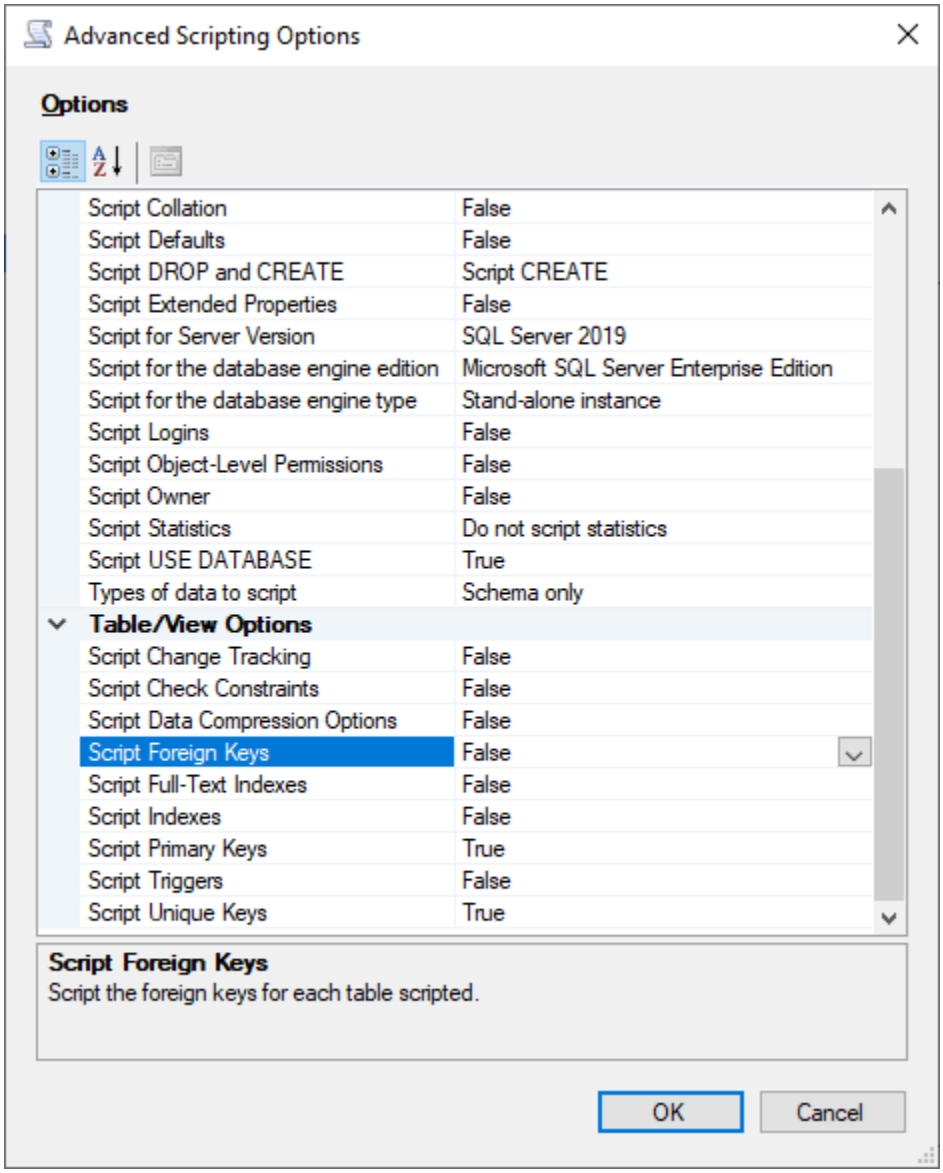
Gunakan langkah-langkah berikut untuk membuat pernyataan create table dasar menggunakan wizard Generate Scripts di SSMS. Ikuti langkah yang sama untuk membuat skrip untuk tipe objek yang berbeda.

1. Hubungkan ke instans SQL Server yang ada.
2. Buka menu konteks (klik kanan) untuk nama basis data.

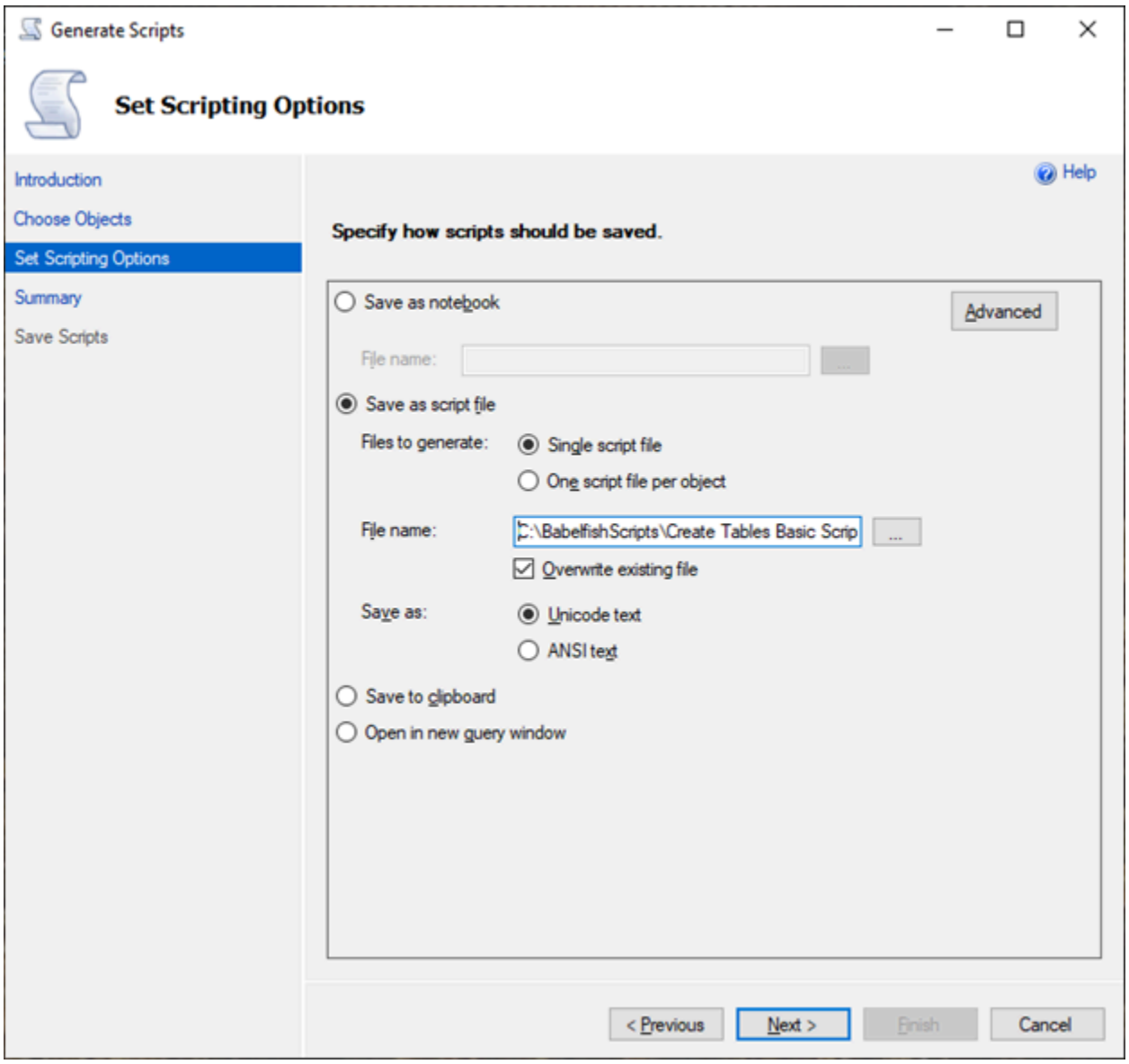
3. Pilih Tasks, lalu pilih Generate Scripts....
4. Pada panel Choose Objects, pilih Select specific database objects. Pilih Table, pilih semua tabel. Pilih Next untuk melanjutkan.



5. Pada halaman Set Scripting Options, pilih Advanced untuk membuka pengaturan Options. Untuk membuat pernyataan create table dasar, ubah nilai default berikut:
   - Script Defaults ke False.
   - Script Extended Properties ke False. Babelfish tidak mendukung properti yang diperluas.
   - Script Check Constraints ke False. Script Foreign Keys ke False.



6. Pilih OK.
7. Pada halaman Set Scripting Options, pilih Save as script file lalu pilih opsi Single script file. Masukkan nama File name Anda.



8. Pilih Next untuk melihat halaman Summary wizard.
9. Pilih Next untuk memulai pembuatan skrip.

   Anda dapat terus menghasilkan skrip untuk jenis objek lain di wizard. Alih-alih memilih Finish setelah file disimpan, pilih tombol Previous tiga kali untuk kembali ke halaman Choose Objects. Kemudian ulangi langkah-langkah di wizard untuk membuat skrip untuk jenis objek lainnya.

Autentikasi basis data dengan Babelfish for Aurora PostgreSQL

Babelfish for Aurora PostgreSQL mendukung dua cara untuk mengautentikasi pengguna basis data. Autentikasi kata sandi tersedia secara default untuk semua klaster DB Babelfish. Anda juga dapat menambahkan autentikasi Kerberos untuk klaster DB yang sama.

Topik

- Autentikasi kata sandi dengan Babelfish
- Autentikasi Kerberos dengan Babelfish

Autentikasi kata sandi dengan Babelfish

Babelfish for Aurora PostgreSQL mendukung autentikasi kata sandi. Kata sandi disimpan dalam bentuk terenkripsi pada disk. Untuk informasi selengkapnya tentang autentikasi pada klaster Aurora PostgreSQL, lihat Keamanan dengan Amazon Aurora PostgreSQL.

Anda mungkin diminta untuk kredensial setiap kali Anda terhubung ke Babelfish. Setiap pengguna yang bermigrasi ke atau dibuat di Aurora PostgreSQL dapat menggunakan kredensial yang sama pada port SQL Server dan port PostgreSQL. Babelfish tidak menerapkan kebijakan kata sandi, namun sebaiknya Anda melakukan hal berikut:

- Memerlukan kata sandi kompleks yang panjangnya setidaknya delapan (8) karakter.
- Menerapkan kebijakan kedaluwarsa kata sandi.

Untuk meninjau daftar lengkap pengguna basis data, gunakan perintah`SELECT * FROM pg_user;`.

Autentikasi Kerberos dengan Babelfish

Babelfish for Aurora PostgreSQL versi 15.2 mendukung autentikasi ke klaster DB Anda menggunakan Kerberos. Metode ini memungkinkan Anda menggunakan Microsoft Windows Authentication untuk mengautentikasi pengguna saat mereka terhubung ke basis data Babelfish Anda. Untuk melakukannya, Anda harus mengonfigurasi klaster DB Anda agar dapat digunakan AWS Directory Service for Microsoft Active Directory untuk autentikasi Kerberos. Untuk informasi lebih lanjut, lihat Apa itu AWS Directory Service? di Panduan Administrasi AWS Directory Service.

Mengatur autentikasi Kerberos

Klaster DB Babelfish for Aurora PostgreSQL dapat terhubung menggunakan dua port yang berbeda, tetapi pengaturan autentikasi Kerberos adalah proses satu kali. Oleh karena itu, Anda harus terlebih dahulu mengatur autentikasi Kerberos untuk klaster DB Anda. Untuk informasi selengkapnya, lihat Mengatur autentikasi Kerberos. Setelah menyelesaikan penyiapan, pastikan Anda dapat terhubung dengan klien PostgreSQL menggunakan Kerberos. Untuk informasi selengkapnya, lihat Mengatur autentikasi Kerberos.

Login dan penyediaan pengguna di Babelfish

Login Windows yang dibuat dari port Aliran Data Tabular (TDS) dapat digunakan baik dengan port TDS atau port PostgreSQL. Pertama, login yang dapat menggunakan Kerberos untuk autentikasi harus disediakan dari port TDS sebelum digunakan oleh pengguna T-SQL dan aplikasi untuk terhubung ke basis data Babelfish. Saat membuat login Windows, administrator dapat memberikan login menggunakan nama domain DNS atau nama domain NetBIOS. Biasanya, domain NetBIOS adalah subdomain dari nama domain DNS. Misalnya, jika nama domain DNS adalah `CORP.EXAMPLE.COM`, maka domain NetBIOS bisa berupa `CORP`. Jika format nama domain NetBIOS disediakan untuk login, pemetaan harus ada ke nama domain DNS.

Mengelola nama domain NetBIOS ke pemetaan nama domain DNS

Untuk mengelola pemetaan antara nama domain NetBIOS dan nama domain DNS, Babelfish menyediakan prosedur tersimpan sistem untuk menambah, menghapus, dan memotong pemetaan. Hanya pengguna dengan peran `sysadmin` yang dapat menjalankan prosedur ini.

Untuk membuat pemetaan antara nama domain NetBIOS dan DNS, gunakan prosedur penyimpanan sistem yang disediakan Babelfish `babelfish_add_domain_mapping_entry`. Kedua argumen harus memiliki nilai yang valid dan tidak NULL.

Example

```
EXEC babelfish_add_domain_mapping_entry 'netbios_domain_name',
 'fully_qualified_domain_name'
```

Contoh berikut menunjukkan cara membuat pemetaan antara domain NetBIOS CORP dan nama domain DNS CORP.EXAMPLE.COM.

Example

```
EXEC babelfish_add_domain_mapping_entry 'corp', 'corp.example.com'
```

Untuk menghapus entri pemetaan yang ada, gunakan prosedur tersimpan sistem babelfish_remove_domain_mapping_entry.

Example

```
EXEC babelfish_remove_domain_mapping_entry 'netbios_domain_name'
```

Contoh berikut menunjukkan cara menghapus pemetaan untuk nama NetBIOS CORP.

Example

```
EXEC babelfish_remove_domain_mapping_entry 'corp'
```

Untuk menghapus entri pemetaan yang ada, gunakan prosedur tersimpan sistem babelfish_truncate_domain_mapping_table:

Example

```
EXEC babelfish_truncate_domain_mapping_table
```

Untuk melihat semua pemetaan antara nama domain NetBIOS dan DNS, gunakan kueri berikut.

Example

```
SELECT netbios_domain_name, fq_domain_name FROM babelfish_domain_mapping;
```

Mengatur Login

Membuat login

Hubungkan ke DB melalui titik akhir TDS menggunakan login yang memiliki izin yang benar. Jika tidak ada pengguna basis data yang dibuat untuk login, maka login dipetakan ke pengguna tamu. Jika pengguna tamu tidak diaktifkan, maka upaya login gagal.

Buat login Windows menggunakan kueri berikut. Opsi `FROM WINDOWS` ini memungkinkan autentikasi menggunakan Active Directory.

```
CREATE LOGIN login_name FROM WINDOWS [WITH DEFAULT_DATABASE=database]
```

Example

Contoh berikut menunjukkan cara membuat login untuk pengguna Active Directory [corp\test1] dengan basis data default db1.

```
CREATE LOGIN [corp\test1] FROM WINDOWS WITH DEFAULT_DATABASE=db1
```

Contoh ini mengasumsikan bahwa ada pemetaan antara domain NetBIOS CORP dan nama domain DNS CORP.EXAMPLE.COM. Jika tidak ada pemetaan, Anda harus memberikan nama domain DNS [CORP.EXAMPLE.COM\test1].

> **Note**
>
> Login berdasarkan pengguna Active Directory, dibatasi nama kurang dari 21 karakter.

Menjatuhkan login

Untuk menjatuhkan login, gunakan sintaks yang sama seperti untuk login lainnya, seperti yang ditunjukkan pada contoh berikut:

```
DROP LOGIN [DNS domain name\login]
```

Mengubah login

Untuk mengubah login, gunakan sintaks yang sama seperti untuk login lainnya, seperti yang ditunjukkan pada contoh berikut:

```
ALTER LOGIN [DNS domain name\login] { ENABLE|DISABLE|WITH DEFAULT_DATABASE=[master] }
```

Perintah ALTER LOGIN mendukung opsi terbatas untuk login Windows, termasuk yang berikut ini:

- DISABLE – Untuk menonaktifkan login. Anda tidak dapat menggunakan login yang dinonaktifkan untuk autentikasi.
- ENABLE – Untuk mengaktifkan login yang dinonaktifkan.
- DEFAULT_DATABASE – Untuk mengubah basis data login default.

> **Note**
>
> Semua manajemen kata sandi dilakukan melalui AWS Directory Service, sehingga perintah ALTER LOGIN tidak mengizinkan administrator basis data untuk mengubah atau mengatur kata sandi untuk login Windows.

Menghubungkan ke Babelfish for Aurora PostgreSQL dengan autentikasi Kerberos

Biasanya, pengguna basis data yang mengautentikasi menggunakan Kerberos melakukannya dari mesin klien mereka. Mesin ini adalah anggota domain Active Directory. Mereka menggunakan Autentikasi Windows dari aplikasi klien mereka untuk mengakses server Babelfish for Aurora PostgreSQL pada port TDS.

Menghubungkan ke Babelfish for Aurora PostgreSQL pada port PostgreSQL dengan autentikasi Kerberos

Anda dapat menggunakan login yang dibuat dari port TDS dengan port TDS atau port PostgreSQL. Namun, PostgreSQL menggunakan perbandingan peka huruf besar/kecil secara default untuk nama pengguna. Agar Aurora PostgreSQL menafsirkan nama pengguna Kerberos sebagai peka huruf besar/kecil, Anda harus menetapkan parameter `krb_caseins_users` sebagai `true` dalam grup parameter klaster Babelfish kustom. Parameter ini diatur ke secara `false` default. Untuk informasi selengkapnya, lihat Mengonfigurasi nama pengguna yang tidak peka huruf besar/kecil. Selain itu, Anda harus menentukan nama pengguna login dalam format <login@nama domain DNS> dari aplikasi klien PostgreSQL. Anda tidak dapat menggunakan format <domain name DNS\login>.

Kesalahan yang sering terjadi

Anda dapat mengonfigurasi hubungan kepercayaan forest antara Microsoft Active Directory on-premise dan AWS Managed Microsoft AD. Untuk informasi selengkapnya, lihat Membuat hubungan kepercayaan. Kemudian, Anda harus terhubung menggunakan titik akhir khusus domain khusus

alih-alih menggunakan domain Amazon `rds.amazonaws.com` di titik akhir host. Jika Anda tidak menggunakan titik akhir khusus domain yang benar, Anda mungkin mengalami kesalahan berikut:

```
Error: “Authentication method "NTLMSSP" not supported (Microsoft SQL Server, Error:
 514)"
```

Kesalahan ini terjadi ketika klien TDS tidak dapat menyimpan tiket layanan untuk URL titik akhir yang disediakan. Untuk informasi selengkapnya, lihat Mengatur autentikasi Kerberos.

Menghubungkan ke klaster DB Babelfish

Untuk terhubung ke Babelfish, Anda terhubung ke titik akhir klaster Aurora PostgreSQL yang menjalankan Babelfish. Klien Anda dapat menggunakan salah satu driver klien berikut yang sesuai dengan TDS versi 7.1 hingga 7.4:

- Open Database Connectivity (ODBC)
- Driver DB OLE/MSOLEDBSQL
- Java Database Connectivity (JDBC) versi 8.2.2 (mssql-jdbc-8.2.2) dan lebih baru
- Penyedia Data SqlClient Microsoft untuk SQL Server
- Penyedia Data .NET untuk SQL Server
- SQL Server Native Client 11.0 (tidak digunakan lagi)
- OLE DB Provider/SQLOLEDB (tidak digunakan lagi)

Dengan Babelfish, Anda menjalankan hal-hal berikut:

- Alat, aplikasi, dan sintaks SQL Server pada port TDS, secara default port 1433.
- Alat, aplikasi, dan sintaks PostgreSQL pada port PostgreSQL, secara default port 5432.

Untuk mempelajari selengkapnya tentang menghubungkan ke Aurora PostgreSQL secara umum, lihat Menghubungkan ke klaster DB Amazon Aurora PostgreSQL..

Topik

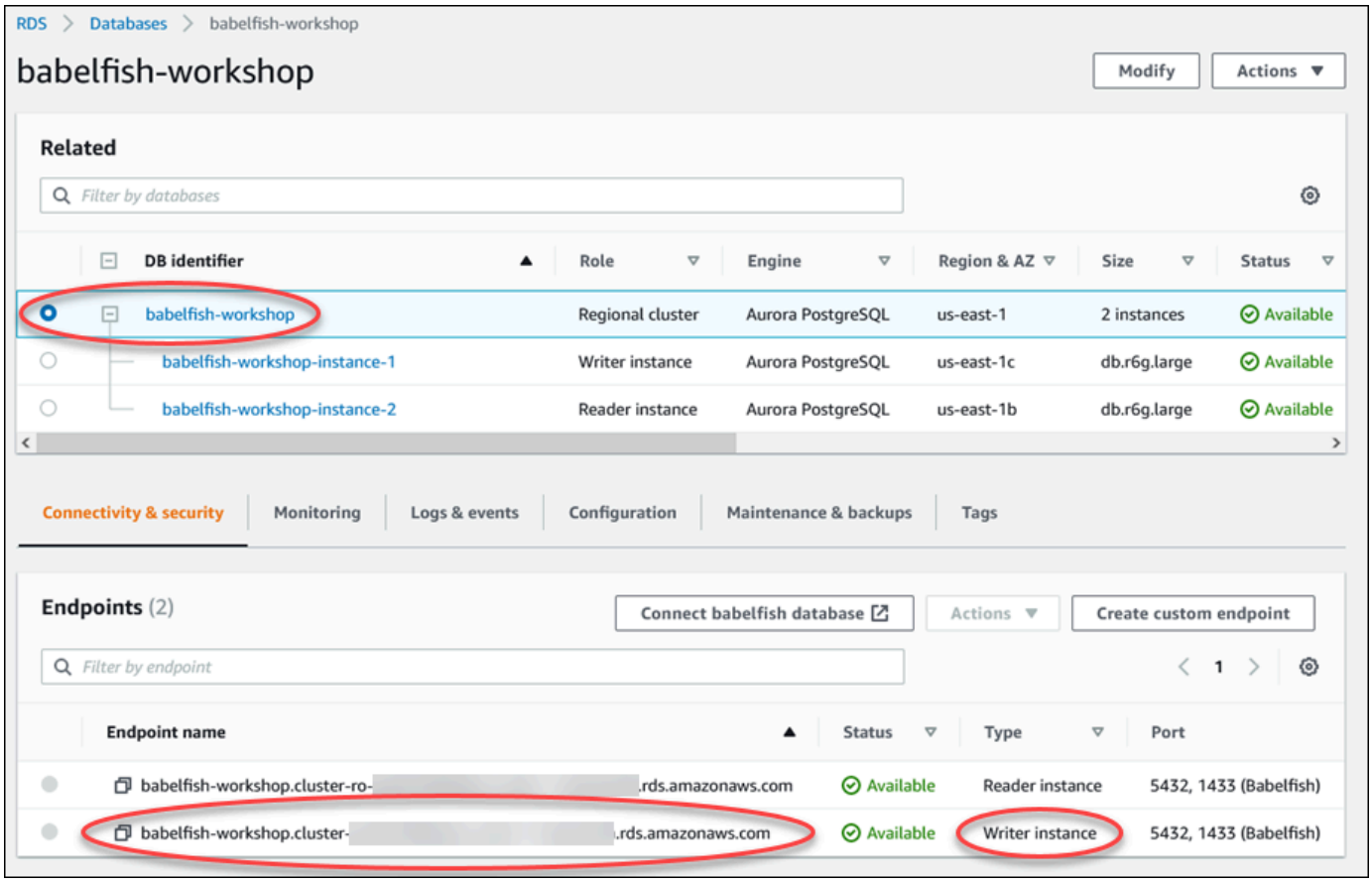
- Menemukan titik akhir dan nomor port penulis
- Membuat koneksi klien C# atau JDBC ke Babelfish
- Menggunakan klien SQL Server untuk terhubung ke klaster DB Anda
- Menggunakan klien PostgreSQL untuk terhubung ke klaster DB Anda

Menemukan titik akhir dan nomor port penulis

Untuk terhubung ke klaster DB Babelfish, Anda menggunakan titik akhir yang terkait dengan instans penulis (utama) klaster DB. Instans harus memiliki status Tersedia. Diperlukan waktu hingga 20 menit agar instans tersedia setelah membuat klaster DB Babelfish for Aurora PostgreSQL.

Menemukan titik akhir basis data Anda

1. Buka konsol untuk Babelfish.
2. Pilih Basis data dari panel navigasi.
3. Pilih klaster DB Babelfish for Aurora PostgreSQL Anda dari yang terdaftar untuk melihat detailnya.
4. Pada tab Konektivitas & keamanan, perhatikan nilai Titik Akhir klaster yang tersedia. Gunakan titik akhir klaster untuk instans penulis dalam string koneksi Anda untuk aplikasi apa pun yang melakukan operasi tulis atau baca pada basis data.



Untuk informasi selengkapnya tentang detail klaster DB Aurora, lihat Membuat klaster DB Amazon Aurora.

Membuat koneksi klien C# atau JDBC ke Babelfish

Berikut ini Anda dapat menemukan beberapa contoh menggunakan kelas C# dan JDBC untuk terhubung ke Babelfish for Aurora PostgreSQL.

Example : Menggunakan kode C # untuk terhubung ke klaster DB

```
string dataSource = 'babelfishServer_11_24';

//Create connection
connectionString = @"Data Source=" + dataSource
    +";Initial Catalog=your-DB-name"
    +";User ID=user-id;Password=password";

SqlConnection cnn = new SqlConnection(connectionString);
cnn.Open();
```

Example : Menggunakan kelas dan antarmuka API JDBC generik untuk terhubung ke klaster DB

```
String dbServer =
   "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";
String connectionUrl = "jdbc:sqlserver://" + dbServer + ":1433;" +
    "databaseName=your-DB-name;user=user-id;password=password";

// Load the SQL Server JDBC driver and establish the connection.
System.out.print("Connecting Babelfish Server ... ");
Connection cnn = DriverManager.getConnection(connectionUrl);
```

Example : Menggunakan kelas dan antarmuka JDBC khusus SQL Server untuk terhubung ke klaster DB

```
// Create datasource.
SQLServerDataSource ds = new SQLServerDataSource();
ds.setUser("user-id");
ds.setPassword("password");
String babelfishServer =
   "database-babelfish.cluster-123abc456def.us-east-1-rds.amazonaws.com";

ds.setServerName(babelfishServer);
ds.setPortNumber(1433);
ds.setDatabaseName("your-DB-name");
```

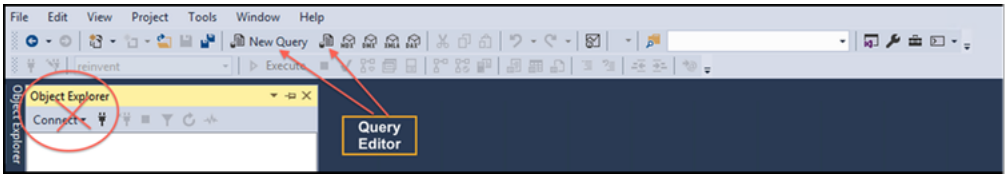
```
Connection con = ds.getConnection();
```

Menggunakan klien SQL Server untuk terhubung ke klaster DB Anda

Anda dapat menggunakan klien SQL Server untuk terhubung dengan Babelfish pada port TDS. Pada Babelfish 2.1.0 dan rilis yang lebih tinggi, Anda dapat menggunakan SSMS Object Explorer atau SSMS Query Editor untuk terhubung ke klaster Babelfish Anda.

Batasan

- Di Babelfish 2.1.0 dan versi yang lebih lama, menggunakan `PARSE` untuk memeriksa sintaks SQL tidak berfungsi sebagaimana mestinya. Daripada memeriksa sintaks tanpa menjalankan kueri, perintah `PARSE` menjalankan kueri tetapi tidak menampilkan hasil apa pun. Menggunakan <Ctrl><F5>kombinasi tombol SMSS untuk memeriksa sintaks memiliki perilaku anomali yang sama, yaitu, Babelfish secara tak terduga menjalankan kueri tanpa memberikan output apa pun.
- Babelfish tidak mendukung MARS (Multiple Active Result Sets). Pastikan bahwa setiap aplikasi klien yang Anda gunakan untuk terhubung ke Babelfish tidak diatur untuk menggunakan MARS.
- Untuk Babelfish 1.3.0 dan versi yang lebih lama, hanya Editor Kueri yang didukung untuk SSMS. Untuk menggunakan SSMS dengan Babelfish, pastikan untuk membuka dialog koneksi Editor Kueri di SSMS, dan bukan Object Explorer. Jika dialog Object Explorer terbuka, batalkan dialog dan buka kembali Editor Kueri. Pada gambar berikut, Anda dapat menemukan opsi menu untuk dipilih saat menghubungkan ke Babelfish 1.3.0 atau versi yang lebih lama.



Untuk informasi selengkapnya tentang interoperabilitas dan perbedaan perilaku antara SQL Server dan Babelfish, lihat Perbedaan antara Babelfish for Aurora PostgreSQL dan SQL Server.

Menggunakan sqlcmd untuk terhubung ke klaster DB

Anda dapat terhubung ke dan berinteraksi dengan cluster Aurora PostgreSQL DB yang mendukung Babelfish dengan hanya menggunakan versi 19.1 dan klien baris perintah SQL Server sebelumnya. `sqlcmd` SSMS versi 19.2 tidak didukung untuk terhubung ke cluster Babelfish. Gunakan perintah berikut ini untuk terhubung.

```
sqlcmd -S endpoint,port -U login-id -P password -d your-DB-name
```

Opsinya adalah sebagai berikut:

- `-S` adalah titik akhir dan port TDS (opsional) dari klaster DB.
- `-U` adalah nama login pengguna.
- `-P` adalah kata sandi yang terkait dengan pengguna.
- `-d` adalah nama basis data Babelfish Anda.

Setelah terhubung, Anda dapat menggunakan banyak perintah yang sama yang Anda gunakan dengan SQL Server. Sebagai contoh, lihat Mendapatkan informasi dari katalog sistem Babelfish.

Menggunakan SSMS untuk terhubung ke klaster DB

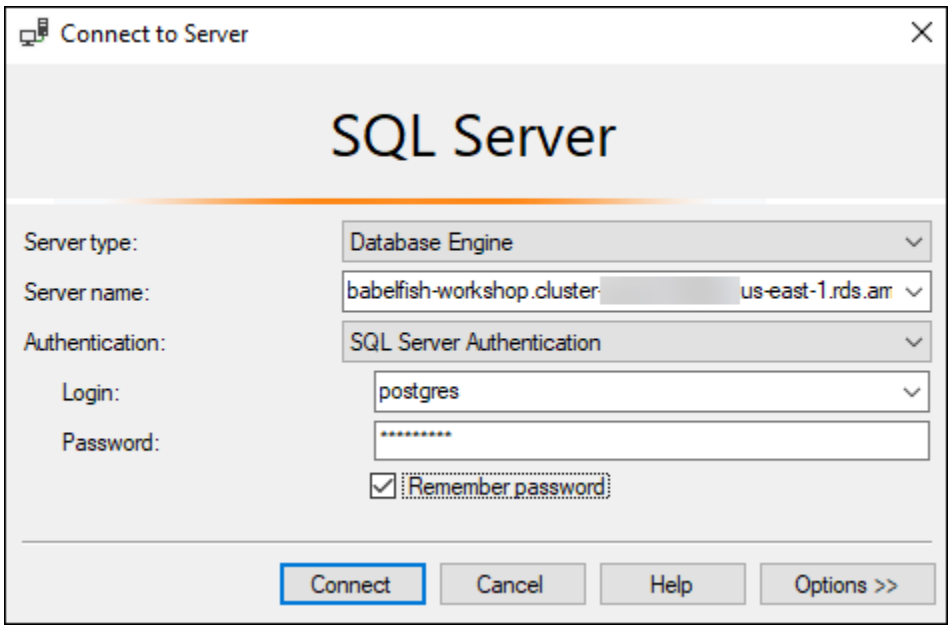
Anda dapat terhubung ke klaster DB Aurora PostgreSQL yang menjalankan Babelfish menggunakan Microsoft SQL Server Management Studio (SSMS). SSMS mencakup berbagai alat, termasuk Wizard Impor dan Ekspor SQL Server dibahas di Memigrasi basis data SQL Server ke Babelfish for Aurora PostgreSQL. Untuk informasi selengkapnya tentang SSMS, lihat Unduh SQL Server Management Studio (SSMS) dalam dokumentasi Microsoft.

Untuk terhubung ke basis data Babelfish Anda dengan SSMS

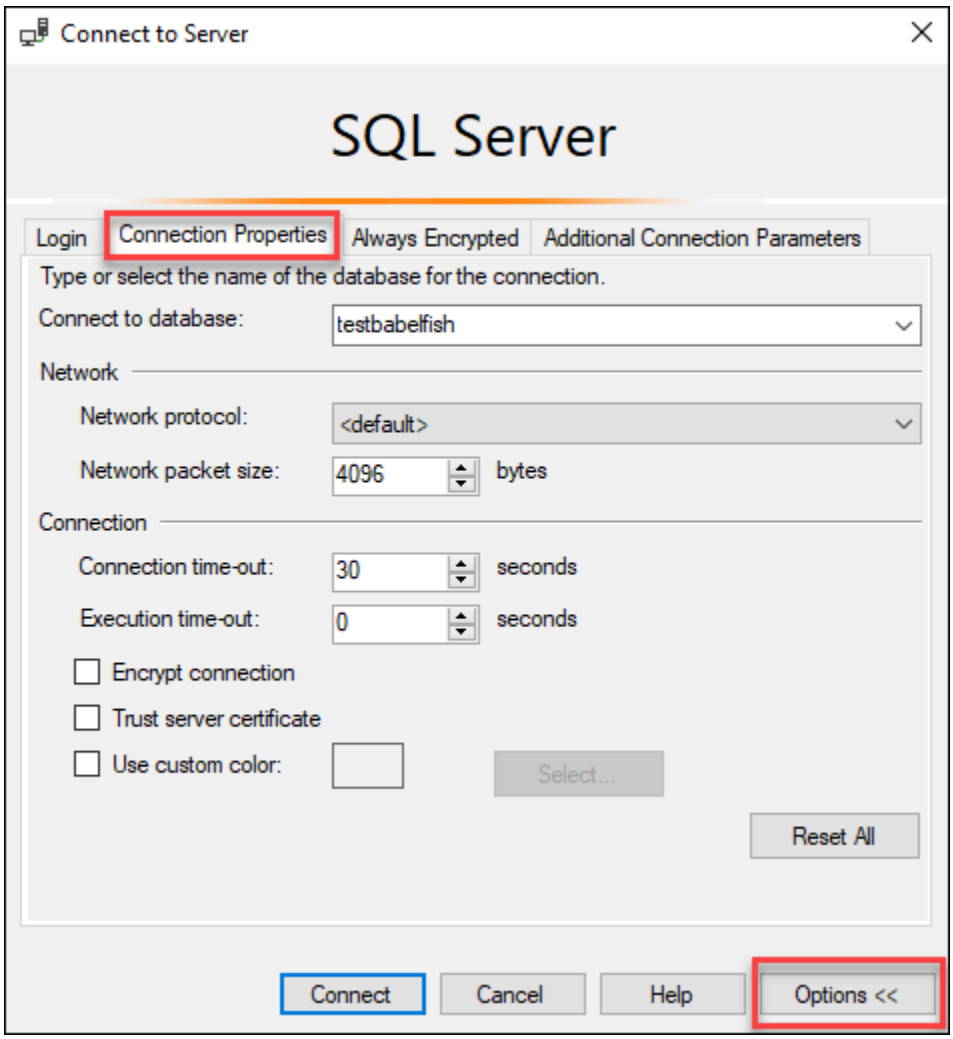
1. Mulai SSMS.
2. Buka kotak dialog Hubungkan ke Server. Untuk melanjutkan koneksi, lakukan salah satu hal berikut:
   - Pilih Kueri Baru.
   - Jika Editor Kueri terbuka, pilih Kueri, Koneksi, Hubungkan.
3. Berikan informasi berikut untuk basis data Anda:
   a. Untuk Jenis server, pilih Mesin Basis Data.
   b. Untuk Nama server, masukkan nama DNS. Misalnya, nama server Anda akan terlihat seperti berikut.

      ```
      cluster-name.cluster-555555555555.aws-region.rds.amazonaws.com,1433
      ```

   c. Untuk Autentikasi, pilih Autentikasi SQL Server.
   d. Untuk Login, masukkan nama pengguna yang dipilih saat Anda membuat basis data.
   e. Untuk Kata Sandi, masukkan kata sandi yang dipilih saat Anda membuat basis data.



4. (Opsional) Pilih Opsi, lalu pilih tab Properti Koneksi.



5. (Opsional) Untuk Hubungkan ke basis data, tentukan nama basis data SQL Server yang dimigrasi untuk disambungkan, dan pilih Hubungkan.

   Jika muncul pesan yang menunjukkan bahwa SSMS tidak dapat menerapkan string koneksi, pilih OK.

   Jika Anda mengalami kesulitan menghubungkan ke Babelfish, lihat Kegagalan koneksi.

   Untuk informasi selengkapnya tentang masalah koneksi SQL Server, lihat Memecahkan masalah koneksi ke instans DB SQL Server Anda di Panduan Pengguna RDS Amazon.

Menggunakan klien PostgreSQL untuk terhubung ke klaster DB Anda

Anda dapat menggunakan klien PostgreSQL untuk terhubung ke Babelfish pada port PostgreSQL.

Menggunakan psql untuk terhubung ke klaster DB

Anda dapat mengunduh klien PostgreSQL dari situs web PostgreSQL. Ikuti instruksi khusus untuk versi sistem operasi Anda untuk menginstal psql.

Anda dapat menanyakan klaster DB PostgreSQL Aurora yang mendukung Babelfish dengan klien baris perintah `psql`. Saat menghubungkan, gunakan port PostgreSQL (secara default, port 5432). Biasanya, Anda tidak perlu menentukan nomor port kecuali Anda mengubahnya dari default. Gunakan perintah berikut untuk terhubung ke Babelfish dari klien `psql`:

```
psql -h bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com
-p 5432 -U postgres -d babelfish_db
```

Parameter tersebut adalah sebagai berikut:

- `-h` – Nama host klaster DB (titik akhir klaster) yang ingin Anda akses.
- `-p` – Nomor port PostgreSQL yang digunakan untuk terhubung ke instans DB Anda.
- `-d` – Basis data yang ingin Anda hubungkan. Default-nya adalah `babelfish_db`.
- `-U` – Akun pengguna basis data yang ingin Anda akses. (Contoh menunjukkan nama pengguna utama default.)

Ketika Anda menjalankan perintah SQL pada klien psql, Anda mengakhiri perintah dengan titik koma. Misalnya, perintah SQL berikut meminta tampilan sistem pg_tables untuk mengembalikan informasi tentang setiap tabel di basis data.

```
SELECT * FROM pg_tables;
```

Klien psql juga memiliki satu set metacommand bawaan. Metacommand adalah pintasan yang menyesuaikan pemformatan atau menyediakan pintasan yang mengembalikan meta-data dalam format yang mudah digunakan. Misalnya, metacommand berikut mengembalikan informasi yang mirip dengan perintah SQL sebelumnya:

```
\d
```

Metacommand tidak perlu diakhiri dengan titik koma (;).

Untuk keluar dari klien psql, masukkan \q.

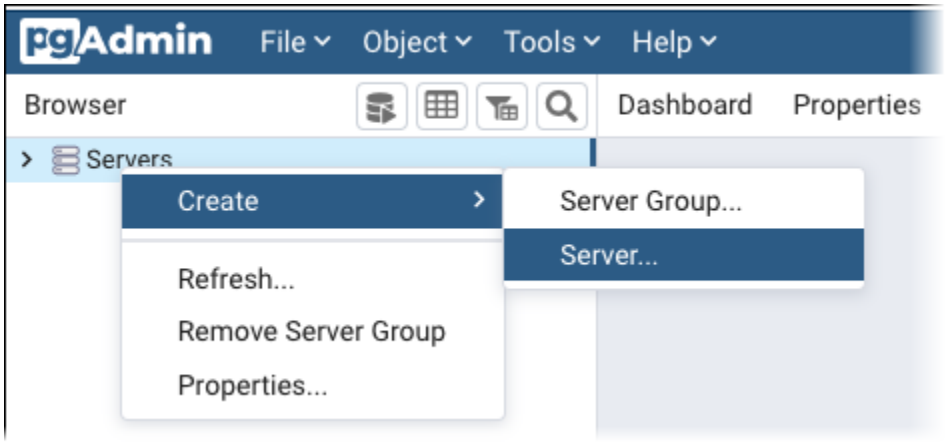
Untuk informasi selengkapnya tentang penggunaan klien psql untuk menanyakan klaster Aurora PostgreSQL, lihat dokumentasi PostgreSQL.

Menggunakan pgAdmin untuk terhubung ke klaster DB

Anda dapat menggunakan klien pgAdmin untuk mengakses data Anda dalam dialek PostgreSQL asli.

Untuk terhubung ke klaster dengan klien pgAdmin

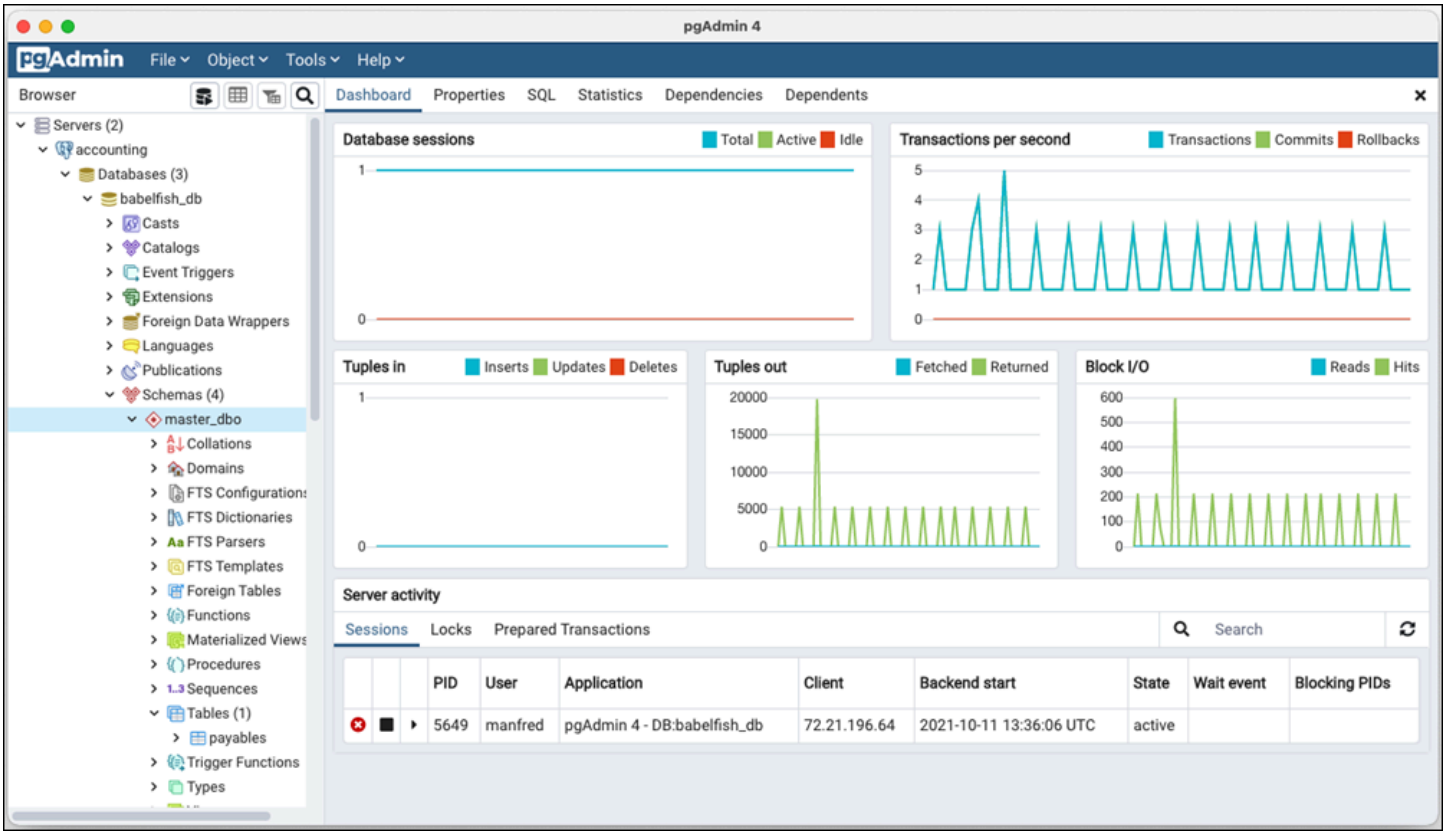
1. Unduh dan instal klien pgadmin dari situs web pgAdmin.
2. Buka klien dan autentikasi dengan pgAdmin.
3. Buka menu konteks (klik kanan) untuk Server, lalu pilih Buat, Server.



4. Masukkan informasi di kotak dialog Buat - Server.

   Pada tab Koneksi, tambahkan alamat klaster Aurora PostgreSQL untuk Host dan nomor port PostgreSQL (secara default, 5432) untuk Port. Berikan detail autentikasi, dan pilih Simpan.

Setelah terhubung, Anda dapat menggunakan fungsionalitas pgAdmin untuk memantau dan mengelola klaster Aurora PostgreSQL Anda di port PostgreSQL.



Untuk mempelajari lebih lanjut, lihat halaman web pgAdmin.

Menggunakan Babelfish

Pada bagian berikut, Anda dapat menemukan informasi penggunaan Babelfish, termasuk beberapa perbedaan antara cara menggunakan Babelfish dan SQL Server serta antara basis data Babelfish dan PostgreSQL.

Topik

- Mendapatkan informasi dari katalog sistem Babelfish
- Perbedaan antara Babelfish for Aurora PostgreSQL dan SQL Server
- Menggunakan fitur Babelfish dengan implementasi terbatas
- Meningkatkan performa kueri Babelfish
- Menggunakan ekstensi Aurora PostgreSQL dengan Babelfish
- Babelfish mendukung server tertaut
- Pencarian Teks Lengkap di Babelfish

Mendapatkan informasi dari katalog sistem Babelfish

Anda dapat memperoleh informasi tentang objek basis data yang disimpan di klaster Babelfish Anda dengan menjalankan kueri banyak tampilan sistem yang sama seperti yang digunakan dalam SQL Server. Setiap rilis baru Babelfish menambahkan dukungan untuk lebih banyak tampilan sistem. Untuk daftar tampilan yang tersedia saat ini, lihat tabel SQL Server system catalog views.

Tampilan sistem ini memberikan informasi dari katalog sistem (`sys.schemas`). Dalam kasus Babelfish, tampilan ini berisi skema sistem SQL Server dan PostgreSQL. Untuk menjalankan kueri seputar informasi katalog sistem pada Babelfish, Anda dapat menggunakan port TDS atau port PostgreSQL, seperti yang ditunjukkan pada contoh berikut.

- Jalankan kueri pada port T-SQL menggunakan **`sqlcmd`** atau klien SQL Server lainnya.

```
1> SELECT * FROM sys.schemas
2> GO
```

  Kueri ini menampilkan skema sistem SQL Server dan Aurora PostgreSQL, seperti yang ditunjukkan pada yang berikut ini.

```
name
-----------------------------------------------------------
```

```
demographic_dbo
public
sys
master_dbo
tempdb_dbo
...
```

- Jalankan kueri pada port PostgreSQL menggunakan **`psql`** atau **`pgAdmin`**. Contoh ini menggunakan metacommand (`\dn`) skema daftar `psql`:

  ```
  babelfish_db=> \dn
  ```

  Kueri menampilkan set hasil yang sama seperti yang ditampilkan oleh `sqlcmd` pada port T-SQL.

  ```
          List of schemas
               Name
  ------------------------------

   demographic_dbo

   public
   sys
   master_dbo
   tempdb_dbo
  ...
  ```

Katalog sistem SQL Server yang tersedia di Babelfish

Dalam tabel berikut, Anda dapat menemukan tampilan SQL Server yang saat ini diimplementasikan di Babelfish. Untuk informasi selengkapnya tentang katalog sistem di SQL Server, lihat Tampilan Katalog Sistem (Transact-SQL) di dokumentasi Microsoft.

Nama tampilan	Keterangan atau batasan Babelfish (jika ada)
`sys.all_columns`	Semua kolom di semua tabel dan tampilan
`sys.all_objects`	Semua objek di semua skema
`sys.all_sql_modules`	Penyatuan `sys.sql_modules` dan `sys.system_sql_modules`

Nama tampilan	Keterangan atau batasan Babelfish (jika ada)
`sys.all_views`	Semua tampilan di semua skema
`sys.columns`	Semua kolom dalam tabel dan tampilan yang ditentukan pengguna
`sys.configurations`	Dukungan Babelfish terbatas pada satu konfigurasi hanya baca.
`sys.data_spaces`	Berisi baris untuk setiap ruang data. Ini bisa berupa filegroup, skema partisi, atau filegroup data FILESTREAM.
`sys.database_files`	Tampilan per basis data yang berisi satu baris untuk setiap file basis data seperti yang disimpan dalam basis data itu sendiri.
`sys.database_mirroring`	Untuk info selengkapnya, lihat sys.database_mirroring di dokumentasi Microsoft Transact-SQL.
`sys.database_principals`	Untuk info selengkapnya, lihat sys.database_principals di dokumentasi Microsoft Transact-SQL.
`sys.database_role_members`	Untuk info selengkapnya, lihat sys.database_role_members di dokumentasi Microsoft Transact-SQL.
`sys.databases`	Semua basis data di semua skema
`sys.dm_exec_connections`	Untuk info selengkapnya, lihat sys.dm_exec_connections di dokumentasi Microsoft Transact-SQL.
`sys.dm_exec_sessions`	Untuk info selengkapnya, lihat sys.dm_exec_sessions di dokumentasi Microsoft Transact-SQL.

Nama tampilan	Keterangan atau batasan Babelfish (jika ada)
`sys.dm_hadr_database_replica_states`	Untuk info selengkapnya, lihat sys.dm_hadr_database_replica_states di dokumentasi Microsoft Transact-SQL.
`sys.dm_os_host_info`	Untuk info selengkapnya, lihat sys.dm_os_host_info di dokumentasi Microsoft Transact-SQL.
`sys.endpoints`	Untuk info selengkapnya, lihat sys.endpoints di dokumentasi Microsoft Transact-SQL.
`sys.indexes`	Untuk info selengkapnya, lihat sys.indexes di dokumentasi Microsoft Transact-SQL.
`sys.languages`	Untuk info selengkapnya, lihat sys.languages di dokumentasi Microsoft Transact-SQL.
`sys.schemas`	Semua skema
`sys.server_principals`	Semua kredensial masuk dan peran
`sys.sql_modules`	Untuk info selengkapnya, lihat sys.sql_modules di dokumentasi Microsoft Transact-SQL.
`sys.sysconfigures`	Dukungan Babelfish terbatas pada satu konfigurasi hanya baca.
`sys.syscurconfigs`	Dukungan Babelfish terbatas pada satu konfigurasi hanya baca.
`sys.sysprocesses`	Untuk info selengkapnya, lihat sys.sysprocesses di dokumentasi Microsoft Transact-SQL.
`sys.system_sql_modules`	Untuk info selengkapnya, lihat sys.system_sql_modules di dokumentasi Microsoft Transact-SQL.

Nama tampilan	Keterangan atau batasan Babelfish (jika ada)
`sys.table_types`	Untuk info selengkapnya, lihat sys.table_types di dokumentasi Microsoft Transact-SQL.
`sys.tables`	Semua tabel dalam skema
`sys.xml_schema_collections`	Untuk info selengkapnya, lihat sys.xml_schema_collections di dokumentasi Microsoft Transact-SQL.

PostgreSQL mengimplementasikan katalog sistem yang mirip dengan tampilan katalog objek SQL Server. Untuk daftar lengkap katalog sistem, lihat Katalog Sistem di dokumentasi PostgreSQL.

Ekspor DDL didukung oleh Babelfish

Dari versi Babelfish 2.4.0 dan 3.1.0, Babelfish mendukung ekspor DDL menggunakan berbagai alat. Misalnya, Anda dapat menggunakan fungsi ini dari SQL Server Management Studio (SSMS) untuk menghasilkan skrip definisi data untuk berbagai objek dalam basis data Babelfish for Aurora PostgreSQL. Anda kemudian dapat menggunakan perintah DDL yang dihasilkan dalam skrip ini untuk membuat objek yang sama di Babelfish for Aurora PostgreSQL lain atau basis data SQL Server.

Babelfish mendukung ekspor DDL untuk objek berikut dalam versi yang ditentukan.

Daftar objek	2.4.0	3.1.0
Tabel pengguna	Ya	Ya
Kunci primer	Ya	Ya
Kunci asing	Ya	Ya
Kendala unik	Ya	Ya
Indeks	Ya	Ya
Kendala pemeriksaan	Ya	Ya

Daftar objek	2.4.0	3.1.0
Tampilan	Ya	Ya
Prosedur tersimpan	Ya	Ya
Fungsi yang ditetapkan pengguna	Ya	Ya
Fungsi bernilai tabel	Ya	Ya
Pemicu	Ya	Ya
Tipe Data yang Ditetapkan Pengguna	Tidak	Tidak
Tipe Tabel yang Ditetapkan Pengguna	Tidak	Tidak
Pengguna	Tidak	Tidak
Kredensial Masuk	Tidak	Tidak
Urutan	Tidak	Tidak
Peran	Tidak	Tidak

Batasan dengan DDL yang diekspor

- Gunakan escape hatch sebelum membuat ulang objek dengan DDL yang diekspor – Babelfish tidak mendukung semua perintah dalam skrip DDL yang diekspor. Gunakan escape hatch untuk menghindari kesalahan yang disebabkan saat membuat ulang objek dari perintah DDL di Babelfish. Untuk informasi selengkapnya tentang escape hatch, lihat Mengelola penanganan kesalahan Babelfish dengan escape hatch
- Objek yang berisi kendala CHECK dengan klausa COLLATE eksplisit – Skrip dengan objek ini yang dihasilkan dari basis data SQL Server memiliki kolasi yang berbeda tetapi setara seperti dalam basis data Babelfish. Misalnya, beberapa kolasi, seperti sql_latin1_general_cp1_cs_as, sql_latin1_general_cp1251_cs_as, dan latin1_general_cs_as dihasilkan seperti latin1_general_cs_as, yang merupakan kolasi Windows terdekat.

Perbedaan antara Babelfish for Aurora PostgreSQL dan SQL Server

Babelfish adalah fitur Aurora PostgreSQL yang berkembang, dengan fungsionalitas baru yang ditambahkan di setiap rilis sejak penawaran awal di Aurora PostgreSQL 13.4. Fitur ini dirancang untuk menyediakan semantik T-SQL di atas PostgreSQL melalui dialek T-SQL menggunakan port TDS. Setiap versi baru Babelfish menambahkan fitur dan fungsi yang lebih selaras dengan fungsionalitas dan perilaku T-SQL, seperti yang ditunjukkan pada tabel Fungsionalitas yang didukung di Babelfish berdasarkan versi. Untuk hasil terbaik saat bekerja dengan Babelfish, sebaiknya pahami perbedaan yang ada saat ini antara T-SQL yang didukung oleh SQL Server dan Babelfish untuk versi terbaru. Untuk mempelajari selengkapnya, lihat Perbedaan T-SQL di Babelfish.

Selain perbedaan antara T-SQL yang didukung oleh Babelfish dan SQL Server, Anda mungkin juga perlu mempertimbangkan masalah interoperabilitas antara Babelfish dan PostgreSQL dalam konteks klaster DB Aurora PostgreSQL. Seperti disebutkan sebelumnya, Babelfish mendukung semantik T-SQL di atas PostgreSQL melalui dialek T-SQL menggunakan port TDS. Pada saat yang sama, basis data Babelfish juga dapat diakses melalui port PostgreSQL standar dengan pernyataan SQL PostgreSQL. Jika Anda mempertimbangkan untuk menggunakan fungsionalitas PostgreSQL dan Babelfish dalam deployment produksi, Anda perlu mengetahui potensi masalah interoperabilitas antara nama skema, pengidentifikasi, izin, semantik transaksional, beberapa set hasil, pengumpulan default, dan sebagainya. Secara sederhana, ketika pernyataan PostgreSQL atau akses PostgreSQL terjadi dalam konteks Babelfish, interferensi antara PostgreSQL dan Babelfish dapat terjadi dan berpotensi mempengaruhi sintaks, semantik, dan kompatibilitas ketika versi baru Babelfish dirilis. Untuk informasi lengkap dan panduan tentang semua pertimbangan, lihat Panduan tentang Interoperabilitas Babelfish dalam dokumentasi Babelfish for PostgreSQL.

> **Note**
>
> Sebelum menggunakan fungsionalitas asli PostgreSQL dan fungsionalitas Babelfish dalam konteks aplikasi yang sama, kami sangat menyarankan Anda mempertimbangkan masalah yang dibahas dalam Panduan Interoperabilitas Babelfish dalam dokumentasi Babelfish for PostgreSQL. Masalah interoperabilitas ini (Aurora PostgreSQL dan Babelfish) hanya relevan jika Anda berencana untuk menggunakan instans basis data PostgreSQL dalam konteks aplikasi yang sama dengan Babelfish.

Topik

- Babelfish membuang dan memulihkan

- Perbedaan T-SQL di Babelfish
- Tingkat Isolasi Transaksi di Babelfish

Babelfish membuang dan memulihkan

Dimulai dengan versi 4.0.0 dan 3.4.0, pengguna Babelfish sekarang dapat memanfaatkan utilitas dump dan restore untuk mencadangkan dan memulihkan database mereka. Untuk informasi lebih lanjut, lihat Babelfish dump and restore. Fitur ini dibangun di atas utilitas dump dan restore PostgreSQL. Untuk informasi selengkapnya, lihat pg_dump dan lihat pg_restore. Untuk menggunakan fitur ini secara efektif di Babelfish, Anda perlu menggunakan alat berbasis PostgreSQL yang secara khusus disesuaikan untuk Babelfish. Fitur backup dan restore untuk Babelfish berbeda secara signifikan dari SQL Server. Untuk informasi selengkapnya tentang perbedaan ini, lihat Membuang dan memulihkan perbedaan fungsionalitas: Babelfish dan SQL Server. Babelfish untuk Aurora PostgreSQL menyediakan kemampuan tambahan untuk mencadangkan dan memulihkan cluster Amazon Aurora PostgreSQL DB. Lihat informasi yang lebih lengkap di Mencadangkan dan memulihkan klaster DB Amazon Aurora.

Perbedaan T-SQL di Babelfish

Berikut cara Anda dapat menemukan tabel fungsionalitas T-SQL seperti yang didukung dalam rilis Babelfish saat ini dengan beberapa catatan tentang perbedaan perilaku dari SQL Server.

Untuk informasi selengkapnya tentang dukungan di berbagai versi, lihat Fungsionalitas yang didukung di Babelfish berdasarkan versi. Untuk informasi tentang fitur yang saat ini tidak didukung, lihat Fungsionalitas yang tidak didukung di Babelfish.

Babelfish tersedia dengan Aurora Edisi Kompatibel PostgreSQL. Untuk informasi lebih lanjut tentang rilis Babelfish, lihat Catatan Rilis untuk Aurora PostgreSQL.

Fungsionalitas atau sintaks	Deskripsi perilaku atau perbedaan
\ (karakter kelanjutan baris)	Karakter kelanjutan baris (garis miring terbalik sebelum baris baru) untuk karakter dan string heksadesimal saat ini tidak didukung. Untuk string karakter, backslash-newline ditafsirkan sebagai karakter dalam string. Untuk string heksadesimal, garis miring terbalik menghasilkan kesalahan sintaks.
@@version	Format nilai yang dikembalikan oleh `@@version` sedikit berbeda dari nilai yang dikembalikan oleh SQL Server. Kode

Fungsionalitas atau sintaks	Deskripsi perilaku atau perbedaan
	Anda mungkin tidak berfungsi dengan benar jika tergantung pada pemformatan `@@version` .
Fungsi agregat	Fungsi agregat didukung sebagian (AVG, COUNT, COUNT_BIG , GROUPING, MAX, MIN, STRING_AGG, dan SUM didukung) . Untuk daftar fungsi agregat yang tidak didukung, lihat. Fungsi yang tidak didukung
ALTER TABLE	Mendukung penambahan atau penjatuhan satu kolom atau kendala saja.
ALTER TABLE..ALTER COLUMN	NULL dan NOT NULL saat ini tidak dapat ditentukan. Untuk mengubah nullability kolom, gunakan pernyataan PostgreSQL ALTER TABLE.. {SET|DROP} NOT NULL.
Nama kolom kosong tanpa alias kolom	Utilitas `sqlcmd` dan `psql` menangani kolom dengan nama kosong secara berbeda: • SQL Server `sqlcmd` mengembalikan nama kolom kosong. • PostgreSQL `psql` mengembalikan nama kolom yang dihasilkan.
Fungsi CHECKSUM	Babelfish dan SQL Server menggunakan algoritma hashing yang berbeda untuk fungsi CHECKSUM. Akibatnya, nilai hash yang dihasilkan oleh fungsi CHECKSUM di Babelfish mungkin berbeda dari yang dihasilkan oleh fungsi CHECKSUM di SQL Server.
Kolom default	Saat membuat kolom default, nama kendala diabaikan. Untuk menjatuhkan kolom default, gunakan sintaks berikut: `ALTER TABLE...ALTER COLUMN..DROP DEFAULT...`
Kekurangan	PostgreSQL tidak mendukung pengaktifan dan penonaktifan kendala individu. Pernyataan itu diabaikan dan peringatan diajukan.

Fungsionalitas atau sintaks	Deskripsi perilaku atau perbedaan
Kendala dibuat dengan kolom DESC (menurun)	Kendala dibuat dengan kolom ASC (naik).
Kendala dengan IGNORE_DUP_KEY	Kendala dibuat tanpa properti ini.
CREATE, ALTER, DROP SERVER ROLE	ALTER SERVER ROLE hanya didukung untuk `sysadmin`. Semua sintaks lainnya tidak didukung. Pengguna T-SQL di Babelfish memiliki pengalaman yang mirip dengan SQL Server untuk konsep login (pengguna utama server), basis data, dan pengguna basis data user (pengguna utama basis data).
Klausa CREATE, ALTER LOGIN didukung dengan sintaks terbatas	CREATE LOGIN... Klausa PASSWORD,... klausa DEFAULT_DATABASE, dan... klausa DEFAULT_LANGUAGE didukung. ALTER LOGIN... Klausa PASSWORD didukung, tetapi ALTER LOGIN... Klausa OLD_PASSWORD tidak didukung. Hanya login yang merupakan anggota sysadmin yang dapat memodifikasi kata sandi.
Kolasi CREATE DATABASE peka huruf besar/kecil	Kolasi peka huruf besar/kecil tidak didukung dengan pernyataan CREATE DATABASE.
Kata kunci dan klausa CREATE DATABASE	Opsi kecuali COLLATE dan CONTAINMENT=NONE tidak didukung. Klausa COLLATE diterima dan selalu diatur ke nilai `babelfishpg_tsql.server_collation_name` .
CREATE SCHEMA... klausa pendukung	Anda dapat menggunakan perintah CREATE SCHEMA untuk membuat skema kosong. Gunakan perintah tambahan untuk membuat objek skema.
Nilai ID basis data berbeda pada Babelfish	Basis data utama dan tempdb tidak akan menjadi ID basis data 1 dan 2.

Fungsionalitas atau sintaks	Deskripsi perilaku atau perbedaan
Fungsi format tipe tanggal TO_CHAR didukung dengan batasan berikut	Meridian karakter tunggal tidak didukung. Format "yyy" di server SQL mengembalikan 4 digit untuk tahun di atas 1000, tetapi hanya 3 digit untuk yang lain. Format "g" dan "R" tidak didukung Terjemahan lokal "Vi-VN" sedikit berbeda.
Pengidentifikasi melebihi 63 karakter	PostgreSQL mendukung maksimal 63 karakter untuk pengident ifikasi. Babelfish mengonversi pengenal lebih dari 63 karakter ke nama yang menyertakan hash dari nama aslinya. Misalnya, tabel yang dibuat sebagai "AB(ABC12345678901234567890 12345678901234567890123456789012345678901234567890" mungkin akan dikonversi menjadi "ABC12345678901234 56789012345678901234567890123456789012345678901234567890".
Dukungan kolom IDENTITY	Kolom `IDENTITY` didukung untuk tipe data `tinyint`, `smallint`, `int`, `bigint`. `numeric`, dan `decimal`. SQL Server mendukung presisi ke 38 tempat untuk tipe data `numeric` dan `decimal` di kolom IDENTITY. PostgreSQL mendukung presisi ke 19 tempat untuk tipe data `numeric` dan `decimal` di kolom IDENTITY.
Indeks dengan IGNORE_DU P_KEY	Sintaks yang membuat indeks yang menyertakan IGNORE_DU P_KEY membuat indeks seolah-olah properti ini dihilangkan.
Indeks dengan lebih dari 32 kolom	Indeks tidak dapat menyertakan lebih dari 32 kolom. Kolom indeks yang disertakan dihitung menuju maksimum di PostgreSQL tetapi tidak di SQL Server.
Indeks (berklaster)	Indeks berklaster dibuat seolah-olah NONCLUSTERED ditentukan.

Fungsionalitas atau sintaks	Deskripsi perilaku atau perbedaan
Klausa indeks	Klausul berikut diabaikan: FILLFACTOR, ALLOW_PAG E_LOCKS, ALLOW_ROW_LOCKS, PAD_INDEX, STATISTIC S_NORECOMPUTE, OPTIMIZE_FOR_SEQUENTIAL_KEY, SORT_IN_TEMPDB, DROP_EXISTING, ONLINE, COMPRESSI ON_DELAY, MAXDOP, dan DATA_COMPRESSION
Dukungan JSON	Urutan pasangan nama-nilai tidak dijamin. Tetapi tipe array tetap tidak terpengaruh.
Objek LOGIN	Semua opsi untuk objek LOGIN didukung kecuali PASSWORD, DEFAULT_DATABASE, ENABLE, DISABLE.
Fungsi NEWSEQUENTIALID	Diimplementasikan sebagai NEWID; perilaku berurutan tidak dijamin. Saat memanggil `NEWSEQUENTIALID` , PostgreSQL menghasilkan nilai GUID baru.
Klausa OUTPUT didukung dengan batasan berikut	OUTPUT dan OUTPUT INTO tidak didukung dalam kueri DML yang sama. Referensi ke tabel non-target operasi UPDATE atau DELETE dalam klausa OUTPUT tidak didukung. OUTPUT... DELETED *, INSERTED * tidak didukung dalam kueri yang sama.
Batas parameter prosedur atau fungsi	Babelfish mendukung maksimum 100 parameter untuk prosedur atau fungsi.
ROWGUIDCOL	Klausa ini diabaikan untuk sekarang. Kueri referensi `$GUIDGOL` menyebabkan kesalahan sintaks.
Dukungan objek SEQUENCE	Objek SEQUENCE didukung untuk tipe data tinyint, smallint, int, bigint, numerik, dan desimal. Aurora PostgreSQL mendukung presisi ke 19 tempat untuk tipe data numerik dan desimal dalam SEQUENCE.
Peran tingkat server	Peran tingkat server `sysadmin` didukung. Peran tingkat server lainnya (selain `sysadmin`) tidak didukung.

Fungsionalitas atau sintaks	Deskripsi perilaku atau perbedaan
Peran tingkat basis data selain `db_owner`	Peran `db_owner` tingkat basis data dan tingkat basis data yang ditentukan pengguna didukung. Peran tingkat basis data lainnya (selain db_owner) tidak didukung.
Kata kunci `SPARSE` SQL	Kata kunci SPARSE diterima dan diabaikan.
Klausa kata kunci SQL `ON filegroup`	Klausa ini diabaikan untuk sekarang.
Kata kunci SQL `CLUSTERED` dan `NONCLUSTERED` untuk indeks dan kendala	Babelfish menerima dan mengabaikan kata kunci `CLUSTERED` dan `NONCLUSTERED` .
`sysdatabases.cmptlevel`	`sysdatabases.cmptlevel` selalu diatur ke 120.
tempdb tidak diinisialisasi ulang saat restart	Objek permanen (seperti tabel dan prosedur) yang dibuat di tempdb tidak dihapus saat basis data direstart.
Kumpulan file TEXTIMAGE_ON	Babelfish mengabaikan klausa `TEXTIMAGE_ON` *`filegroup`* .
Ketepatan waktu	Babelfish mendukung presisi 6 digit untuk detik pecahan. Tidak ada efek samping yang diantisipasi dengan perilaku ini.
Tingkat isolasi transaksi	READUNCOMMITTED diperlakukan sama dengan READCOMMITTED.
Kolom komputasi virtual (nonpersisten)	Kolom komputasi virtual dibuat sebagai persisten.
Tanpa klausa SCHEMABINDING	Klausa ini tidak didukung dalam fungsi, prosedur, pemicu, atau tampilan. Objek dibuat, tetapi seolah-olah WITH SCHEMABINDING ditentukan.

Tingkat Isolasi Transaksi di Babelfish

Babelfish mendukung Tingkat Isolasi Transaksi READ UNCOMMITTED, READ COMMITTED dan SNAPSHOT. Mulai dari versi Babelfish 3.4 Tingkat Isolasi tambahan REPEATABLE READING dan SERIALIZABLE didukung. Semua Tingkat Isolasi di Babelfish didukung dengan perilaku Tingkat Isolasi yang sesuai di PostgreSQL. SQL Server dan Babelfish menggunakan mekanisme dasar yang berbeda untuk menerapkan Tingkat Isolasi Transaksi (memblokir untuk akses bersamaan, kunci yang dipegang oleh transaksi, penanganan kesalahan, dll). Dan, ada beberapa perbedaan halus dalam cara akses bersamaan dapat bekerja untuk beban kerja yang berbeda. Untuk informasi selengkapnya tentang perilaku PostgreSQL ini, lihat Isolasi Transaksi.

Topik

- Ikhtisar Tingkat Isolasi Transaksi
- Menyiapkan Tingkat Isolasi Transaksi
- Mengaktifkan atau menonaktifkan Tingkat Isolasi Transaksi
- Perbedaan Antara Tingkat Isolasi Babelfish dan SQL Server

Ikhtisar Tingkat Isolasi Transaksi

Tingkat Isolasi Transaksi SQL Server asli didasarkan pada penguncian pesimis di mana hanya satu salinan data yang ada dan kueri harus mengunci sumber daya seperti baris sebelum mengaksesnya. Kemudian, variasi Tingkat Isolasi Berkomitmen Baca diperkenalkan. Ini memungkinkan penggunaan versi baris untuk memberikan konkurensi yang lebih baik antara pembaca dan penulis menggunakan akses non-pemblokiran. Selain itu, Level Isolasi baru yang disebut Snapshot tersedia. Ini juga menggunakan versi baris untuk memberikan konkurensi yang lebih baik daripada Tingkat Isolasi BACA YANG DAPAT DIULANG dengan menghindari kunci bersama pada data baca yang disimpan hingga akhir transaksi.

Tidak seperti SQL Server, semua Tingkat Isolasi Transaksi di Babelfish didasarkan pada Optimistic Locking (MVCC). Setiap transaksi melihat snapshot data baik di awal pernyataan (READ COMMITTED) atau di awal transaksi (REPEATABLE READ, SERIALIZABLE), terlepas dari keadaan saat ini dari data yang mendasarinya. Oleh karena itu, perilaku eksekusi transaksi bersamaan di Babelfish mungkin berbeda dari SQL Server.

Misalnya, pertimbangkan transaksi dengan Isolation Level SERIALIZABLE yang awalnya diblokir di SQL Server tetapi berhasil nanti. Ini mungkin berakhir gagal di Babelfish karena konflik serialisasi dengan transaksi bersamaan yang membaca atau memperbarui baris yang sama. Mungkin juga ada kasus di mana mengeksekusi beberapa transaksi bersamaan menghasilkan hasil akhir yang berbeda

di Babelfish dibandingkan dengan SQL Server. Aplikasi yang menggunakan Tingkat Isolasi, harus diuji secara menyeluruh untuk skenario konkurensi.

Tingkat Isolasi di SQL Server	Tingkat Isolasi Babelfish	Tingkat Isolasi PostgreSQL	Komentar
BACA UNCOMMIT	BACA UNCOMMIT	BACA UNCOMMIT	Read Uncommitted sama dengan Read Comited di BabelFish/ PostgreSQL
BACA BERKOMITM EN	BACA BERKOMITM EN	BACA BERKOMITM EN	SQL Server Read Comited berbasis penguncian pesimis, Babelfish Read Committed berbasis snapshot (MVCC).
BACA SNAPSHOT YANG BERKOMITM EN	BACA BERKOMITM EN	BACA BERKOMITM EN	Keduanya berbasis snapshot (MVCC) tetapi tidak persis sama.
REKAM JEPRET	REKAM JEPRET	DIBACA BERULANG	Persis sama.
DIBACA BERULANG	DIBACA BERULANG	DIBACA BERULANG	SQL Server Repeatable Read berbasis penguncia n pesimis, Babelfish Repeatable Read berbasis snapshot (MVCC).
DAPAT DISERIALK AN	DAPAT DISERIALK AN	DAPAT DISERIALK AN	SQL Server Serializa ble adalah isolasi pesimis, Babelfish Serializable berbasis snapshot (MVCC).

> **Note**
>
> Petunjuk tabel saat ini tidak didukung dan perilakunya dikendalikan dengan menggunakan palka pelarian Babelfish yang telah ditentukan sebelumnya. `escape_hatch_table_hints`

Menyiapkan Tingkat Isolasi Transaksi

Gunakan perintah berikut untuk mengatur Tingkat Isolasi Transaksi:

Example

```
SET TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ |
 SNAPSHOT | SERIALIZABLE }
```

Mengaktifkan atau menonaktifkan Tingkat Isolasi Transaksi

Tingkat Isolasi Transaksi REPEATABLE READ dan SERIALIZABLE dinonaktifkan secara default di Babelfish dan Anda harus mengaktifkannya secara eksplisit dengan menyetel atau escape hatch untuk digunakan. `babelfishpg_tsql.isolation_level_serializable` `babelfishpg_tsql.isolation_level_repeatable_read` `pg_isolation` `sp_babelfish_configure` Untuk informasi selengkapnya, lihat Mengelola penanganan kesalahan Babelfish dengan escape hatch.

Di bawah ini adalah contoh untuk mengaktifkan atau menonaktifkan penggunaan REPEATABLE READ dan SERIALIZABLE di sesi saat ini dengan mengatur lubang keluar masing-masing. Secara opsional sertakan `server` parameter untuk mengatur escape hatch untuk sesi saat ini serta untuk semua sesi baru berikutnya.

Untuk mengaktifkan penggunaan SET TRANSACTION ISOLATION LEVEL REPEATABLE REPEATABLE READ hanya pada sesi saat ini.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation'
```

Untuk mengaktifkan penggunaan SET TRANSACTION ISOLATION LEVEL REPEATABLE REPEATABLE READING di sesi saat ini dan semua sesi baru yang diakibatkannya.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'pg_isolation',
 'server'
```

Untuk menonaktifkan penggunaan SET TRANSACTION ISOLATION LEVEL REPEATABLE REPEATABLE REPEATABLE READ di sesi saat ini dan sesi baru konsekuen.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_repeatable_read', 'off', 'server'
```

Untuk mengaktifkan penggunaan SET TRANSACTION ISOLATION LEVEL SERIALIZABLE pada sesi saat ini saja.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation'
```

Untuk mengaktifkan penggunaan SET TRANSACTION ISOLATION LEVEL SERIALIZABLE di sesi saat ini dan semua sesi baru yang diakibatkannya.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'pg_isolation', 'server'
```

Untuk menonaktifkan penggunaan SET TRANSACTION ISOLATION LEVEL SERIALIZABLE di sesi saat ini dan selanjutnya sesi baru.

Example

```
EXECUTE sp_babelfish_configure 'isolation_level_serializable', 'off', 'server'
```

Perbedaan Antara Tingkat Isolasi Babelfish dan SQL Server

Di bawah ini adalah beberapa contoh tentang nuansa bagaimana SQL Server dan Babelfish mengimplementasikan ANSI Isolation Levels.

> **Note**
>
> - Tingkat Isolasi Repeatable Read dan Snapshot sama di Babelfish.
> - Tingkat Isolasi Baca Tidak Berkomitmen dan Baca Berkomitmen sama di Babelfish.

Contoh berikut menunjukkan cara membuat tabel dasar untuk semua contoh yang disebutkan di bawah ini:

```
CREATE TABLE employee (
    id sys.INT NOT NULL PRIMARY KEY,
    name sys.VARCHAR(255)NOT NULL,
    age sys.INT NOT NULL
);
INSERT INTO employee (id, name, age) VALUES (1, 'A', 10);
INSERT INTO employee (id, name, age) VALUES (2, 'B', 20);
INSERT INTO employee (id, name, age) VALUES (3, 'C', 30);
```

Topik

- BABELFISH MEMBACA UNCOMMITTED VS SQL SERVER MEMBACA TINGKAT ISOLASI TANPA KOMITMEN
- BABELFISH READ COMMIT VS SQL SERVER MEMBACA TINGKAT ISOLASI YANG BERKOMITMEN
- BABELFISH READ COMMIT VS SQL SERVER MEMBACA TINGKAT ISOLASI SNAPSHOT YANG BERKOMITMEN
- BABELFISH REPEABLE READ VS SQL SERVER TINGKAT ISOLASI BACA BERULANG
- BABELFISH SERIALIZABLE VS SQL SERVER TINGKAT ISOLASI SERIALIZABLE

BABELFISH MEMBACA UNCOMMITTED VS SQL SERVER MEMBACA TINGKAT ISOLASI TANPA KOMITMEN

BACAAN KOTOR DI SERVER SQL

Transaksi 1	Transaksi 2	SQL Server Baca Tidak Berkomitmen	Babelfish Baca Tanpa Komitmen
MULAI TRANSAKSI	MULAI TRANSAKSI		
MENGATUR TINGKAT ISOLASI TRANSAKSI BACA TANPA KOMITMEN;	MENGATUR TINGKAT ISOLASI TRANSAKSI BACA TANPA KOMITMEN;		
	UPDATE karyawan SET age = 0;	Pembaruan berhasil.	Pembaruan berhasil.
	MASUKKAN KE DALAM NILAI KARYAWAN (4, 'D', 40);	Masukkan berhasil.	Masukkan berhasil.
PILIH * DARI KARYAWAN;		Transaksi 1 dapat melihat perubahan tanpa komitmen dari Transaksi 2.	Sama seperti Read Comited di Babelfish . Perubahan yang tidak dilakukan dari Transaksi 2 tidak terlihat oleh Transaksi 1.
	COMMIT		
PILIH * DARI KARYAWAN;		Melihat perubahan yang dilakukan oleh Transaksi 2.	Melihat perubahan yang dilakukan oleh Transaksi 2.

BABELFISH READ COMMIT VS SQL SERVER MEMBACA TINGKAT ISOLASI YANG BERKOMITMEN

BACA - TULIS PEMBLOKIRAN

Transaksi 1	Transaksi 2	SQL Server Baca Berkomitmen	Babelfish Baca Berkomitmen
MULAI TRANSAKSI	MULAI TRANSAKSI		
MENGATUR TINGKAT ISOLASI TRANSAKSI BACA KOMITMEN;	MENGATUR TINGKAT ISOLASI TRANSAKSI BACA KOMITMEN;		
PILIH * DARI KARYAWAN;			
	UPDATE karyawan SET age = 100 WHERE id = 1;	Pembaruan berhasil.	Pembaruan berhasil.
UPDATE karyawan SET usia = 0 DIMANA usia DI (PILIH MAX (usia) DARI karyawan);		Langkah diblokir hingga Transaksi 2 dilakukan.	Transaksi 2 perubahan belum terlihat. Update baris dengan id = 3.
	COMMIT	Transaksi 2 berhasil dilakukan. Transaksi 1 sekarang tidak diblokir dan melihat pembaruan dari Transaksi 2.	Transaksi 2 berhasil dilakukan.
PILIH * DARI KARYAWAN;		Transaksi 1 memperbarui baris dengan id = 1.	Transaksi 1 memperbarui baris dengan id = 3.

BABELFISH READ COMMIT VS SQL SERVER MEMBACA TINGKAT ISOLASI SNAPSHOT YANG BERKOMITMEN

MEMBLOKIR PERILAKU PADA BARIS BARU YANG DISISIPKAN

Transaksi 1	Transaksi 2	SQL Server Membaca Snapshot Berkomitmen	Babelfish Baca Berkomitmen
MULAI TRANSAKSI	MULAI TRANSAKSI		
MENGATUR TINGKAT ISOLASI TRANSAKSI BACA KOMITMEN;	MENGATUR TINGKAT ISOLASI TRANSAKSI BACA KOMITMEN;		
MASUKKAN KE DALAM NILAI KARYAWAN (4, 'D', 40);			
	UPDATE karyawan SET usia = 99;	Langkah diblokir sampai transaksi 1 dilakukan. Baris yang dimasukkan dikunci oleh transaksi 1.	Diperbarui tiga baris. Baris yang baru dimasukkan belum terlihat.
COMMIT		Berkomitmen sukses. Transaksi 2 sekarang tidak diblokir.	Berkomitmen sukses.
	PILIH * DARI KARYAWAN;	Semua 4 baris memiliki usia = 99.	Baris dengan id = 4 memiliki nilai usia 40 karena tidak terlihat oleh transaksi 2 selama kueri pembaruan. Baris lainnya diperbarui ke age=99.

BABELFISH REPEABLE READ VS SQL SERVER TINGKAT ISOLASI BACA BERULANG

MEMBACA/MENULIS PERILAKU PEMBLOKIRAN

Transaksi 1	Transaksi 2	SQL Server yang Dapat Dibaca Berulang	Babelfish yang Dapat Dibaca Berulang
MULAI TRANSAKSI	MULAI TRANSAKSI		
MENGATUR TINGKAT ISOLASI TRANSAKSI YANG DAPAT DIBACA BERULANG;	MENGATUR TINGKAT ISOLASI TRANSAKSI YANG DAPAT DIBACA BERULANG;		
PILIH * DARI KARYAWAN;			
UPDATE karyawan SET NAME='A_TXN1' WHERE id = 1;			
	PILIH * DARI karyawan WHERE id! = 1;		
	PILIH * DARI KARYAWAN;	Transaksi 2 diblokir sampai Transaksi 1 dilakukan.	Transaksi 2 berlangsung secara normal.
COMMIT			
	PILIH * DARI KARYAWAN;	Pembaruan dari Transaksi 1 terlihat.	Pembaruan dari Transaksi 1 tidak terlihat.
COMMIT			

Transaksi 1	Transaksi 2	SQL Server yang Dapat Dibaca Berulang	Babelfish yang Dapat Dibaca Berulang
	PILIH * DARI KARYAWAN;	melihat pembaruan dari Transaksi 1.	melihat pembaruan dari Transaksi 1.

MENULIS/MENULIS PERILAKU PEMBLOKIRAN

Transaksi 1	Transaksi 2	SQL Server yang Dapat Dibaca Berulang	Babelfish yang Dapat Dibaca Berulang
MULAI TRANSAKSI	MULAI TRANSAKSI		
MENGATUR TINGKAT ISOLASI TRANSAKSI YANG DAPAT DIBACA BERULANG;	MENGATUR TINGKAT ISOLASI TRANSAKSI YANG DAPAT DIBACA BERULANG;		
UPDATE karyawan SET NAME='A_TXN1' WHERE id = 1;			
	UPDATE karyawan SET NAME='A_TXN2' WHERE id = 1;	Transaksi 2 diblokir.	Transaksi 2 diblokir.
COMMIT		Komit berhasil dan transaksi 2 telah diblokir.	Komit berhasil dan transaksi 2 gagal dengan kesalahan tidak dapat membuat serial akses karena pembaruan bersamaan.

Transaksi 1	Transaksi 2	SQL Server yang Dapat Dibaca Berulang	Babelfish yang Dapat Dibaca Berulang
	COMMIT	Berkomitmen sukses.	Transaksi 2 telah dibatalkan.
	PILIH * DARI KARYAWAN;	Baris dengan id=1 memiliki name='A_T X2'.	Baris dengan id=1 memiliki name='A_T X1'.

HANTU MEMBACA

Transaksi 1	Transaksi 2	SQL Server yang Dapat Dibaca Berulang	Babelfish yang Dapat Dibaca Berulang
MULAI TRANSAKSI	MULAI TRANSAKSI		
MENGATUR TINGKAT ISOLASI TRANSAKSI YANG DAPAT DIBACA BERULANG;	MENGATUR TINGKAT ISOLASI TRANSAKSI YANG DAPAT DIBACA BERULANG;		
PILIH * DARI KARYAWAN;			
	MASUKKAN KE DALAM NILAI KARYAWAN (4, NewRowName '', 20);	Transaksi 2 berlangsung tanpa pemblokiran apa pun.	Transaksi 2 berlangsung tanpa pemblokiran apa pun.
	PILIH * DARI KARYAWAN;	Baris yang baru dimasukkan terlihat.	Baris yang baru dimasukkan terlihat.
	COMMIT		

Transaksi 1	Transaksi 2	SQL Server yang Dapat Dibaca Berulang	Babelfish yang Dapat Dibaca Berulang
PILIH * DARI KARYAWAN;		Baris baru yang dimasukkan oleh transaksi 2 terlihat.	Baris baru yang dimasukkan oleh transaksi 2 tidak terlihat.
COMMIT			
PILIH * DARI KARYAWAN;		Baris yang baru dimasukkan terlihat.	Baris yang baru dimasukkan terlihat.

HASIL AKHIR YANG BERBEDA

Transaksi 1	Transaksi 2	SQL Server yang Dapat Dibaca Berulang	Babelfish yang Dapat Dibaca Berulang
MULAI TRANSAKSI	MULAI TRANSAKSI		
MENGATUR TINGKAT ISOLASI TRANSAKSI YANG DAPAT DIBACA BERULANG;	MENGATUR TINGKAT ISOLASI TRANSAKSI YANG DAPAT DIBACA BERULANG;		
UPDATE karyawan SET usia = 100 WHERE age IN (PILIH MIN (usia) DARI karyawan);		Transaksi 1 memperbarui baris dengan id 1.	Transaksi 1 memperbarui baris dengan id 1.
	UPDATE karyawan SET usia = 0 DIMANA usia DI (PILIH	Transaksi 2 diblokir karena pernyataan SELECT mencoba membaca baris yang	Transaksi 2 berlangsung tanpa pemblokiran karena baca tidak pernah diblokir,

Transaksi 1	Transaksi 2	SQL Server yang Dapat Dibaca Berulang	Babelfish yang Dapat Dibaca Berulang
	MAX (usia) DARI karyawan);	dikunci oleh kueri UPDATE dalam transaksi 1.	pernyataan SELECT dijalankan dan akhirnya baris dengan id = 3 diperbarui karena perubahan transaksi 1 belum terlihat.
	PILIH * DARI KARYAWAN;	Langkah ini dilakukan setelah transaksi 1 dilakukan. Baris dengan id = 1 diperbarui oleh transaksi 2 pada langkah sebelumnya dan terlihat di sini.	Baris dengan id = 3 diperbarui oleh Transaksi 2.
COMMIT		Transaksi 2 sekarang tidak diblokir.	Berkomitmen sukses.
	COMMIT		
PILIH * DARI KARYAWAN;		Kedua transaksi mengeksekusi pembaruan pada baris dengan id = 1.	Baris yang berbeda diperbarui dengan transaksi 1 dan 2.

BABELFISH SERIALIZABLE VS SQL SERVER TINGKAT ISOLASI SERIALIZABLE

KUNCI RENTANG DI SERVER SQL

Transaksi 1	Transaksi 2	SQL Server Serial	Babelfish Serializable
MULAI TRANSAKSI	MULAI TRANSAKSI		

Transaksi 1	Transaksi 2	SQL Server Serial	Babelfish Serializable
MENGATUR TINGKAT ISOLASI TRANSAKSI SERILAIZABLE;	MENGATUR TINGKAT ISOLASI TRANSAKSI SERILAIZABLE;		
PILIH * DARI KARYAWAN;			
	MASUKKAN KE DALAM NILAI KARYAWAN (4, 'D', 35);	Transaksi 2 diblokir sampai Transaksi 1 dilakukan.	Transaksi 2 berlangsung tanpa pemblokiran apa pun.
	PILIH * DARI KARYAWAN;		
COMMIT		Transaksi 1 berhasil dilakukan. Transaksi 2 sekarang tidak diblokir.	Transaksi 1 berhasil dilakukan.
	COMMIT		
PILIH * DARI KARYAWAN;		Baris yang baru dimasukkan terlihat.	Baris yang baru dimasukkan terlihat.

HASIL AKHIR YANG BERBEDA

Transaksi 1	Transaksi 2	SQL Server Serial	Babelfish Serializable
MULAI TRANSAKSI	MULAI TRANSAKSI		
MENGATUR TINGKAT ISOLASI TRANSAKSI SERILAIZABLE;	MENGATUR TINGKAT ISOLASI TRANSAKSI SERILAIZABLE;		

Transaksi 1	Transaksi 2	SQL Server Serial	Babelfish Serializable
	MASUKKAN KE DALAM NILAI KARYAWAN (4, 'D', 40);		
UPDATE karyawan SET usia = 99 WHERE id = 4;		Transaksi 1 diblokir hingga Transaksi 2 dilakukan.	Transaksi 1 berlangsung tanpa pemblokiran apa pun.
	COMMIT	Transaksi 2 berhasil dilakukan. Transaksi 1 sekarang tidak diblokir.	Transaksi 2 berhasil dilakukan.
COMMIT			
PILIH * DARI KARYAWAN;		Baris yang baru dimasukkan terlihat dengan nilai usia = 99.	Baris yang baru dimasukkan terlihat dengan nilai usia = 40.

MASUKKAN KE DALAM TABEL DENGAN KENDALA UNIK

Transaksi 1	Transaksi 2	SQL Server Serial	Babelfish Serializable
MULAI TRANSAKSI	MULAI TRANSAKSI		
MENGATUR TINGKAT ISOLASI TRANSAKSI SERILAIZABLE;	MENGATUR TINGKAT ISOLASI TRANSAKSI SERILAIZABLE;		
	MASUKKAN KE DALAM NILAI KARYAWAN (4, 'D', 40);		

Transaksi 1	Transaksi 2	SQL Server Serial	Babelfish Serializable
MASUKKAN KE DALAM NILAI karyawan ((PILIH MAX (id) +1 DARI karyawan), 'E', 50);		Transaksi 1 diblokir hingga Transaksi 2 dilakukan.	Transaksi 1 diblokir hingga Transaksi 2 dilakukan.
	COMMIT	Transaksi 2 berhasil dilakukan. Transaksi 1 sekarang tidak diblokir.	Transaksi 2 berhasil dilakukan. Transaksi 1 dibatalkan dengan kesalahan nilai kunci duplikat melanggar batasan unik.
COMMIT		Transaksi 1 berhasil dilakukan.	Transaksi 1 komit gagal dengan tidak dapat membuat serial akses karena ketergantungan baca/tulis di antara transaksi.
PILIH * DARI KARYAWAN;		baris (5, 'E', 50) dimasukkan.	Hanya ada 4 baris.

Di Babelfish, transaksi bersamaan yang berjalan dengan Isolation Level serializable akan gagal dengan kesalahan anomali serialisasi jika eksekusi transaksi ini tidak konsisten dengan semua kemungkinan eksekusi serial (satu per satu) dari transaksi tersebut.

ANOMALI SERIALISASI

Transaksi 1	Transaksi 2	SQL Server Serial	Babelfish Serializable
MULAI TRANSAKSI	MULAI TRANSAKSI		
MENGATUR TINGKAT ISOLASI	MENGATUR TINGKAT ISOLASI		

Transaksi 1	Transaksi 2	SQL Server Serial	Babelfish Serializable
TRANSAKSI SERILAIZABLE;	TRANSAKSI SERILAIZABLE;		
PILIH * DARI KARYAWAN;			
UPDATE karyawan SET age = 5 WHERE age = 10;			
	PILIH * DARI KARYAWAN;	Transaksi 2 diblokir sampai Transaksi 1 dilakukan.	Transaksi 2 berlangsung tanpa pemblokiran apa pun.
	UPDATE karyawan SET age = 35 WHERE age = 30;		
COMMIT		Transaksi 1 berhasil dilakukan.	Transaksi 1 dilakukan terlebih dahulu dan mampu melakukan komit dengan sukses.
	COMMIT	Transaksi 2 berhasil dilakukan.	Komit transaksi 2 gagal dengan kesalahan serialisasi, seluruh transaksi telah dibatalkan. Coba lagi transaksi 2.
PILIH * DARI KARYAWAN;		Perubahan dari kedua transaksi terlihat.	Transaksi 2 digulirkan kembali. Hanya transaksi 1 perubahan yang terlihat.

Di Babelfish, anomali serialisasi hanya mungkin jika semua transaksi bersamaan dijalankan di Isolation Level SERIALIZABLE. Misalnya mari kita ambil contoh di atas tetapi atur transaksi 2 ke Tingkat Isolasi REPEATABLE READING sebagai gantinya.

Transaksi 1	Transaksi 2	Tingkat Isolasi SQL Server	Tingkat Isolasi Babelfish
MULAI TRANSAKSI	MULAI TRANSAKSI		
MENGATUR TINGKAT ISOLASI TRANSAKSI SERILAIZABLE;	MENGATUR TINGKAT ISOLASI TRANSAKSI YANG DAPAT DIBACA BERULANG;		
PILIH * DARI KARYAWAN;			
UPDATE karyawan SET age = 5 WHERE age = 10;			
	PILIH * DARI KARYAWAN;	Transaksi 2 diblokir sampai transaksi 1 dilakukan.	Transaksi 2 berlangsung tanpa pemblokiran apa pun.
	UPDATE karyawan SET age = 35 WHERE age = 30;		
COMMIT		Transaksi 1 berhasil dilakukan.	Transaksi 1 berhasil dilakukan.
	COMMIT	Transaksi 2 berhasil dilakukan.	Transaksi 2 berhasil dilakukan.
PILIH * DARI KARYAWAN;		Perubahan dari kedua transaksi terlihat.	Perubahan dari kedua transaksi terlihat.

Menggunakan fitur Babelfish dengan implementasi terbatas

Setiap versi baru Babelfish menambahkan dukungan untuk fitur yang lebih selaras dengan fungsionalitas dan perilaku T-SQL. Namun, ada beberapa fitur yang tidak didukung dan perbedaan dalam implementasi saat ini. Berikut ini, Anda dapat menemukan informasi tentang perbedaan fungsional antara Babelfish dan T-SQL, dengan beberapa solusi atau catatan penggunaan.

Pada versi 1.2.0 dari Babelfish, fitur berikut saat ini memiliki implementasi terbatas:

- Katalog SQL Server (tampilan sistem) – Katalog `sys.sysconfigures`, `sys.syscurconfigs`, dan `sys.configurations` mendukung konfigurasi hanya-baca tunggal saja. Saat ini, `sp_configure` belum didukung. Untuk informasi selengkapnya tentang tampilan SQL Server lainnya yang diterapkan oleh Babelfish, lihat Mendapatkan informasi dari katalog sistem Babelfish.
- Izin GRANT – GRANT…TO PUBLIC didukung, tetapi GRANT..TO PUBLIC WITH GRANT OPTION saat ini tidak didukung.
- Rantai kepemilikan dan batasan mekanisme izin SQL Server – Di Babelfish, rantai kepemilikan SQL Server berfungsi untuk tampilan tetapi tidak untuk prosedur yang disimpan. Ini berarti bahwa prosedur harus diberikan akses eksplisit ke objek lain yang dimiliki oleh pemilik yang sama dengan prosedur panggilan. Di SQL Server, memberikan izin EXECUTE pemanggil pada prosedur cukup untuk memanggil objek lain yang dimiliki oleh pemilik yang sama. Di Babelfish, pemanggil juga harus diberikan izin pada objek yang diakses oleh prosedur.
- Resolusi referensi objek yang tidak memenuhi syarat (tanpa nama skema) – Ketika objek SQL (prosedur, tampilan, fungsi atau pemicu) mereferensikan objek tanpa memenuhi syarat dengan nama skema, SQL Server menyelesaikan nama skema objek dengan menggunakan nama skema objek SQL tempat referensi terjadi. Saat ini, Babelfish memiliki penyelesaian yang berbeda, dengan menggunakan skema default dari pengguna basis data yang menjalankan prosedur.
- Perubahan, sesi, dan koneksi skema default – Jika pengguna mengubah skema defaultnya dengan `ALTER USER...WITH DEFAULT SCHEMA`, perubahan akan segera berlaku di sesi tersebut. Namun, untuk sesi lain yang saat ini terhubung milik pengguna yang sama, waktunya berbeda, sebagai berikut:
  - Untuk SQL Server: – Perubahan akan langsung berlaku di semua koneksi lain untuk pengguna ini.
  - Untuk Babelfish: – Perubahan hanya berlaku di koneksi baru untuk pengguna ini.
- Implementasi tipe data ROWVERSION dan TIMESTAMP dan pengaturan escape hatch – Tipe data ROWVERSION dan TIMESTAMP sekarang didukung di Babelfish. Untuk menggunakan ROWVERSION atau TIMESTAMP di Babelfish, Anda harus mengubah pengaturan untuk escape

hatch `babelfishpg_tsql.escape_hatch_rowversion` dari default (strict) menjadi `ignore`. Implementasi Babelfish dari tipe data ROWVERSION dan TIMESTAMP sebagian besar identik secara semantik dengan SQL Server, dengan pengecualian berikut:

- Fungsi @@DBTS bawaan berperilaku mirip dengan SQL Server, tetapi dengan perbedaan kecil. Daripada mengembalikan nilai yang terakhir digunakan untuk `SELECT @@DBTS`, Babelfish menghasilkan stempel waktu baru, karena mesin basis data PostgreSQL yang mendasarinya dan implementasi kontrol konkurensi multi-versi (MVCC).
- Di SQL Server, setiap baris yang disisipkan atau diperbarui mendapatkan nilai ROWVERSION/TIMESTAMP yang unik. Di Babelfish, setiap baris yang disisipkan diperbarui oleh pernyataan yang sama diberi nilai ROWVERSION/TIMESTAMP yang sama.

  Misalnya, ketika pernyataan UPDATE atau pernyataan INSERT-SELECT memengaruhi beberapa baris, di SQL Server, semua baris yang terpengaruh memiliki nilai yang berbeda di kolom ROWVERSION/TIMESTAMP masing-masing. Dalam Babelfish (PostgreSQL), baris memiliki nilai yang sama.
- Di SQL Server, saat Anda membuat tabel baru dengan SELECT-INTO, Anda dapat mentransmisikan nilai eksplisit (seperti NULL) ke kolom ROWVERSION/TIMESTAMP yang akan dibuat. Saat Anda melakukan hal yang sama di Babelfish, nilai ROWVERSION/TIMESTAMP yang sebenarnya ditetapkan ke setiap baris di tabel baru untuk Anda, oleh Babelfish.

Perbedaan kecil dalam tipe data ROWVERSION/TIMESTAMP ini seharusnya tidak berdampak buruk pada aplikasi yang berjalan di Babelfish.

Pembuatan skema, kepemilikan, dan izin – Izin untuk membuat dan mengakses objek dalam skema yang dimiliki oleh pengguna non-DBO (menggunakan `CREATE SCHEMA` *`schema name`* `AUTHORIZATION` *`user name`*) berbeda untuk pengguna SQL Server dan Babelfish non-DBO, seperti yang ditunjukkan pada tabel berikut:

Pengguna basis data (non-DBO) yang memiliki skema dapat melakukan hal berikut:	SQL Server	Babelfish
Buat objek dalam skema tanpa pemberian izin tambahan oleh DBO?	Tidak	Ya
	Ya	Tidak

Pengguna basis data (non-DBO) yang memiliki skema dapat melakukan hal berikut:	SQL Server	Babelfish
Akses objek yang dibuat oleh DBO dalam skema tanpa pemberian izin tambahan?		

Meningkatkan performa kueri Babelfish

Anda dapat mencapai pemrosesan kueri yang lebih cepat di Babelfish menggunakan petunjuk kueri dan pengoptimal PostgreSQL.

Topik

- Menggunakan rencana explain untuk meningkatkan performa kueri Babelfish
- Menggunakan petunjuk kueri T-SQL untuk meningkatkan kinerja kueri Babelfish

Anda juga dapat meningkatkan performa kueri menggunakan prosedur `sp_babelfish_volatility`. Untuk informasi selengkapnya, lihat sp_babelfish_volatility.

Menggunakan rencana explain untuk meningkatkan performa kueri Babelfish

Mulai versi 2.1.0, Babelfish menyertakan dua fungsi yang secara transparan menggunakan pengoptimal PostgreSQL guna menghasilkan perkiraan dan rencana kueri aktual untuk kueri T-SQL di port TDS. Fungsi tersebut mirip dengan penggunaan SET STATISTICS PROFILE atau SET SHOWPLAN_ALL dengan basis data SQL Server untuk mengidentifikasi dan meningkatkan kueri yang berjalan lambat.

> **Note**
>
> Mendapatkan paket kueri dari fungsi, alur kontrol, dan kursor saat ini tidak didukung.

Di tabel, Anda dapat menemukan perbandingan fungsi explain rencana kueri di SQL Server, Babelfish, dan PostgreSQL.

SQL Server	Babelfish	PostgreSQL
SHOWPLAN_ALL	BABELFISH_SHOWPLAN_ALL	EXPLAIN
STATISTICS PROFILE	BABELFISH_STATISTICS PROFILE	EXPLAIN ANALYZE
Menggunakan pengoptimal SQL Server	Menggunakan pengoptimal PostgreSQL	Menggunakan pengoptimal PostgreSQL
Format input dan output SQL Server	Format input SQL Server dan output PostgreSQL	Format input dan output PostgreSQL
Ditetapkan untuk sesi	Ditetapkan untuk sesi	Berlaku untuk pernyataan tertentu
Mendukung hal berikut: • SELECT • INSERT • UPDATE • DELETE • CURSOR • CREATE • EXECUTE • EXEC dan fungsi, termasuk aliran kontrol (CASE, WHILE-BREAK-CONTINUE, WAITFOR, BEGIN-END, IF-ELSE, dan sebagainya)	Mendukung hal berikut: • SELECT • INSERT • UPDATE • DELETE • CREATE • EXECUTE • EXEC • RAISEERROR • THROW • PRINT • USE	Mendukung hal berikut: • SELECT • INSERT • UPDATE • DELETE • CURSOR • CREATE • EXECUTE

Gunakan fungsi Babelfish sebagai berikut:

- SET BABELFISH_SHOWPLAN_ALL [ON|OFF] – Tetapkan ke ON untuk menghasilkan perkiraan rencana eksekusi kueri. Fungsi ini mengimplementasikan perilaku perintah `EXPLAIN` PostgreSQL. Gunakan perintah ini untuk mendapatkan rencana explain untuk kueri tertentu.
- SET BABELFISH_STATISTICS PROFILE [ON|OFF] – Tetapkan ke ON untuk rencana eksekusi kueri aktual. Fungsi ini mengimplementasikan perilaku perintah `EXPLAIN ANALYZE` PostgreSQL.

Untuk informasi selengkapnya tentang `EXPLAIN` dan `EXPLAIN ANALYZE` PostgreSQL, lihat EXPLAIN di dokumentasi PostgreSQL.

> **Note**
>
> Mulai versi 2.2.0, Anda dapat menetapkan parameter `escape_hatch_showplan_all` ke ignore untuk menghindari penggunaan awalan BABELFISH_ dalam sintaks SQL Server untuk perintah SET `SHOWPLAN_ALL` dan `STATISTICS PROFILE`.

Misalnya, urutan perintah berikut mengaktifkan perencanaan kueri, lalu mengembalikan perkiraan rencana eksekusi kueri untuk pernyataan SELECT tanpa menjalankan kueri. Contoh ini menggunakan contoh basis data `northwind` SQL Server menggunakan alat baris perintah `sqlcmd` untuk mengkueri port TDS:

```
1> SET BABELFISH_SHOWPLAN_ALL ON
2> GO
1> SELECT t.territoryid, e.employeeid FROM
2> dbo.employeeterritories e, dbo.territories t
3> WHERE e.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO

QUERY PLAN


------------------------------------------------------------------------------


Query Text: SELECT t.territoryid, e.employeeid FROM
dbo.employeeterritories e, dbo.territories t
WHERE e.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=6231.74..6399.22 rows=66992 width=10)
  Sort Key: t.territoryid NULLS FIRST
  ->  Nested Loop  (cost=0.00..861.76 rows=66992 width=10)
```

```
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1264 width=4)
              Filter: ((territoryid)::"varchar" IS NOT NULL)
        ->  Materialize  (cost=0.00..1.79 rows=53 width=6)
              ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

Ketika Anda selesai meninjau dan menyesuaikan kueri, Anda dapat menonaktifkan fungsi seperti yang ditunjukkan berikut:

```
1> SET BABELFISH_SHOWPLAN_ALL OFF
```

Dengan menetapkan BABELFISH_STATISTICS PROFILE ke ON, setiap kueri yang dieksekusi akan mengembalikan set hasil regulernya, yang diikuti oleh kumpulan hasil tambahan yang menunjukkan rencana eksekusi kueri aktual. Babelfish menghasilkan rencana kueri yang menyediakan set hasil tercepat saat menginvokasi pernyataan SELECT.

```
1> SET BABELFISH_STATISTICS PROFILE ON
1>
2> GO
1> SELECT e.employeeid, t.territoryid FROM
2> dbo.employeeterritories e, dbo.territories t
3> WHERE t.territoryid=e.territoryid ORDER BY t.territoryid;
4> GO
```

Set hasil dan rencana kueri dikembalikan (contoh ini hanya menunjukkan rencana kueri).

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT e.employeeid, t.territoryid FROM
dbo.employeeterritories e, dbo.territories t
WHERE t.territoryid=e.territoryid ORDER BY t.territoryid
Sort  (cost=42.44..43.28 rows=337 width=10)
  Sort Key: t.territoryid NULLS FIRST


  ->  Hash Join  (cost=2.19..28.29 rows=337 width=10)
       Hash Cond: ((e.territoryid)::"varchar" = (t.territoryid)::"varchar")
        ->  Seq Scan on employeeterritories e  (cost=0.00..22.70 rows=1270 width=36)
        ->  Hash  (cost=1.53..1.53 rows=53 width=6)
             ->  Seq Scan on territories t  (cost=0.00..1.53 rows=53 width=6)
```

Untuk mempelajari cara menganalisis kueri dan hasil yang dikembalikan oleh pengoptimal PostgreSQL, lihat explain.depesz.com. Untuk informasi lebih lanjut tentang EXPLAIN dan EXPLAIN ANALYZE PostgreSQL, lihat EXPLAIN dalam dokumentasi PostgreSQL.

Parameter yang mengontrol opsi explain Babelfish

Anda dapat menggunakan parameter yang ditunjukkan dalam tabel berikut untuk mengontrol jenis informasi yang ditampilkan oleh rencana kueri Anda.

Parameter	Deskripsi
babelfishpg_tsql.explain_buffers	Boolean yang mengaktifkan (dan menonakti fkan) informasi penggunaan buffer untuk pengoptimal. (Default: off) (Diizinkan: off, on)
babelfishpg_tsql.explain_costs	Boolean yang mengaktifkan (dan menonakti fkan) perkiraan startup dan informasi total biaya untuk pengoptimal. (Default: on) (Diizinkan: off, on)
babelfishpg_tsql.explain_format	Menentukan format output untuk rencana `EXPLAIN`. (Default: text) (Diizinkan: text, xml/ json, yaml)
babelfishpg_tsql.explain_settings	Boolean yang mengaktifkan (atau menonakti fkan) penyertaan informasi tentang parameter konfigurasi dalam output rencana EXPLAIN. (Default: off) (Diizinkan: off, on)
babelfishpg_tsql.explain_summary	Boolean yang mengaktifkan (atau menonakti fkan) informasi ringkasan seperti total waktu setelah rencana kueri. (Default: on) (Diizinkan: off, on)
babelfishpg_tsql.explain_timing	Boolean yang mengaktifkan (atau menonakti fkan) waktu startup aktual dan waktu yang dihabiskan di setiap simpul dalam output. (Default: on) (Diizinkan: off, on)

Parameter	Deskripsi
babelfishpg_tsql.explain_verbose	Boolean yang mengaktifkan (atau menonaktifkan) versi rencana explain paling detail. (Default: off) (Diizinkan: off, on)
babelfishpg_tsql.explain_wal	Boolean yang mengaktifkan (atau menonaktifkan) pembuatan informasi catatan WAL sebagai bagian dari rencana explain. (Default: off) (Diizinkan: off, on)

Anda dapat memeriksa nilai parameter terkait Babelfish pada sistem Anda dengan menggunakan klien PostgreSQL atau klien SQL Server. Jalankan perintah berikut untuk mendapatkan nilai parameter Anda saat ini:

```
1> execute sp_babelfish_configure '%explain%';
2> GO
```

Di output berikut, Anda dapat melihat bahwa semua pengaturan di klaster Babelfish DB khusus ini berada pada nilai defaultnya. Tidak semua output ditampilkan dalam contoh ini.

```
            name                    setting                     short_desc
--------------------------------- --------
 ---------------------------------------------------------
babelfishpg_tsql.explain_buffers   off      Include information on buffer usage
babelfishpg_tsql.explain_costs     on       Include information on estimated startup
 and total cost
babelfishpg_tsql.explain_format    text     Specify the output format, which can be
 TEXT, XML, JSON, or YAML
babelfishpg_tsql.explain_settings  off      Include information on configuration
 parameters
babelfishpg_tsql.explain_summary   on       Include summary information (e.g.,totaled
 timing information) after the query plan
babelfishpg_tsql.explain_timing    on       Include actual startup time and time spent
 in each node in the output
babelfishpg_tsql.explain_verbose   off      Display additional information regarding
 the plan
babelfishpg_tsql.explain_wal       off      Include information on WAL record
 generation
```

```
(8 rows affected)
```

Anda dapat mengubah pengaturan untuk parameter ini menggunakan `sp_babelfish_configure`, seperti yang ditunjukkan pada contoh berikut.

```
1> execute sp_babelfish_configure 'explain_verbose', 'on';
2> GO
```

Jika Anda ingin membuat pengaturan permanen di tingkat seluruh klaster, sertakan kata kunci server, seperti yang ditunjukkan di contoh berikut.

```
1> execute sp_babelfish_configure 'explain_verbose', 'on', 'server';
2> GO
```

Menggunakan petunjuk kueri T-SQL untuk meningkatkan kinerja kueri Babelfish

Dimulai dengan versi 2.3.0, Babelfish mendukung penggunaan petunjuk kueri. menggunakan `pg_hint_plan`. Di Aurora PostgreSQL, `pg_hint_plan` diinstal secara default. Untuk informasi selengkapnya tentang ekstensi PostgreSQL `pg_hint_plan`, lihat https://github.com/ossc-db/pg_hint_plan. Untuk detail tentang versi ekstensi ini yang didukung oleh Aurora PostgreSQL, lihat Versi ekstensi untuk Amazon Aurora PostgreSQL di Catatan Rilis untuk Aurora PostgreSQL.

Pengoptimal kueri dirancang dengan baik untuk menemukan rencana eksekusi yang optimal untuk pernyataan SQL. Saat memilih paket, pengoptimal kueri mempertimbangkan model biaya mesin, serta statistik kolom dan tabel. Namun, rencana yang disarankan mungkin tidak memenuhi kebutuhan set data Anda. Dengan demikian, petunjuk kueri mengatasi masalah kinerja untuk meningkatkan rencana eksekusi. `query hint` adalah sintaks yang ditambahkan ke standar SQL yang menginstruksikan mesin basis data tentang cara mengeksekusi kueri. Misalnya, petunjuk dapat menginstruksikan mesin untuk mengikuti pemindaian berurutan dan mengganti rencana apa pun yang telah dipilih oleh pengoptimal kueri.

Mengaktifkan petunjuk kueri T-SQL di Babelfish

Saat ini, Babelfish mengabaikan semua petunjuk T-SQL secara default. Untuk menerapkan petunjuk T-SQL, jalankan perintah `sp_babelfish_configure` dengan nilai enable_pg_hint dalam kondisi ON.

```
EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on' [, 'server']
```

Anda dapat membuat pengaturan aktif secara permanen pada tingkat klaster dengan memasukkan kata kunci server. Untuk mengonfigurasi pengaturan hanya untuk sesi saat ini, jangan gunakan server.

Setelah `enable_pg_hint` ON, Babelfish menerapkan petunjuk T-SQL berikut.

- Petunjuk INDEX
- Petunjuk JOIN
- Petunjuk FORCE ORDER
- Petunjuk MAXDOP

Misalnya, urutan perintah berikut mengaktifkan `pg_hint_plan`.

```
1> CREATE TABLE t1 (a1 INT PRIMARY KEY, b1 INT);
2> CREATE TABLE t2 (a2 INT PRIMARY KEY, b2 INT);
3> GO
1> EXECUTE sp_babelfish_configure 'enable_pg_hint', 'on';
2> GO
1> SET BABELFISH_SHOWPLAN_ALL ON;
2> GO
1> SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2; --NO HINTS (HASH JOIN)
2> GO
```

Tidak ada petunjuk yang diterapkan pada pernyataan SELECT. Rencana kueri tanpa petunjuk dikembalikan.

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT * FROM t1 JOIN t2 ON t1.a1 = t2.a2
Hash Join (cost=60.85..99.39 rows=2260 width=16)
 Hash Cond: (t1.a1 = t2.a2)
 -> Seq Scan on t1 (cost=0.00..32.60 rows=2260 width=8)
 -> Hash (cost=32.60..32.60 rows=2260 width=8)
 -> Seq Scan on t2 (cost=0.00..32.60 rows=2260 width=8)
```

```
1> SELECT * FROM t1 INNER MERGE JOIN t2 ON t1.a1 = t2.a2;
2> GO
```

Tidak ada petunjuk yang diterapkan pada pernyataan SELECT. Output berikut menunjukkan bahwa rencana kueri dengan penggabungan yang digabung dikembalikan.

```
QUERY PLAN


------------------------------------------------------------------------------
Query Text: SELECT/*+ MergeJoin(t1 t2) Leading(t1 t2)*/ * FROM t1 INNER JOIN t2 ON
 t1.a1 = t2.a2
Merge Join (cost=0.31..190.01 rows=2260 width=16)
 Merge Cond: (t1.a1 = t2.a2)
 -> Index Scan using t1_pkey on t1 (cost=0.15..78.06 rows=2260 width=8)
 -> Index Scan using t2_pkey on t2 (cost=0.15..78.06 rows=2260 width=8)
```

```
1> SET BABELFISH_SHOWPLAN_ALL OFF;
2> GO
```

Batasan

Saat menggunakan petunjuk kueri, pertimbangkan batasan berikut:

- Jika rencana kueri di-cache sebelum `enable_pg_hint` diaktifkan, petunjuk tidak akan diterapkan di sesi yang sama. Ini akan diterapkan di sesi baru.
- Jika nama skema diberikan secara eksplisit, petunjuk tidak dapat diterapkan. Anda dapat menggunakan alias tabel sebagai solusi.
- Petunjuk kueri tidak dapat diterapkan ke tampilan dan sub-kueri.
- Petunjuk tidak berfungsi untuk pernyataan UPDATE/DELETE dengan JOIN.
- Petunjuk indeks untuk indeks atau tabel yang tidak ada diabaikan.
- Petunjuk FORCE ORDER tidak berfungsi untuk HASH JOIN dan yang bukan ANSI JOIN.

Menggunakan ekstensi Aurora PostgreSQL dengan Babelfish

Aurora PostgreSQL menyediakan ekstensi untuk bekerja dengan layanan AWS lain. Ini adalah ekstensi opsional yang mendukung berbagai kasus penggunaan, seperti menggunakan Amazon S3 dengan klaster DB Anda untuk mengimpor atau mengekspor data.

- Untuk mengimpor data dari bucket Amazon S3 ke klaster DB Babelfish, siapkan ekstensi Aurora PostgreSQL `aws_s3`. Ekstensi ini juga memungkinkan Anda mengekspor data dari klaster DB Aurora PostgreSQL ke bucket Amazon S3.
- AWS Lambda adalah layanan komputasi yang memungkinkan Anda menjalankan kode tanpa perlu menyediakan atau mengelola server. Anda dapat menggunakan fungsi Lambda untuk melakukan hal-hal seperti memproses notifikasi peristiwa dari instans DB Anda. Untuk mempelajari lebih lanjut tentang Lambda, lihat Apa itu AWS Lambda? di Panduan Pengembang AWS Lambda. Untuk menginvokasi fungsi Lambda dari klaster DB Babelfish Anda, siapkan ekstensi Aurora PostgreSQL `aws_lambda`.

Untuk menyiapkan ekstensi ini untuk klaster Babelfish Anda, Anda harus terlebih dahulu memberikan izin kepada pengguna internal Babelfish untuk memuat ekstensi. Setelah memberikan izin, Anda kemudian dapat memuat ekstensi Aurora PostgreSQL.

Mengaktifkan ekstensi Aurora PostgreSQL di klaster DB Babelfish Anda

Sebelum Anda dapat memuat ekstensi `aws_s3` atau `aws_lambda`, Anda memberikan hak istimewa yang diperlukan untuk klaster DB Babelfish Anda.

Prosedur berikut menggunakan alat baris perintah PostgreSQL `psql` untuk terhubung ke klaster DB. Untuk informasi selengkapnya, lihat Menggunakan psql untuk terhubung ke klaster DB. Anda juga dapat menggunakan pgAdmin. Lihat perinciannya di Menggunakan pgAdmin untuk terhubung ke klaster DB.

Prosedur ini memuat keduanya `aws_s3` dan `aws_lambda`, satu demi satu. Anda tidak perlu memuat keduanya jika hanya ingin menggunakan salah satu dari ekstensi ini. Ekstensi `aws_commons` diperlukan oleh masing-masing, dan itu dimuat secara default seperti yang ditunjukkan pada output.

Untuk mengatur klaster DB Babelfish Anda dengan hak istimewa untuk ekstensi Aurora PostgreSQL

1. Hubungkan ke klaster DB Babelfish Anda. Gunakan nama untuk pengguna “master” (-U) yang Anda tentukan saat Anda membuat klaster DB Babelfish. Default (`postgres`) ditampilkan dalam contoh.

Untuk Linux, macOS, atau Unix:

```
psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com \
-U postgres \
-d babelfish_db \
-p 5432
```

Untuk Windows:

```
psql -h your-Babelfish.cluster.444455556666-us-east-1.rds.amazonaws.com ^
-U postgres ^
-d babelfish_db ^
-p 5432
```

Perintah merespons dengan prompt untuk memasukkan kata sandi untuk nama pengguna (-U).

```
Password:
```

Anda perlu memasukkan kata sandi untuk nama pengguna (-U) untuk klaster DB. Ketika Anda berhasil terhubung, Anda melihat output yang serupa dengan berikut ini.

```
psql (13.4)
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,
 compression: off)
Type "help" for help.

postgres=>
```

2. Berikan hak istimewa kepada pengguna Babelfish internal untuk membuat dan memuat ekstensi.

```
babelfish_db=> GRANT rds_superuser TO master_dbo;
GRANT ROLE
```

3. Buat dan muat ekstensi `aws_s3`. Ekstensi `aws_commons` diperlukan, dan diinstal secara otomatis ketika `aws_s3` diinstal.

```
babelfish_db=> create extension aws_s3 cascade;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

4. Buat dan muat ekstensi `aws_lambda`.

```
babelfish_db=> create extension aws_lambda cascade;
CREATE EXTENSION
babelfish_db=>
```

Menggunakan Babelfish dengan Amazon S3

Jika Anda belum memiliki bucket Amazon S3 untuk digunakan dengan klaster DB Babelfish Anda, Anda dapat membuatnya. Untuk bucket Amazon S3 yang ingin Anda gunakan, Anda menyediakan akses.

Sebelum mencoba mengimpor atau mengekspor data menggunakan bucket Amazon S3, selesaikan langkah-langkah satu kali berikut.

Cara menyiapkan akses instans DB Babelfish ke bucket Amazon S3

1. Buat bucket Amazon S3 untuk instans Babelfish Anda, jika diperlukan. Untuk melakukannya, ikuti petunjuk di Buat bucket di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.
2. Unggah file ke bucket Amazon S3. Untuk melakukannya, ikuti langkah-langkah di Tambahkan objek ke bucket di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.
3. Siapkan izin sesuai kebutuhan:
   - Untuk mengimpor data dari Amazon S3, klaster DB Babelfish memerlukan izin untuk mengakses bucket. Sebaiknya gunakan peran (IAM) AWS Identity and Access Management dan lampirkan kebijakan IAM ke peran tersebut untuk klaster Anda. Untuk melakukannya, ikuti langkah yang ada di Menggunakan peran IAM untuk mengakses bucket Amazon S3 .
   - Untuk mengekspor data dari klaster DB Babelfish, klaster Anda harus diberikan akses ke bucket Amazon S3. Seperti halnya mengimpor, sebaiknya gunakan kebijakan dan peran IAM. Untuk melakukannya, ikuti langkah yang ada di Menyiapkan akses ke bucket Amazon S3.

Sekarang Anda dapat menggunakan Amazon S3 dengan ekstensi `aws_s3` dengan klaster DB Babelfish Anda.

Untuk mengimpor data dari Amazon S3 ke Babelfish dan untuk mengekspor data Babelfish ke Amazon S3

1. Gunakan ekstensi `aws_s3` dengan klaster DB Babelfish Anda.

Ketika Anda melakukannya, pastikan untuk mereferensikan tabel seperti yang ada dalam konteks PostgreSQL. Artinya, jika Anda ingin mengimpor ke dalam tabel Babelfish bernama `[database].[schema].[tableA]`, lihat tabel itu seperti `database_schema_tableA` dalam fungsi `aws_s3`:

- Untuk contoh penggunaan fungsi `aws_s3` untuk mengimpor data, lihat Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL.
- Untuk contoh penggunaan fungsi `aws_s3` untuk mengekspor data, lihat Mengekspor data kueri menggunakan fungsi aws_s3.query_export_to_s3.

2. Pastikan untuk mereferensikan tabel Babelfish menggunakan penamaan PostgreSQL saat menggunakan ekstensi `aws_s3` dan Amazon S3, seperti yang ditunjukkan pada tabel berikut.

Tabel Babelfish	Tabel Aurora PostgreSQL
database.*schema*.*table*	*database*_*schema*_*table*

Untuk mempelajari selengkapnya tentang menggunakan Amazon S3 dengan Aurora PostgreSQL, lihat Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL dan Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3.

Menggunakan Babelfish dengan AWS Lambda

Setelah ekstensi `aws_lambda` dimuat di klaster DB Babelfish Anda tetapi sebelum Anda dapat menginvokasi fungsi Lambda, beri Lambda akses ke klaster DB Anda dengan mengikuti prosedur ini.

Untuk mengatur akses klaster DB Babelfish Anda agar berfungsi dengan Lambda

Prosedur ini menggunakan AWS CLI untuk membuat peran dan kebijakan IAM, dan mengaitkannya dengan klaster DB Babelfish.

1. Buat kebijakan IAM yang memungkinkan akses ke Lambda dari klaster DB Babelfish Anda.

```
aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "AllowAccessToExampleFunction",
```

```
        "Effect": "Allow",
        "Action": "lambda:InvokeFunction",
        "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
        }
    ]
}'
```

2. Buat peran IAM yang dapat diambil oleh kebijakan saat runtime.

```
aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. Lampirkan kebijakan pada peran tersebut.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. Melampirkan peran ke klaster DB Babelfish Anda

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

Setelah Anda menyelesaikan tugas-tugas ini, Anda dapat menginvokasi fungsi Lambda Anda. Untuk informasi lebih lanjut dan contoh pengaturan AWS Lambda untuk klaster DB Aurora PostgreSQL dengan AWS Lambda, lihat AWS Lambda.

Cara menginvokasi fungsi Lambda dari klaster DB Babelfish Anda

AWS Lambda mendukung fungsi yang ditulis dalam Java, Node.js, Python, Ruby, dan bahasa pemrograman lainnya. Jika fungsi menampilkan teks saat diinvokasi, Anda dapat memanggilnya dari klaster DB Babelfish Anda. Contoh berikut adalah fungsi python placeholder yang menampilkan salam.

```
lambda_function.py
import json
def lambda_handler(event, context):
    #TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
```

Saat ini, Babelfish tidak mendukung JSON. Jika fungsi Anda menampilkan JSON, gunakan pembungkus untuk menangani JSON. Misalnya, `lambda_function.py` yang ditunjukkan sebelumnya disimpan di Lambda sebagai `my-function`.

1. Hubungkan ke klaster DB Babelfish Anda menggunakan klien `psql` (atau klien pgAdmin). Untuk informasi selengkapnya, lihat Menggunakan psql untuk terhubung ke klaster DB.
2. Buat pembungkusnya. Contoh ini menggunakan bahasa prosedural PostgreSQL untuk SQL, PL/pgSQL. Untuk mempelajari selengkapnya, lihat PL/pgSQL–Bahasa Prosedural SQL.

   ```
   create or replace function master_dbo.lambda_wrapper()
   returns text
   language plpgsql
   as
   $$
   declare
      r_status_code integer;
      r_payload text;
   begin
      SELECT payload INTO r_payload
        FROM aws_lambda.invoke(  aws_commons.create_lambda_function_arn('my-function',
    'us-east-1')
                               ,'{"body": "Hello from Postgres!"}'::json );
      return r_payload ;
   end;
   $$;
   ```

Fungsi ini sekarang dapat dijalankan dari port TDS Babelfish (1433) atau dari port PostgreSQL (5433).

a. Untuk menginvokasi (memanggil) fungsi ini dari port PostgreSQL Anda:

```
SELECT * from aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function', 'us-east-1'), '{"body": "Hello from Postgres!"}'::json );
```

Output Anda akan serupa dengan yang berikut ini.

```
status_code |                        payload                         |
 executed_version | log_result
-------------+--------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
         |
(1 row)
```

b. Untuk menginvokasi (memanggil) fungsi ini dari port TDS, sambungkan ke port menggunakan klien baris perintah `sqlcmd` SQL Server. Lihat perinciannya di Menggunakan klien SQL Server untuk terhubung ke klaster DB Anda. Saat terhubung, jalankan yang berikut ini:

```
1> select lambda_wrapper();
2> go
```

Perintah tersebut menghasilkan output yang serupa dengan berikut ini:

```
{"statusCode": 200, "body": "\"Hello from Lambda!\""}
```

Untuk mempelajari selengkapnya tentang menggunakan Lambda dengan Aurora PostgreSQL, lihat . Untuk informasi selengkapnya tentang bekerja dengan fungsi Lambda, lihat Memulai dengan Lambda di Panduan Pengembang AWS Lambda.

Menggunakan pg_stat_statements di Babelfish

Babelfish for Aurora PostgreSQL mendukung ekstensi `pg_stat_statements` dari 3.3.0. Untuk mempelajari lebih lanjut, lihat pg_stat_statements.

Untuk detail tentang versi ekstensi ini didukung oleh Aurora PostgreSQL, lihat Versi ekstensi.

Membuat ekstensi pg_stat_statements

Untuk mengaktifkan `pg_stat_statements`, Anda harus mengaktifkan perhitungan pengidentifikasi Kueri. Hal ini dilakukan secara otomatis jika `compute_query_id` diatur ke on atau `auto` di grup parameter. Nilai default untuk parameter `compute_query_id` adalah `auto`. Anda juga perlu membuat ekstensi ini untuk mengaktifkan fitur ini. Gunakan perintah berikut untuk menginstal ekstensi dari titik akhir T-SQL:

```
1>EXEC sp_execute_postgresql 'CREATE EXTENSION pg_stat_statements WITH SCHEMA sys';
```

Anda dapat mengakses statistik kueri menggunakan kueri berikut:

```
postgres=>select * from pg_stat_statements;
```

> **Note**
>
> Selama instalasi, jika Anda tidak memberikan nama skema untuk ekstensi, maka secara default akan membuatnya dalam skema publik. Untuk mengaksesnya, Anda harus menggunakan tanda kurung siku dengan kualifikasi skema seperti yang ditunjukkan di bawah ini:

```
postgres=>select * from [public].pg_stat_statements;
```

Anda juga dapat membuat ekstensi dari titik akhir PSQL.

Mengotorisasi ekstensi

Secara default, Anda dapat melihat statistik untuk kueri yang dilakukan dalam basis data T-SQL Anda tanpa memerlukan otorisasi apa pun.

Untuk mengakses statistik kueri yang dibuat oleh orang lain, Anda harus memiliki peran PostgreSQL `pg_read_all_stats`. Ikuti langkah-langkah yang disebutkan di bawah ini untuk membuat perintah GRANT pg_read_all_stats.

1. Di T-SQL, gunakan kueri berikut yang menampilkan nama peran PG internal.

```
SELECT rolname FROM pg_roles WHERE oid = USER_ID();
```

2. Hubungkan ke basis data Babelfish for Aurora PostgreSQL dengan hak istimewa rds_superuser dan gunakan perintah berikut:

```
GRANT pg_read_all_stats TO <rolname_from_above_query>
```

Contoh

Dari titik akhir T-SQL:

```
1>SELECT rolname FROM pg_roles WHERE oid = USER_ID();
2>go
```

```

rolname
-------
master_dbo
(1 rows affected)
```

Dari titik akhir PSQL:

```
babelfish_db=# grant pg_read_all_stats to master_dbo;
```

```
GRANT ROLE
```

Anda dapat mengakses statistik kueri menggunakan tampilan pg_stat_statements:

```
1>create table t1(cola int);
2>go
1>insert into t1 values (1),(2),(3);
2>go
```

```
(3 rows affected)
```

```
1>select userid, dbid, queryid, query from pg_stat_statements;
2>go
```

```
userid dbid queryid             query
------ ---- -------             -----
37503 34582 6487973085327558478 select * from t1
37503 34582 6284378402749466286 SET QUOTED_IDENTIFIER OFF
37503 34582 2864302298511657420 insert into t1 values ($1),($2),($3)
10    34582 NULL                <insufficient privilege>
37503 34582 5615368793313871642 SET TEXTSIZE 4096
37503 34582 639400815330803392  create table t1(cola int)
(6 rows affected)
```

Mengatur ulang statistik kueri

Anda dapat menggunakan `pg_stat_statements_reset()` untuk mengatur ulang statistik yang dikumpulkan sejauh ini oleh pg_stat_statements. Untuk mempelajari lebih lanjut, lihat pg_stat_statements. Saat ini didukung melalui titik akhir PSQL saja. Hubungkan ke Babelfish for Aurora PostgreSQL dengan hak istimewa `rds_superuser`, gunakan perintah berikut:

```
SELECT pg_stat_statements_reset();
```

Batasan

- Saat ini, `pg_stat_statements()` tidak didukung melalui titik akhir T-SQL. Tampilan `pg_stat_statements` adalah cara yang disarankan untuk mengumpulkan statistik.

- Beberapa kueri mungkin ditulis ulang oleh pengurai T-SQL yang diimplementasikan oleh mesin Aurora PostgreSQL, tampilan `pg_stat_statements` akan menampilkan kueri yang ditulis ulang dan bukan kueri asli.

  Contoh

  ```
  select next value for [dbo].[newCounter];
  ```

  Kueri di atas ditulis ulang sebagai berikut dalam tampilan pg_stat_statements.

  ```
  select nextval($1);
  ```

- Berdasarkan alur eksekusi pernyataan, beberapa kueri mungkin tidak dilacak oleh pg_stat_statements dan tidak akan terlihat dalam tampilan. Ini termasuk pernyataan berikut: `use dbname`, `goto`, `print`, `raise error`, `set`, `throw`, `declare cursor`.
- Untuk pernyataan CREATE LOGIN dan ALTER LOGIN, query dan queryid tidak akan ditampilkan. Ini akan menunjukkan hak istimewa yang tidak memadai.
- Tampilan `pg_stat_statements` selalu berisi dua entri di bawah ini, karena ini dijalankan secara internal oleh klien `sqlcmd`.
  - SET QUOTED_IDENTIFIER OFF
  - SET TEXTSIZE 4096

Babelfish mendukung server tertaut

Babelfish for Aurora PostgreSQL mendukung server tertaut dengan menggunakan ekstensi `tds_fdw` PostgreSQL di versi 3.1.0. Untuk bekerja dengan server yang terhubung, Anda harus menginstal ekstensi `tds_fdw`. Untuk informasi selengkapnya tentang ekstensi `tds_fdw`, lihat Bekerja dengan pembungkus data asing yang didukung untuk Amazon Aurora PostgreSQL.

Menginstal ekstensi **`tds_fdw`**

Anda dapat menginstal ekstensi `tds_fdw` menggunakan metode berikut ini.

Menggunakan CREATE EXTENSION dari titik akhir PostgreSQL

1. Hubungkan ke instans DB PostgreSQL Anda pada basis data Babelfish di port PostgreSQL. Gunakan akun yang memiliki peran rds_superuser.

```
psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
username=test --dbname=babelfish_db --password
```

2. Instal ekstensi `tds_fdw`. Ini adalah proses instalasi satu kali. Anda tidak perlu menginstal ulang saat klaster DB dimulai ulang.

```
babelfish_db=> CREATE EXTENSION tds_fdw;
CREATE EXTENSION
```

Memanggil prosedur tersimpan **`sp_execute_postgresql`** dari titik akhir TDS

Babelfish mendukung instalasi ekstensi `tds_fdw` dengan memanggil prosedur `sp_execute_postgresql` dari versi 3.3.0. Anda dapat menjalankan pernyataan PostgreSQL dari titik akhir T-SQL tanpa keluar dari port T-SQL. Untuk informasi selengkapnya, lihat Referensi prosedur Babelfish for Aurora PostgreSQL

1. Hubungkan ke instans DB PostgreSQL Anda pada basis data Babelfish di port T-SQL.

```
sqlcmd -S your-DB-instance.aws-region.rds.amazonaws.com -U test -P password
```

2. Instal ekstensi `tds_fdw`.

```
1>EXEC sp_execute_postgresql N'CREATE EXTENSION tds_fdw';
2>go
```

Fungsionalitas yang didukung

Babelfish mendukung penambahan RDS for SQL Server jarak jauh atau titik akhir Babelfish for Aurora PostgreSQL sebagai server tertaut. Anda juga dapat menambahkan instans SQL Server jarak jauh lainnya sebagai server tertaut. Kemudian, gunakan `OPENQUERY( )` untuk mengambil data dari server tertaut ini. Mulai dari Babelfish versi 3.2.0, nama empat bagian juga didukung.

Prosedur tersimpan dan tampilan katalog berikut didukung untuk menggunakan server tertaut.

Prosedur tersimpan

- sp_addlinkedserver – Babelfish tidak mendukung parameter `@provstr`.

- sp_addlinkedsrvlogin
  - Anda harus memberikan nama pengguna dan kata sandi jarak jauh eksplisit untuk terhubung ke sumber data jarak jauh. Anda tidak dapat terhubung dengan kredensial mandiri pengguna. Babelfish hanya mendukung `@useself = false`.
  - Babelfish tidak mendukung parameter `@locallogin` karena mengonfigurasi akses server jarak jauh khusus untuk proses masuk lokal tidak didukung.
- sp_linkedservers
- sp_helplinkedsrvlogin
- sp_dropserver
- sp_droplinkedsrvlogin – Babelfish tidak mendukung parameter `@locallogin` karena mengonfigurasi akses server jarak jauh khusus untuk proses masuk lokal tidak didukung.
- sp_serveroption – Babelfish mendukung opsi server berikut:
  - waktu habis kueri (dari Babelfish versi 3.2.0)
  - waktu habis koneksi (dari Babelfish versi 3.3.0)
- sp_testlinkedserver (dari Babelfish versi 3.3.0)
- sp_enum_oledb_providers (dari Babelfish versi 3.3.0)

Tampilan katalog

- sys.servers
- sys.linked_logins

Menggunakan enkripsi dalam transit untuk koneksi

Koneksi dari server Babelfish for Aurora PostgreSQL sumber ke server jarak jauh target menggunakan enkripsi dalam transit (TLS/SSL), tergantung konfigurasi basis data server jarak jauh. Jika server jarak jauh tidak dikonfigurasi untuk enkripsi, server Babelfish yang membuat permintaan ke basis data jarak jauh kembali ke tidak terenkripsi.

Untuk menegakkan enkripsi koneksi

- Jika server tertaut target adalah instans RDS for SQL Server, tetapkan `rds.force_ssl = on` untuk instans SQL Server target. Untuk informasi selengkapnya tentang konfigurasi SSL/TLS untuk RDS for SQL Server, lihat Menggunakan SSL dengan instans DB Microsoft SQL Server

- Jika server tertaut target adalah klaster Babelfish for Aurora PostgreSQL, atur `babelfishpg_tsql.tds_ssl_encrypt = on` dan `ssl = on` untuk server target. Untuk informasi selengkapnya tentang SSL/TLS, lihat Pengaturan SSL Babelfish dan koneksi klien.

Menambahkan Babelfish sebagai server tertaut dari SQL Server

Babelfish for Aurora PostgreSQL dapat ditambahkan sebagai server tertaut dari SQL Server. Pada basis data SQL Server, Anda dapat menambahkan Babelfish sebagai server tertaut menggunakan penyedia Microsoft OLE DB untuk ODBC : MSDASQL.

Ada dua cara untuk mengonfigurasi Babelfish sebagai server tertaut dari SQL Server menggunakan penyedia MSDASQL:

- Menyediakan string koneksi ODBC sebagai string penyedia.
- Sediakan DSN Sistem dari sumber data ODBC selagi menambahkan server tertaut.

Batasan

- OPENQUERY () hanya berfungsi untuk SELECT dan tidak berfungsi untuk DML.
- Nama objek empat bagian hanya berfungsi untuk membaca dan tidak berfungsi untuk memodifikasi tabel jarak jauh. UPDATE dapat mereferensikan tabel jarak jauh dalam klausa FROM tanpa memodifikasinya.
- Menjalankan prosedur tersimpan terhadap server tertaut Babelfish tidak didukung.
- Peningkatan versi mayor Babelfish mungkin tidak berfungsi jika ada objek yang bergantung pada `OPENQUERY()` atau objek yang direferensikan melalui nama empat bagian. Anda harus memastikan bahwa objek apa pun yang mereferensikan `OPENQUERY()` atau nama empat bagian dihapus sebelum peningkatan versi mayor.
- Jenis data berikut tidak berfungsi seperti yang diharapkan terhadap server Babelfish jarak jauh: `nvarchar(max)`, `varchar(max)`, `varbinary(max)`, `binary(max)`, dan `time`. Sebaiknya gunakan fungsi CAST untuk mengonversinya ke tipe data yang didukung.

Contoh

Dalam contoh berikut, instans Babelfish for Aurora PostgreSQL menghubungkan ke instans RDS for SQL Server di cloud.

```
EXEC master.dbo.sp_addlinkedserver @server=N'rds_sqlserver', @srvproduct=N'',
 @provider=N'SQLNCLI', @datasrc=N'myserver.CB2XKFSFFMY7.US-WEST-2.RDS.AMAZONAWS.COM';
EXEC master.dbo.sp_addlinkedsrvlogin
 @rmtsrvname=N'rds_sqlserver',@useself=N'False',@locallogin=NULL,@rmtuser=N'username',@rmtpassw
```

Saat server tertaut sudah terpasang, Anda dapat menggunakan T-SQL OPENQUERY() atau penamaan empat bagian standar untuk mereferensikan tabel, tampilan, atau objek lain yang didukung, di server jarak jauh:

```
SELECT * FROM OPENQUERY(rds_sqlserver, 'SELECT * FROM TestDB.dbo.t1');
SELECT * FROM rds_sqlserver.TestDB.dbo.t1;
```

Untuk menghapus server tertaut dan semua kredensial masuk terkait:

```
EXEC master.dbo.sp_dropserver @server=N'rds_sqlserver', @droplogins=N'droplogins';
```

Pemecahan Masalah

Anda dapat menggunakan grup keamanan yang sama untuk server sumber dan jarak jauh untuk memungkinkan keduanya berkomunikasi satu sama lain. Grup keamanan hanya boleh mengizinkan lalu lintas masuk pada port TDS (1433 secara default) dan IP sumber dalam grup keamanan dapat diatur sebagai ID grup keamanan itu sendiri. Untuk informasi selengkapnya tentang cara mengatur aturan untuk menghubungkan ke satu instans dari instans yang lain dengan grup keamanan yang sama, lihat Aturan untuk menghubungkan ke satu instans dari sebuah instans dengan grup keamanan yang sama.

Jika akses tidak dikonfigurasi dengan benar, pesan kesalahan yang mirip dengan contoh berikut akan muncul saat Anda mencoba menjalankan kueri server jarak jauh.

```
TDS client library error: DB #: 20009, DB Msg: Unable to connect: server is unavailable
 or does not exist (mssql2019.aws-region.rds.amazonaws.com), OS #: 110, OS Msg:
 Connection timed out, Level: 9
```

Pencarian Teks Lengkap di Babelfish

Dimulai dengan versi 4.0.0, Babelfish menyediakan dukungan terbatas untuk Pencarian Teks Lengkap (FTS). FTS adalah fitur yang kuat dalam database relasional yang memungkinkan pencarian dan pengindeksan data teks yang efisien. Ini memungkinkan pengguna untuk melakukan pencarian teks yang kompleks dan mengambil hasil yang relevan dengan cepat. FTS sangat berharga untuk aplikasi yang berhubungan dengan volume besar data tekstual, seperti sistem manajemen konten, platform e-commerce, dan arsip dokumen.

Untuk mengaktifkan Pencarian Teks Lengkap

Jalankan perintah berikut dan atur escape hatch untuk mengaktifkan FTS:

```
1>EXEC sp_babelfish_configure 'babelfishpg_tsql.escape_hatch_fulltext', 'ignore';
2>GO
```

Fitur yang didukung di Babelfish untuk Pencarian Teks Lengkap

- BERISI Klausul Support:
  - Dukungan dasar untuk klausa CONTAINS.

    ```
    CONTAINS (
         {
            column_name
         }
         , '<contains_search_condition>'
    )
    ```

    > **Note**
    >
    > Saat ini, hanya bahasa Inggris yang didukung.

  - Penanganan dan terjemahan string `simple_term` pencarian yang komprehensif.
- `FULLTEXT INDEX`Support Klausul:
  - Hanya mendukung CREATE FULLTEXT INDEX ON table_name (column_name [... n]) pernyataan INDEX_name INDEX KUNCI.

- Mendukung pernyataan DROP FULLTEXT INDEX penuh.

> **Note**
>
> Untuk mengindeks ulang Indeks Teks Lengkap, Anda harus menjatuhkan Indeks Teks Lengkap dan membuat yang baru di kolom yang sama.

Fitur yang tidak didukung di Babelfish untuk Pencarian Teks Lengkap

- Saat ini, opsi berikut tidak didukung di Babelfish for `CONTAINS` Clause.
  - Karakter khusus dan Bahasa selain bahasa Inggris tidak didukung. Anda akan menerima pesan kesalahan umum untuk karakter dan bahasa yang tidak didukung

    ```
    Full-text search conditions with special characters or languages other than English
     are not currently supported in Babelfish
    ```

  - Beberapa kolom seperti `column_list`
  - Atribut PROPERTI
  - `prefix_term`,`generation_term`,`generic_proximity_term`,`custom_proximity_term`, dan `weighted_term`
  - Operator Boolean
- Saat ini, opsi berikut tidak didukung di Babelfish for `CREATE FULLTEXT INDEX` Clause.
  - [JENIS KOLOM type_column_name]
  - [LANGUAGE LANGUAGE_TERM]
  - [STATISTIK_SEMANTIK]
  - opsi filegroup katalog
  - dengan opsi
- Membuat katalog teks lengkap tidak didukung. Membuat indeks teks lengkap tidak memerlukan katalog teks lengkap.

Memecahkan Masalah Babelfish

Di bawah ini adalah solusi dan ide pemecahan masalah untuk beberapa masalah klaster DB Babelfish.

Topik

- Kegagalan koneksi

Kegagalan koneksi

Penyebab umum kegagalan koneksi ke klaster DB Aurora baru dengan Babelfish adalah sebagai berikut:

- Grup keamanan tidak mengizinkan akses - Jika Anda mengalami masalah saat menyambung ke Babelfish, pastikan Anda menambahkan alamat IP Anda ke grup keamanan Amazon EC2 default. Anda dapat menggunakan https://checkip.amazonaws.com/ untuk menentukan alamat IP Anda lalu menambahkannya ke aturan in-bound Anda untuk port TDS dan port PostgreSQL. Untuk informasi selengkapnya, lihat Menambahkan aturan ke grup keamanan di Panduan Pengguna Amazon EC2.
- Konfigurasi SSL tidak cocok - Jika parameter `rds.force_ssl` diaktifkan (diatur ke 1) pada Aurora PostgreSQL, klien harus terhubung ke Babelfish melalui SSL. Jika klien Anda tidak diatur dengan benar, Anda akan melihat pesan kesalahan seperti berikut ini:

  ```
  Cannot connect to your-Babelfish-DB-cluster, 1433
  ---------------------
  ADDITIONAL INFORMATION:
  no pg_hba_conf entry for host "256.256.256.256", user "your-user-name",
  "database babelfish_db", SSL off (Microsoft SQL Server, Error: 33557097)
  ...
  ```

  Kesalahan ini menunjukkan kemungkinan masalah konfigurasi SSL antara klien lokal Anda dan klaster Babelfish DB, dan klaster mengharuskan klien untuk menggunakan SSL (parameter `rds.force_ssl` disetel ke 1). Untuk informasi selengkapnya tentang mengonfigurasi SSL, lihat Menggunakan SSL dengan instans DB PostgreSQL di Panduan Pengguna Amazon RDS.

  Jika Anda menggunakan SQL Server Management Studio (SSMS) untuk terhubung ke Babelfish dan Anda melihat kesalahan ini, Anda dapat memilih opsi koneksi Enkripsi koneksi dan Percayai sertifikat server pada panel Properti Koneksi dan coba lagi. Pengaturan ini menangani persyaratan koneksi SSL untuk SSMS.

Untuk informasi selengkapnya tentang pemecahan masalah koneksi Aurora, lihat Tidak dapat terhubung ke instans DB Amazon RDS.

Menonaktifkan Babelfish

Saat Anda tidak lagi membutuhkan Babelfish, Anda dapat menonaktifkan fungsionalitas Babelfish.

Ketahui beberapa pertimbangannya:

- Dalam beberapa kasus, Anda mungkin menonaktifkan Babelfish sebelum menyelesaikan migrasi ke Aurora PostgreSQL. Jika Anda melakukannya dan DDL Anda bergantung pada tipe data SQL Server atau Anda menggunakan fungsionalitas T-SQL apa pun dalam kode Anda, kode Anda akan gagal.
- Jika setelah menyediakan instans Babelfish Anda menonaktifkan ekstensi Babelfish, Anda tidak dapat menyediakan basis data yang sama lagi di klaster yang sama.

Untuk menonaktifkan Babelfish, ubah grup parameter Anda, atur `rds.babelfish_status` ke `OFF`. Anda dapat terus menggunakan tipe data SQL Server Anda dalam keadaan Babelfish nonaktif, dengan mengatur `rds.babelfish_status` ke `datatypeonly`.

Jika Anda menonaktifkan Babelfish di grup parameter, semua klaster yang menggunakan grup parameter tersebut kehilangan fungsionalitas Babelfish.

Untuk informasi selengkapnya tentang memodifikasi grup parameter, lihat Bekerja dengan grup parameter. Untuk informasi tentang parameter khusus Babelfish, lihat Pengaturan grup parameter klaster DB untuk Babelfish.

Pembaruan versi Babelfish

Babelfish adalah sebuah opsi yang tersedia dengan Aurora PostgreSQL versi 13.4 dan versi rilis yang lebih tinggi. Pembaruan untuk Babelfish menjadi tersedia dengan versi rilis baru tertentu dari mesin basis data Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Catatan Rilis untuk Aurora PostgreSQL.

> **Note**
>
> Klaster DB Babelfish yang berjalan di Aurora PostgreSQL 13 dalam versi apa pun tidak dapat ditingkatkan ke Aurora PostgreSQL 14.3, 14.4, dan 14.5. Selain itu, Babelfish tidak mendukung peningkatan langsung dari 13.x ke 15.x. Anda harus terlebih dahulu meningkatkan klaster DB 13.x Anda ke 14.6 dan versi yang lebih tinggi lalu meningkatkan ke versi 15.x.

Untuk daftar fungsionalitas yang didukung di berbagai rilis Babelfish, lihat Fungsionalitas yang didukung di Babelfish berdasarkan versi.

Untuk daftar fungsionalitas yang saat ini tidak didukung, lihat Fungsionalitas yang tidak didukung di Babelfish.

Anda dapat menggunakan describe-db-engine-versionsAWS CLIperintah untuk mendapatkan daftar versi Aurora PostgreSQL di Anda yang mendukung Babelfish, seperti Wilayah AWS yang ditunjukkan pada contoh berikut.

Untuk Linux, macOS, atau Unix:

```
$ aws rds describe-db-engine-versions --region us-east-1 \
    --engine aurora-postgresql \
    --query '*[]|[?SupportsBabelfish==`true`].[EngineVersion]' \
    --output text
13.4
13.5
13.6
13.7
13.8
14.3
14.4
14.5
```

```
14.6
14.7
14.8
14.9
14.10
15.2
15.3
15.4
15.5
16.1
```

Untuk informasi selengkapnya, lihat describe-db-engine-versions dalam AWS CLI Referensi Perintah.

Dalam topik berikut, Anda dapat mempelajari cara mengidentifikasi versi Babelfish yang berjalan di klaster DB Aurora PostgreSQL Anda, dan cara meningkatkannya ke versi baru.

Daftar Isi

- Mengidentifikasi versi Babelfish Anda
- Meningkatkan klaster Babelfish Anda ke versi baru
  - Meningkatkan Babelfish ke versi minor baru
  - Meningkatkan Babelfish ke versi mayor baru
    - Sebelum meningkatkan Babelfish ke versi mayor baru
    - Melakukan peningkatan versi mayor
    - Setelah meningkatkan ke versi mayor baru
    - Contoh: Meningkatkan klaster DB Babelfish ke rilis mayor
- Menggunakan parameter versi produk Babelfish
  - Mengonfigurasi parameter versi produk Babelfish
  - Kueri dan parameter yang terpengaruh
  - Antarmuka dengan parameter babelfishpg_tsql.version

Mengidentifikasi versi Babelfish Anda

Anda dapat meminta Babelfish untuk menemukan detail tentang versi Babelfish, versi Aurora PostgreSQL, dan versi Microsoft SQL Server yang kompatibel. Anda dapat menggunakan port TDS atau port PostgreSQL.

- To use the TDS port to query for version information

- To use the PostgreSQL port to query for version information

Untuk menggunakan port TDS untuk meminta informasi versi

1. Gunakan `sqlcmd` atau `ssms` untuk terhubung ke titik akhir untuk klaster DB Babelfish Anda.

```
sqlcmd -S bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
    login-id -P password -d db_name
```

2. Untuk mengidentifikasi versi Babelfish, jalankan kueri berikut ini:

```
1> SELECT CAST(serverproperty('babelfishversion') AS VARCHAR)
2> GO
```

   Kueri ini mengembalikan hasil yang serupa dengan berikut ini:

```
serverproperty
------------------------------
3.4.0

(1 rows affected)
```

3. Untuk mengidentifikasi versi klaster DB Aurora PostgreSQL, jalankan kueri berikut:

```
1> SELECT aurora_version() AS aurora_version
2> GO
```

   Kueri ini mengembalikan hasil yang serupa dengan berikut ini:

```
aurora_version


------------------------------------------------
15.5.0

(1 rows affected)
```

4. Untuk mengidentifikasi versi Microsoft SQL Server yang kompatibel, jalankan kueri berikut:

```
1> SELECT @@VERSION AS version
```

```
2> GO
```

Kueri ini mengembalikan hasil yang serupa dengan berikut ini:

```
Babelfish for Aurora PostgreSQL with SQL Server Compatibility - 12.0.2000.8
Dec 7 2023 09:43:06
Copyright (c) Amazon Web Services
PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)

(1 rows affected)
```

Sebagai contoh yang menunjukkan satu perbedaan kecil antara Babelfish dan Microsoft SQL Server, Anda dapat menjalankan kueri berikut. Di Babelfish, kueri mengembalikan 1, sementara di Microsoft SQL Server, query mengembalikan NULL.

```
SELECT CAST(serverproperty('babelfish') AS VARCHAR) AS runs_on_babelfish
```

Anda juga dapat menggunakan port PostgreSQL untuk memperoleh informasi versi, seperti yang diperlihatkan dalam prosedur berikut ini.

Untuk menggunakan port PostgreSQL untuk meminta informasi versi

1. Gunakan `psql` atau `pgAdmin` untuk terhubung ke titik akhir untuk klaster DB Babelfish Anda.

   ```
   psql host=bfish_db.cluster-123456789012.aws-region.rds.amazonaws.com
        port=5432 dbname=babelfish_db user=sa
   ```

2. Aktifkan fitur tambahan (`\x`) dari `psql` untuk output yang lebih mudah dibaca.

   ```
   babelfish_db=> \x
   babelfish_db=> SELECT
   babelfish_db=> aurora_version() AS aurora_version,
   babelfish_db=> version() AS postgresql_version,
   babelfish_db=> sys.version() AS Babelfish_compatibility,
   babelfish_db=> sys.SERVERPROPERTY('BabelfishVersion') AS Babelfish_Version;
   ```

   Kueri ini menghasilkan output serupa dengan berikut ini:

```
-[ RECORD 1 ]-----------
+-------------------------------------------------------------------------------------------
aurora_version          | 15.5.0
postgresql_version      | PostgreSQL 15.5 on x86_64-pc-linux-gnu, compiled by
 x86_64-pc-linux-gnu-gcc (GCC) 9.5.0, 64-bit
babelfish_compatibility | Babelfish for Aurora Postgres with SQL Server
 Compatibility - 12.0.2000.8                    +
                         | Dec 7 2023 09:43:06
                                    +
                         | Copyright (c) Amazon Web Services
                                     +
                         | PostgreSQL 15.5 on x86_64-pc-linux-gnu (Babelfish 3.4.0)
babelfish_version       | 3.4.0
```

Meningkatkan klaster Babelfish Anda ke versi baru

Versi baru Babelfish tersedia dengan beberapa rilis baru dari mesin basis data Aurora PostgreSQL setelah versi 13.4. Setiap rilis baru Babelfish memiliki nomor versi sendiri. Sama halnya dengan Aurora PostgreSQL, Babelfish menggunakan skema penamaan *major*.*minor*.*patch* untuk versi. Misalnya, rilis Babelfish pertama, Babelfish versi 1.0.0, tersedia sebagai bagian dari Aurora PostgreSQL 13.4.0.

Babelfish tidak memerlukan proses instalasi terpisah. Seperti dibahas dalam Membuat klaster DB Babelfish for Aurora PostgreSQL, Aktifkan Babelfish adalah opsi yang Anda pilih saat membuat klaster DB Aurora PostgreSQL.

Demikian pula, Anda tidak dapat meningkatkan Babelfish secara independen dari klaster Aurora DB yang mendukung. Untuk meningkatkan klaster DB Babelfish for Aurora PostgreSQL yang ada ke versi baru Babelfish, tingkatkan klaster DB Aurora PostgreSQL ke versi baru yang mendukung versi Babelfish yang ingin Anda gunakan. Prosedur yang Anda ikuti untuk peningkatan tergantung versi Aurora PostgreSQL yang mendukung deployment Babelfish Anda, sebagai berikut.

Peningkatan versi mayor

Anda harus meningkatkan versi Aurora PostgreSQL berikut ke Aurora PostgreSQL 14.6 dan versi yang lebih tinggi sebelum meningkatkan ke Aurora PostgreSQL versi 15.2.

- Aurora PostgreSQL 13.8 dan semua versi yang lebih tinggi
- Aurora PostgreSQL 13.7.1 dan semua versi minor yang lebih tinggi
- Aurora PostgreSQL 13.6.4 dan semua versi minor yang lebih tinggi

Anda dapat meningkatkan Aurora PostgreSQL 14.6 dan versi yang lebih tinggi ke Aurora PostgreSQL 15.2 dan versi yang lebih tinggi.

Meningkatkan klaster DB Aurora PostgreSQL ke versi mayor baru melibatkan beberapa tugas awal. Untuk informasi selengkapnya, lihat Cara melakukan peningkatan versi mayor. Agar berhasil meningkatkan klaster DB Babelfish for Aurora PostgreSQL, Anda perlu membuat grup parameter klaster DB khusus untuk versi PostgreSQL Aurora baru. Grup parameter baru ini harus berisi pengaturan parameter Babelfish yang sama dengan klaster yang Anda tingkatkan. Untuk informasi selengkapnya dan untuk tabel sumber dan peningkatan pemutakhiran versi mayor, lihat Meningkatkan Babelfish ke versi mayor baru.

Peningkatan dan patch versi minor

Versi dan patch minor tidak memerlukan pembuatan grup parameter klaster DB baru untuk peningkatan. Versi dan patch minor dapat menggunakan proses peningkatan versi minor, baik itu diterapkan secara otomatis atau manual. Untuk informasi selengkapnya dan tabel sumber dan target versi, lihat Meningkatkan Babelfish ke versi minor baru.

> **Note**
>
> Sebelum melakukan peningkatan mayor atau minor, terapkan semua tugas pemeliharaan yang tertunda ke klaster Babelfish for Aurora PostgreSQL Anda.

Topik

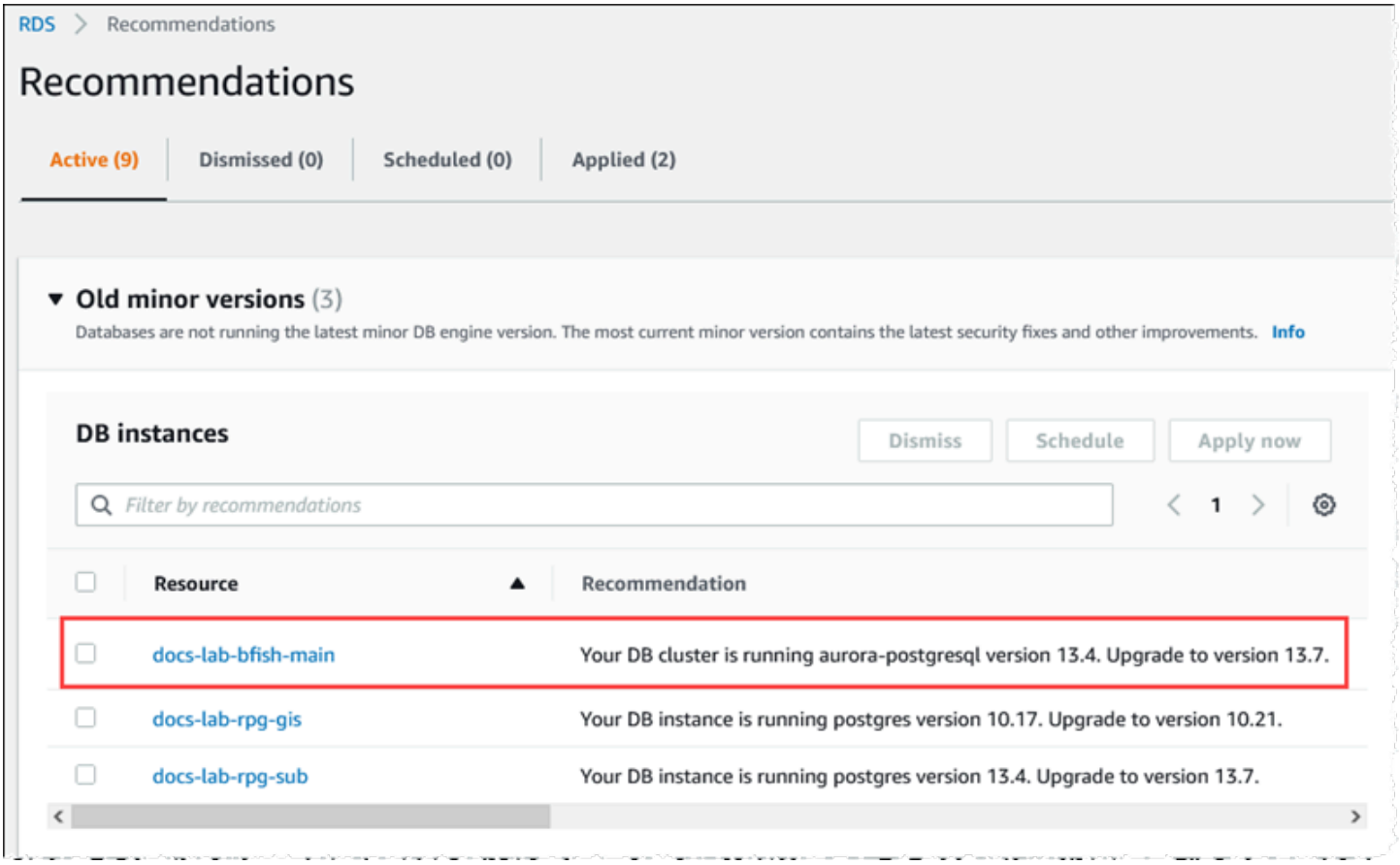
- Meningkatkan Babelfish ke versi minor baru
- Meningkatkan Babelfish ke versi mayor baru

Meningkatkan Babelfish ke versi minor baru

Versi minor baru hanya mencakup perubahan yang kompatibel dengan versi sebelumnya. Versi patch mencakup perbaikan penting untuk versi minor setelah dirilis. Misalnya, label versi untuk rilis pertama Aurora PostgreSQL 13.4 adalah Aurora PostgreSQL 13.4.0. Beberapa patch untuk versi minor telah dirilis hingga saat ini, termasuk Aurora PostgreSQL 13.4.1, 13.4.2, dan 13.4.4. Anda dapat menemukan patch yang tersedia untuk setiap versi Aurora PostgreSQL dalam daftar rilis Patch di bagian atas catatan rilis Aurora PostgreSQL untuk versi tersebut. Untuk mengetahui contohnya, lihat PostgreSQL 14.3 di Catatan Rilis untuk Aurora PostgreSQL.

Jika klaster DB Aurora PostgreSQL Anda dikonfigurasi dengan opsi Peningkatan versi minor otomatis, klaster DB Babelfish for Aurora PostgreSQL Anda ditingkatkan secara otomatis selama jendela pemeliharaan cluster. Untuk mempelajari lebih lanjut tentang peningkatan versi minor otomatis (AMvU) dan cara menggunakannya, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora. Jika klaster Anda tidak menggunakan AmVU, Anda dapat meningkatkan klaster DB Babelfish for Aurora PostgreSQL secara manual ke versi minor baru baik dengan merespons tugas pemeliharaan, atau mengubah klaster untuk menggunakan versi baru.

Saat Anda memilih versi Aurora PostgreSQL yang akan diinstal dan saat Anda melihat klaster DB Aurora PostgreSQL yang ada di AWS Management Console, versi hanya menampilkan digit *major*.*minor*. Misalnya, gambar berikut dari Konsol untuk klaster DB Babelfish for Aurora PostgreSQL dengan Aurora PostgreSQL 13.4 merekomendasikan untuk meningkatkan klaster ke versi 13.7, rilis minor baru dari Aurora PostgreSQL.



Untuk mendapatkan detail versi lengkap, termasuk level *patch*, Anda dapat menjalankan kueri pada klaster DB Aurora PostgreSQL menggunakan fungsi Aurora PostgreSQL `aurora_version`. Untuk informasi selengkapnya, lihat aurora_version di Referensi fungsi Aurora PostgreSQL. Anda dapat

menemukan contoh penggunaan fungsi dalam prosedur To use the PostgreSQL port to query for version information di Mengidentifikasi versi Babelfish Anda.

Tabel berikut menunjukkan versi Aurora PostgreSQL dan Babelfish dan versi target yang tersedia yang dapat mendukung proses peningkatan versi minor.

Versi sumber saat ini		Target peningkatan terbaru		Versi peningkatan lainnya yang tersedia			
Aurora PostgreSQL	Babelfish	Aurora PostgreSQL	Babelfish	Versi Aurora PostgreSQL dengan opsi Babelfish			
15.4	3.3.0	15.5	3.4.0				
15.3.2	3.2.1	15.5	3.4.0	15.4			
15.2.4	3.1.3	15.5	3.4.0	15.4	15.3		
14.9.1	2.6.0	14.10	2.7.0				
14.8.2	2.5.1	14.10	2.7.0	14.9.1			
14.7.4	2.4.3	14.10	2.7.0	14.9.1	14.8.2		
14.6.4	2.3.3	14.10	2.7.0	14.9.1	14.8.2	14.7.4	
14.5.3	2.2.3	14.10	2.7.0	14.9.1	14.8.2	14.7.4	14.6.4
14.3.1	2.1.1	14.6	2.3.0				
14.3.0	2.1.0	14.6	2.3.0	14.3.1			
13.8	1.4.0	13.9	1.5				
13.7.1	1.3.1	13.9	1.5	13.8			
13.7.0	1.3.0	13.9	1.5	13.7.1			
13.6.4	1.2.4	13.9	1.5	13.7			

Versi sumber saat ini		Target peningkatan terbaru		Versi peningkatan lainnya yang tersedia			
13.6.3	1.2.1	13.9	1.5	13.7	13.6.4		
13.6.2	1.2.1	13.9	1.5	13.7	13.6.4		
13.6.1	1.2.0	13.9	1.5	13.7	13.6.4		
13.6.0	1.2.0	13.9	1.5	13.7	13.6.4		
13,5	1.1.0	13.9	1.5	13.7	13.6		
13.4	1.0.0	13.9	1.5	13.7	13.6	13,5	

Meningkatkan Babelfish ke versi mayor baru

Untuk peningkatan versi mayor, Anda harus terlebih dahulu meningkatkan klaster DB Babelfish for Aurora PostgreSQL ke versi yang mendukung peningkatan versi mayor. Untuk mewujudkannya, terapkan pembaruan patch atau peningkatan versi minor ke klaster DB Anda. Untuk informasi selengkapnya, lihat Meningkatkan Babelfish ke versi minor baru.

Tabel berikut menunjukkan versi Aurora PostgreSQL dan versi Babelfish yang dapat mendukung peningkatan versi mayor.

Versi sumber saat ini		Target peningkatan terbaru yang tersedia		Versi lain yang tersedia (peningkatan versi minor)		
Aurora PostgreSQL	Babelfish	Aurora PostgreSQL	Babelfish	Versi Aurora PostgreSQL (versi Babelfish)		
15.5	3.4.0	16.1	4.0.0			
15.4	3.3.0	16.1	4.0.0			
15.3	3.2.0	16.1	4.0.0			
15.2	3.1.0	16.1	4.0.0			

Versi sumber saat ini		Target peningkatan terbaru yang tersedia		Versi lain yang tersedia (peningkatan versi minor)		
14.10	2.7.0	15.5	3.4.0			
14.9	2.6.0	15.5	3.4.0	15.4 (3.3.0)		
14.8	2.5.0	15.5	3.4.0	15.4 (3.3.0)	15.3 (3.2.0)	
14.7	2.4.0	15.5	3.4.0	15.4 (3.3.0)	15.3 (3.2.0)	15.2 (3.1.0)
14.6	2.3.0	15.5	3.4.0	15.4 (3.3.0)	15.3 (3.2.0)	15.2 (3.1.0)
13.9	1.5.0	14.6	2.3.0			
13.8	1.4.0	14.6	2.3.0			
13.7.1	1.3.1	14.6	2.3.0	13.8 (1.4)		
13.6.4	1.2.2	14.6	2.3.0	13.8 (1.4)	13.7 (1.3)	

Sebelum meningkatkan Babelfish ke versi mayor baru

Peningkatan mungkin melibatkan pemadaman singkat. Oleh karena itu, sebaiknya Anda melakukan atau menjadwalkan peningkatan selama periode pemeliharaan atau selama periode penggunaan rendah lainnya.

Sebelum melakukan peningkatan versi mayor

1. Identifikasi versi Babelfish dari klaster DB Aurora PostgreSQL Anda yang ada dengan menggunakan perintah yang diuraikan di Mengidentifikasi versi Babelfish Anda. Versi Aurora PostgreSQL dan informasi versi Babelfish ditangani oleh PostgreSQL, jadi ikuti langkah-langkah terperinci dalam prosedur To use the PostgreSQL port to query for version information untuk mengetahui detailnya.
2. Verifikasi apakah versi Anda mendukung peningkatan versi mayor. Untuk daftar versi yang mendukung fitur peningkatan versi mayor, lihat Meningkatkan Babelfish ke versi minor baru dan lakukan tugas prapeningkatan yang diperlukan.

Misalnya, jika versi Babelfish Anda berjalan pada klaster DB Aurora PostgreSQL 13.5 dan Anda ingin meningkatkan ke Aurora PostgreSQL 15.2, maka pertama-tama terapkan semua rilis dan patch minor untuk meningkatkan klaster Anda ke Aurora PostgreSQL 14.6 atau versi yang lebih tinggi. Saat klaster Anda berada di versi 14.6 atau lebih tinggi, lanjutkan dengan proses peningkatan versi mayor.

3. Buat snapshot manual dari klaster DB Babelfish Anda saat ini sebagai cadangan. Cadangan ini memungkinkan Anda memulihkan klaster ke versi Aurora PostgreSQL, versi Babelfish, dan memulihkan semua data ke status sebelum peningkatan. Untuk informasi selengkapnya, lihat Membuat snapshot klaster DB. Pastikan untuk menyimpan grup parameter klaster DB kustom Anda yang ada untuk menggunakannya lagi jika Anda memutuskan untuk memulihkan klaster ini ke status prapeningkatan. Lihat informasi yang lebih lengkap di Memulihkan dari snapshot klaster DB dan Pertimbangan grup parameter.
4. Siapkan grup parameter klaster DB khusus untuk versi Aurora PostgreSQL target. Duplikasi pengaturan untuk parameter Babelfish dari klaster DB Babelfish for Aurora PostgreSQL Anda saat ini. Untuk menemukan daftar semua parameter Babelfish, lihat Pengaturan grup parameter klaster DB untuk Babelfish. Untuk peningkatan versi mayor, parameter berikut memerlukan pengaturan yang sama dengan klaster DB sumber. Agar peningkatan berhasil, semua pengaturan harus sama.

   - rds.babelfish_status
   - babelfishpg_tds.tds_default_numeric_precision
   - babelfishpg_tds.tds_default_numeric_scale
   - babelfishpg_tsql.database_name
   - babelfishpg_tsql.default_locale
   - babelfishpg_tsql.migration_mode
   - babelfishpg_tsql.server_collation_name

   > **Warning**
   >
   > Jika pengaturan untuk parameter Babelfish dalam grup parameter klaster DB kustom untuk versi PostgreSQL Aurora baru tidak cocok dengan nilai parameter klaster yang Anda tingkatkan, operasi `ModifyDBCluster` gagal. Pesan kesalahan `InvalidParameterCombination` muncul di AWS Management Console atau di output dari perintah AWS CLI `modify-db-cluster`.

5. Gunakan AWS Management Console atau AWS CLI untuk membuat grup parameter klaster DB kustom. Pilih keluarga Aurora PostgreSQL yang berlaku untuk versi Aurora PostgreSQL yang Anda inginkan untuk peningkatan.

   > **Tip**
   >
   > Grup parameter dikelola di level Wilayah AWS tersebut. Ketika Anda bekerja dengan AWS CLI, Anda dapat mengonfigurasi dengan Wilayah default alih-alih menentukan `--region` dalam perintah. Untuk mempelajari selengkapnya tentang menggunakan AWS CLI, lihat Pengaturan cepat dalam Panduan Pengguna AWS Command Line Interface.

Melakukan peningkatan versi mayor

1. Tingkatkan klaster DB Aurora PostgreSQL ke versi mayor baru. Untuk informasi selengkapnya, lihat Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru.
2. Reboot instans penulis klaster, sehingga pengaturan parameter dapat berlaku.

Setelah meningkatkan ke versi mayor baru

Setelah peningkatan versi mayor ke versi Aurora PostgreSQL baru, nilai `IDENTITY` dalam tabel dengan kolom `IDENTITY` mungkin lebih besar (+32) daripada nilai sebelum peningkatan. Hasilnya adalah ketika baris berikutnya dimasukkan ke dalam tabel tersebut, nilai kolom identitas yang dibuat melompat ke angka +32 dan memulai urutan dari sana. Kondisi ini tidak akan berdampak negatif pada fungsi klaster DB Babelfish Anda. Namun, jika mau, Anda dapat mengatur ulang objek urutan berdasarkan nilai maksimum kolom. Untuk melakukannya, sambungkan ke port T-SQL pada instans penulis Babelfish Anda menggunakan `sqlcmd` atau klien SQL Server lainnya. Untuk informasi selengkapnya, lihat Menggunakan klien SQL Server untuk terhubung ke klaster DB Anda.

```
sqlcmd -S bfish-db.cluster-123456789012.aws-region.rds.amazonaws.com,1433 -U
     sa -P ******** -d dbname
```

Saat terhubung, gunakan perintah SQL berikut untuk menghasilkan pernyataan yang dapat Anda gunakan untuk menyemai objek urutan terkait. Perintah SQL ini berfungsi untuk konfigurasi Babelfish basis data tunggal dan beberapa basis data. Untuk informasi selengkapnya tentang kedua model deployment ini, lihat Menggunakan Babelfish dengan satu atau beberapa basis data.

```
DECLARE @schema_prefix NVARCHAR(200) = ''
```

```
IF current_setting('babelfishpg_tsql.migration_mode') = 'multi-db'
    SET @schema_prefix = db_name() + '_'
SELECT 'SELECT setval(pg_get_serial_sequence(''' + @schema_prefix +
 schema_name(tables.schema_id)
    + '.' + tables.name + ''', ''' + columns.name + '''),(select max(' + columns.name +
 ')
    FROM ' + schema_name(tables.schema_id) + '.' + tables.name + '));
    'FROM sys.tables tables JOIN sys.columns
    columns ON tables.object_id = columns.object_id
    WHERE columns.is_identity = 1
GO
```

Kueri menghasilkan serangkaian pernyataan SELECT yang kemudian dapat Anda jalankan untuk mengatur ulang nilai IDENTITY maksimum dan menutup celah apa pun. Berikut ini menunjukkan output saat menggunakan basis data SQL Server sampel, Northwind, berjalan pada klaster Babelfish.

```
------------------------------------------------------------
SELECT setval(pg_get_serial_sequence('northwind_dbo.categories', 'categoryid'),(select
 max(categoryid)
    FROM dbo.categories));

SELECT setval(pg_get_serial_sequence('northwind_dbo.orders', 'orderid'),(select
 max(orderid)
    FROM dbo.orders));


SELECT setval(pg_get_serial_sequence('northwind_dbo.products', 'productid'),(select
 max(productid)
    FROM dbo.products));


SELECT setval(pg_get_serial_sequence('northwind_dbo.shippers', 'shipperid'),(select
 max(shipperid)
    FROM dbo.shippers));


SELECT setval(pg_get_serial_sequence('northwind_dbo.suppliers', 'supplierid'),(select
 max(supplierid)
    FROM dbo.suppliers));


(5 rows affected)
```

Jalankan pernyataan satu per satu untuk mengatur ulang nilai urutan.

Contoh: Meningkatkan klaster DB Babelfish ke rilis mayor

Dalam contoh ini, Anda dapat menemukan serangkaian perintah AWS CLI yang menjelaskan cara meningkatkan klaster DB Aurora PostgreSQL 13.6.4 yang menjalankan Babelfish versi 1.2.2 ke Aurora PostgreSQL 14.6. Pertama, Anda membuat grup parameter klaster DB kustom untuk Aurora PostgreSQL 14. Selanjutnya, Anda memodifikasi nilai parameter agar sesuai dengan sumber Aurora PostgreSQL versi 13 Anda. Terakhir, Anda melakukan peningkatan dengan memodifikasi klaster sumber. Untuk informasi selengkapnya, lihat Pengaturan grup parameter klaster DB untuk Babelfish. Dalam topik itu, Anda juga dapat menemukan informasi tentang menggunakan AWS Management Console untuk melakukan peningkatan.

Gunakan perintah create-db-cluster-parameter-group CLI untuk membuat grup parameter cluster DB untuk versi baru.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster-parameter-group \
    --db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
    --db-parameter-group-family aurora-postgresql14 \
    --description 'New custom parameter group for upgrade to new major version' \
    --region us-west-1
```

Saat Anda mengeluarkan perintah ini, grup parameter klaster DB kustom dibuat di Wilayah AWS. Output Anda akan terlihat seperti berikut ini.

```
{
    "DBClusterParameterGroup": {
        "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14",
        "DBParameterGroupFamily": "aurora-postgresql14",
        "Description": "New custom parameter group for upgrade to new major version",
        "DBClusterParameterGroupArn": "arn:aws:rds:us-west-1:111122223333:cluster-
pg:docs-lab-babelfish-apg-14"
    }
}
```

Untuk informasi selengkapnya, lihat Membuat grup parameter klaster DB.

Gunakan perintah modify-db-cluster-parameter-group CLI untuk memodifikasi pengaturan sehingga cocok dengan cluster sumber.

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group --db-cluster-parameter-group-name docs-lab-
babelfish-apg-14 ^
  --parameters
 "ParameterName=rds.babelfish_status,ParameterValue=on,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_precision,ParameterValue=38,ApplyMethod=pen
reboot" ^
 "ParameterName=babelfishpg_tds.tds_default_numeric_scale,ParameterValue=8,ApplyMethod=pending-
reboot" ^
 "ParameterName=babelfishpg_tsql.database_name,ParameterValue=babelfish_db,ApplyMethod=pending-
reboot" ^
 "ParameterName=babelfishpg_tsql.default_locale,ParameterValue=en-
US,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tsql.migration_mode,ParameterValue=single-
db,ApplyMethod=pending-reboot" ^
 "ParameterName=babelfishpg_tsql.server_collation_name,ParameterValue=sql_latin1_general_cp1_ci
reboot"
```

Responsnya terlihat seperti berikut.

```
{
    "DBClusterParameterGroupName": "docs-lab-babelfish-apg-14"
}
```

Gunakan perintah modify-db-clusterCLI untuk memodifikasi cluster untuk menggunakan versi baru dan grup parameter cluster DB kustom baru. Anda juga menentukan argumen `--allow-major-version-upgrade`, seperti yang ditunjukkan dalam sampel berikut.

```
aws rds modify-db-cluster \
--db-cluster-identifier docs-lab-bfish-apg-14 \
--engine-version 14.6 \
--db-cluster-parameter-group-name docs-lab-babelfish-apg-14 \
--allow-major-version-upgrade \
--region us-west-1 \
--apply-immediately
```

Gunakan perintah reboot-db-instanceCLI untuk me-reboot instance penulis cluster, sehingga pengaturan parameter dapat berlaku.

```
aws rds reboot-db-instance \
--db-instance-identifier docs-lab-bfish-apg-14-instance-1\
--region us-west-1
```

Menggunakan parameter versi produk Babelfish

Parameter Grand Unified Configuration (GUC) baru yang disebut `babelfishpg_tds.product_version` diperkenalkan dari Babelfish versi 2.4.0 dan 3.1.0. Parameter ini memungkinkan Anda untuk mengatur nomor versi produk SQL Server sebagai output dari Babelfish.

Parameternya adalah string ID versi 4 bagian, dan setiap bagian harus dipisahkan dengan ".".

Sintaks

```
Major.Minor.Build.Revision
```

- Versi Utama: Angka antara 11 dan 16.
- Versi Minor: Angka antara 0 dan 255.
- Versi Build: Angka antara 0 dan 65535.
- Revisi: 0 dan angka positif berapa pun.

Mengonfigurasi parameter versi produk Babelfish

Anda harus menggunakan grup parameter klaster untuk mengatur parameter babelfishpg_tds.product_version di konsol. Untuk informasi selengkapnya tentang cara memodifikasi parameter klaster DB, lihat Memodifikasi parameter dalam grup parameter klaster DB.

Saat Anda menyetel parameter versi produk ke nilai yang tidak valid, perubahan tidak akan berlaku. Meskipun konsol mungkin menunjukkan nilai baru, parameter akan mempertahankan nilai sebelumnya. Periksa file log mesin untuk detail tentang pesan kesalahan.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
--db-cluster-parameter-group-name mydbparametergroup \
```

```
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
--db-cluster-parameter-group-name mydbparametergroup ^
--parameters
 "ParameterName=babelfishpg_tds.product_version,ParameterValue=15.2.4000.1,ApplyMethod=immediat
```

Kueri dan parameter yang terpengaruh

Kueri/Parameter	Hasil	Waktu efektif
SELECT @@VERSION	Mengembalikan versi SQL Server yang ditentukan pengguna (nilai babelfish pg_tsql.version = Default)	Langsung
PILIH SERVERPROPERTY ("ProductVersion)	Mengembalikan versi SQL Server yang ditentukan pengguna	Langsung
PILIH SERVERPROPERTY ("ProductMajorVersion)	Mengembalikan Versi Utama dari versi SQL Server yang ditentukan pengguna	Langsung
Token VERSION dalam Pesan Respons PRELOGIN	Server mengembalikan pesan PRELOGIN dengan versi SQL Server yang ditentukan pengguna	Berlaku ketika pengguna membuat sesi baru
SQL ServerVersion di LoginAck saat menggunakan JDBC	DatabaseMetaData. getDatabaseProductVersion () mengembalikan versi SQL Server yang ditentukan pengguna	Berlaku ketika pengguna membuat sesi baru

Antarmuka dengan parameter babelfishpg_tsql.version

Anda dapat mengatur output dari @@VERSION menggunakan parameter babelfishpg_tsql.version dan babelfishpg_tds.product_version. Contoh berikut menunjukkan bagaimana antarmuka dua parameter ini.

- Ketika parameter babelfishpg_tsql.version adalah "default" dan babelfishpg_tds.product_version adalah 15.0.2000.8.
  - Output dari @@version – 15.0.2000.8.
- Ketika parameter babelfishpg_tsql.version disetel ke 13.0.2000.8 dan parameter babelfishpg_tds.product_version adalah 15.0.2000.8.
  - Output dari @@version – 13.0.2000.8.

Referensi Babelfish for Aurora PostgreSQL

Topik

- Fungsionalitas yang tidak didukung di Babelfish
- Fungsionalitas yang didukung di Babelfish berdasarkan versi
- Referensi prosedur Babelfish for Aurora PostgreSQL

Fungsionalitas yang tidak didukung di Babelfish

Dalam tabel dan daftar berikut, Anda dapat menemukan fungsionalitas yang saat ini tidak didukung di Babelfish. Pembaruan untuk Babelfish termasuk dalam versi Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Catatan rilis Aurora PostgreSQL.

Topik

- Fungsionalitas yang saat ini tidak didukung
- Pengaturan yang tidak didukung
- Perintah yang tidak didukung
- Nama kolom atau atribut yang tidak didukung
- Tipe data yang tidak didukung
- Tipe objek yang tidak didukung
- Fungsi yang tidak didukung
- Sintaks yang tidak didukung

Fungsionalitas yang saat ini tidak didukung

Dalam tabel, Anda dapat menemukan informasi tentang fungsionalitas tertentu yang saat ini tidak didukung.

Fungsionalitas atau sintaks	Deskripsi
Modul perakitan dan rutinitas SQL Common Language Runtime (CLR)	Fungsionalitas yang terkait dengan modul perakitan dan rutinitas CLR tidak didukung.

Fungsionalitas atau sintaks	Deskripsi
Atribut kolom	ROWGUIDCOL, SPARSE, FILESTREAM, dan MASKED tidak didukung.
Basis data bermuatan	Basis data bermuatan dengan login yang diautentikasi di tingkat basis data alih-alih di tingkat server tidak didukung.
Kursor (dapat diperbarui)	Kursor yang dapat diperbarui tidak didukung.
Kursor (global)	Kursor GLOBAL tidak didukung.
Kursor (perilaku pengambilan)	Perilaku pengambilan kursor berikut tidak didukung: FETCH PRIOR, FIRST, LAST, ABSOLUTE, abd RELATIVE
Parameter output tipe kursor	Variabel dan parameter tipe kursor tidak didukung untuk parameter output (kesalahan muncul).
Opsi kursor	SCROLL, KEYSET, DYNAMIC, FAST_FORWARD, SCROLL_LOCKS, OPTIMISTIC, TYPE_WARNING, dan FOR UPDATE
Enkripsi data	Enkripsi data tidak didukung.
Aplikasi tingkat data (DAC)	Operasi impor atau ekspor aplikasi tingkat data (DAC) dengan paket DAC (.dacpac) atau file cadangan DAC (.bacpac) tidak didukung.
Perintah DBCC	Perintah Konsol Basis Data (DBCC) Microsoft SQL Server tidak didukung. DBCC CHECKIDENT didukung di Babelfish 3.4.0 dan rilis yang lebih tinggi.
DROP IF EXISTS	Sintaks ini tidak didukung untuk objek USER dan SCHEMA. Ini didukung untuk objek TABLE, VIEW, PROCEDURE, FUNCTION, dan DATABASE.
Enkripsi	Fungsi dan pernyataan default tidak mendukung enkripsi.
Koneksi ENCRYPT_CLIENT_CERT	Koneksi sertifikat klien tidak didukung.

Fungsionalitas atau sintaks	Deskripsi
Pernyataan EXECUTE AS	Pernyataan ini tidak didukung.
Klausa EXECUTE AS SELF	Klausa ini tidak didukung dalam fungsi, prosedur, atau pemicu.
Klausa EXECUTE AS USER	Klausa ini tidak didukung dalam fungsi, prosedur, atau pemicu.
Batasan kunci asing yang mereferensikan nama basis data	Batasan kunci asing yang mereferensikan nama basis data tidak didukung.
FORMAT	Tipe yang ditentukan pengguna tidak didukung.
Deklarasi fungsi dengan lebih dari 100 parameter	Deklarasi fungsi yang berisi lebih dari 100 parameter tidak didukung.
Panggilan fungsi yang menyertakan DEFAULT sebagai nilai parameter	DEFAULT bukan nilai parameter yang didukung untuk panggilan fungsi. DEFAULT sebagai nilai parameter untuk panggilan fungsi didukung dari Babelfish 3.4.0 dan rilis yang lebih tinggi.
Fungsi, didefinisikan secara eksternal	Fungsi eksternal, termasuk fungsi SQL CLR, tidak didukung.
Tabel sementara global (tabel dengan nama yang dimulai dengan ##)	Tabel temporer global tidak didukung.
Fungsionalitas grafik	Semua fungsionalitas grafik SQL tidak didukung.
Pengidentifikasi (variabel atau parameter) dengan beberapa karakter @ utama	Pengidentifikasi yang dimulai dengan lebih dari satu @ utama tidak didukung.
Pengidentifikasi, nama tabel atau kolom yang berisi karakter @ atau]]	Nama tabel atau kolom yang berisi tanda @ atau tanda kurung siku tidak didukung.
Indeks inline	Indeks inline tidak didukung.

Fungsionalitas atau sintaks	Deskripsi
Menginvokasi prosedur yang namanya ada dalam variabel	Menggunakan variabel sebagai nama prosedur tidak didukung.
Tampilan terwujud	Tampilan terwujud tidak didukung.
Klausa NOT FOR REPLICATION	Sintaks ini diterima dan diabaikan.
Fungsi escape ODBC	Fungsi escape ODBC tidak didukung.
Partisi	Partisi tabel dan indeks tidak didukung.
Panggilan prosedur yang menyertakan DEFAULT sebagai nilai parameter	DEFAULT bukan nilai parameter yang didukung. DEFAULT sebagai nilai parameter untuk panggilan fungsi didukung dari Babelfish 3.4.0 dan rilis yang lebih tinggi.
Deklarasi prosedur dengan lebih dari 100 parameter	Deklarasi yang berisi lebih dari 100 parameter tidak didukung.
Prosedur, didefinisikan secara eksternal	Prosedur yang ditentukan secara eksternal, termasuk prosedur SQL CLR, tidak didukung.
Versioning prosedur	Versioning prosedur tidak didukung.
Prosedur WITH RECOMPILE	WITH RECOMPILE (saat digunakan bersama dengan pernyataan DECLARE dan EXECUTE) tidak didukung.
Referensi objek jarak jauh	Menjalankan prosedur dan fungsi menggunakan nama empat bagian tidak didukung. Dalam objek jarak jauh, mendukung nama objek empat bagian untuk kueri yang dipilih. Untuk informasi selengkapnya, lihat Pengaturan grup parameter klaster DB untuk Babelfish.
Keamanan tingkat baris	Keamanan tingkat baris dengan CREATE SECURITY POLICY dan fungsi bernilai tabel inline tidak didukung.
Fungsi broker layanan	Fungsi broker layanan tidak didukung.

Fungsionalitas atau sintaks	Deskripsi
PROPERTI SESI	Properti yang tidak didukung: ANSI_NULLS, ANSI_PADDING, ANSI_WARNINGS, ARITHABORT, CONCAT_NULL_YIELDS _NULL, dan NUMERIC_ROUNDABORT
SET LANGUAGE	Sintaks ini tidak didukung dengan nilai apa pun selain `english` atauus_english .
SP_CONFIGURE	Prosedur tersimpan sistem ini tidak didukung.
Kata kunci SPARSE SQL	Kata kunci SPARSE diterima dan diabaikan.
Sintaks konstruktor nilai tabel (klausa FROM)	Sintaks yang tidak didukung adalah untuk tabel turunan yang dibangun dengan klausa FROM.
Tabel temporal	Tabel temporal tidak didukung.
Prosedur sementara tidak dijatuhkan secara otomatis	Fungsionalitas ini tidak didukung.
Pemicu, didefinisikan secara eksternal	Pemicu ini tidak didukung, termasuk SQL Common Language Runtime (CLR).
Nilai string yang tidak dikutip dalam panggilan prosedur tersimpan dan nilai default	Parameter string ke panggilan prosedur tersimpan, dan default untuk parameter string di CREATE PROCEDURE, tidak didukung.
Tanpa klausa SCHEMABIN DING	Membuat tampilan tanpa SCHEMABINDING tidak didukung, tetapi tampilan dibuat seolah-olah WITH SCHEMABINDING ditentukan. Menggunakan SCHEMABINDING saat membuat fungsi, prosedur, pemicu diabaikan secara diam-diam.

Pengaturan yang tidak didukung

Pengaturan berikut tidak didukung:

- SET ANSI_NULL_DFLT_OFF ON
- SET ANSI_NULL_DFLT_ON OFF

- SET ANSI_PADDING OFF
- SET ANSI_WARNINGS OFF
- SET ARITHABORT OFF
- SET ARITHIGNORE ON
- SET CURSOR_CLOSE_ON_COMMIT ON
- SET NUMERIC_ROUNDABORT ON
- SET PARSEONLY ON (perintah tidak berfungsi seperti yang diharapkan)
- SET FMTONLY ON (perintah tidak berfungsi seperti yang diharapkan. Ini hanya menekan eksekusi pernyataan SELECT tetapi tidak yang lain.)

Perintah yang tidak didukung

Fungsionalitas tertentu untuk perintah berikut tidak didukung:

- ADD SIGNATURE
- ALTER DATABASE, ALTER DATABASE SET
- BACKUP/RESTORE DATABASE/LOG
- BACPAC and DACPAC FILES RESTORE
- BUAT, UBAH, JATUHKAN OTORISASI. OTORISASI ALTER didukung untuk objek database.
- CREATE, ALTER, DROP AVAILABILITY GROUP
- CREATE, ALTER, DROP BROKER PRIORITY
- CREATE, ALTER, DROP COLUMN ENCRYPTION KEY
- CREATE, ALTER, DROP DATABASE ENCRYPTION KEY
- CREATE, ALTER, DROP, BACKUP CERTIFICATE
- CREATE AGGREGATE
- CREATE CONTRACT
- CHECKPOINT

Nama kolom atau atribut yang tidak didukung

Nama kolom berikut tidak didukung:

- $IDENTITY
- $ROWGUID
- IDENTITYCOL

Tipe data yang tidak didukung

Tipe data berikut tidak didukung:

- Geospasial (GEOGRAPHY dan GEOMETRY)
- HIERARCHYID

Tipe objek yang tidak didukung

Tipe objek berikut tidak didukung:

- COLUMN MASTER KEY
- CREATE, ALTER EXTERNAL DATA SOURCE
- CREATE, ALTER, DROP DATABASE AUDIT SPECIFICATION
- CREATE, ALTER, DROP EXTERNAL LIBRARY
- CREATE, ALTER, DROP SERVER AUDIT
- CREATE, ALTER, DROP SERVER AUDIT SPECIFICATION
- CREATE, ALTER, DROP, OPEN/CLOSE SYMMETRIC KEY
- CREATE, DROP DEFAULT
- CREDENTIAL
- CRYPTOGRAPHIC PROVIDER
- DIAGNOSTIC SESSION
- Tampilan yang diindeks
- SERVICE MASTER KEY
- SYNONYM

Fungsi yang tidak didukung

Fungsi default berikut tidak didukung:

Fungsi agregat

- APPROX_COUNT_DISTINCT
- CHECKSUM_AGG
- GROUPING_ID
- STRING_AGG menggunakan klausa WITHIN GROUP

Fungsi kriptografi

- Fungsi CERTENCODED
- Fungsi CERTID
- Fungsi CERTPROPERTY

Fungsi metadata

- COLUMNPROPERTY
- TYPEPROPERTY
- Fungsi SERVERPROPERTY - Properti berikut tidak didukung:
  - BuildClrVersion
  - ComparisonStyle
  - ComputerNamePhysicalNetBIOS
  - HadrManagerStatus
  - InstanceDefaultDataPath
  - InstanceDefaultLogPath
  - IsClustered
  - IsHadrEnabled
  - LCID
  - NumLicenses
  - ProcessID
  - ProductBuild
  - ProductBuildType
  - ProductUpdateReference

- ResourceLastUpdateDateTime
- ResourceVersion
- ServerName
- SqlCharSet
- SqlCharSetName
- SqlSortOrder
- SqlSortOrderName
- FilestreamShareName
- FilestreamConfiguredLevel
- FilestreamEffectiveLevel

Fungsi keamanan

- CERTPRIVATEKEY
- LOGINPROPERTY

Pernyataan, operator, fungsi lainnya

- Fungsi EVENTDATA
- GET_TRANSMISSION_STATUS
- OPENXML

Sintaks yang tidak didukung

Sintaks berikut tidak didukung:

- ALTER DATABASE
- ALTER DATABASE SCOPED CONFIGURATION
- ALTER DATABASE SCOPED CREDENTIAL
- ALTER DATABASE SET HADR
- ALTER FUNCTION
- ALTER INDEX
- Pernyataan ALTER PROCEDURE

- ALTER SCHEMA
- ALTER SERVER CONFIGURATION
- Klausa ALTER SERVICE, BACKUP/RESTORE SERVICE MASTER KEY
- ALTER VIEW
- BEGIN CONVERSATION TIMER
- BEGIN DISTRIBUTED TRANSACTION
- BEGIN DIALOG CONVERSATION
- BULK INSERT
- CREATE COLUMNSTORE INDEX
- CREATE EXTERNAL FILE FORMAT
- CREATE EXTERNAL TABLE
- CREATE, ALTER, DROP APPLICATION ROLE
- CREATE, ALTER, DROP ASSEMBLY
- CREATE, ALTER, DROP ASYMMETRIC KEY
- CREATE, ALTER, DROP CREDENTIAL
- CREATE, ALTER, DROP CRYPTOGRAPHIC PROVIDER
- CREATE, ALTER, DROP ENDPOINT
- CREATE, ALTER, DROP EVENT SESSION
- CREATE, ALTER, DROP EXTERNAL LANGUAGE
- CREATE, ALTER, DROP EXTERNAL RESOURCE POOL
- CREATE, ALTER, DROP FULLTEXT CATALOG
- CREATE, ALTER, DROP FULLTEXT INDEX
- CREATE, ALTER, DROP FULLTEXT STOPLIST
- CREATE, ALTER, DROP MESSAGE TYPE
- CREATE, ALTER, DROP, OPEN/CLOSE, BACKUP/RESTORE MASTER KEY
- CREATE, ALTER, DROP PARTITION FUNCTION
- CREATE, ALTER, DROP PARTITION SCHEME
- CREATE, ALTER, DROP QUEUE
- CREATE, ALTER, DROP RESOURCE GOVERNOR
- CREATE, ALTER, DROP RESOURCE POOL

- CREATE, ALTER, DROP ROUTE
- CREATE, ALTER, DROP SEARCH PROPERTY LIST
- CREATE, ALTER, DROP SECURITY POLICY
- Klausa CREATE, ALTER, DROP SELECTIVE XML INDEX
- CREATE, ALTER, DROP SERVICE
- CREATE, ALTER, DROP SPATIAL INDEX
- CREATE, ALTER, DROP TYPE
- CREATE, ALTER, DROP XML INDEX
- CREATE, ALTER, DROP XML SCHEMA COLLECTION
- CREATE/DROP RULE
- CREATE, DROP WORKLOAD CLASSIFIER
- CREATE, ALTER, DROP WORKLOAD GROUP
- MENGUBAH PEMICU
- CREATE TABLE... Klausa GRANT
- CREATE TABLE... Klausa IDENTITY
- CREATE USER – Sintaks ini tidak didukung. Pernyataan PostgreSQL CREATE USER tidak membuat pengguna yang setara dengan sintaks CREATE USER SQL Server. Untuk informasi selengkapnya, lihat Perbedaan T-SQL di Babelfish.
- DENY
- END, MOVE CONVERSATION
- EXECUTE dengan opsi AS LOGIN atau AT
- GET CONVERSATION GROUP
- Klausa GROUP BY ALL
- Klausa GROUP BY CUBE
- Klausa GROUP BY ROLLUP
- INSERT... DEFAULT VALUES
- MERGE
- TEKS BACA
- REVERT
- PILIH PIVOT (didukung dari 3.4.0 dan rilis yang lebih tinggi kecuali bila digunakan dalam definisi tampilan, ekspresi tabel umum, atau gabungan) /UNPIVOT

- SELECT TOP x PERCENT WHERE x <> 100
- SELECT TOP... WITH TIES
- SELECT... FOR BROWSE
- SELECT... FOR XML AUTO
- SELECT... FOR XML EXPLICIT
- SEND
- SET DATEFORMAT
- SET DEADLOCK_PRIORITY
- SET FMTONLY
- SET FORCEPLAN
- SET NUMERIC_ROUNDABORT ON
- SET OFFSETS
- SET REMOTE_PROC_TRANSACTIONS
- SET SHOWPLAN_TEXT
- SET SHOWPLAN_XML
- SET STATISTICS
- SET STATISTICS PROFILE
- MENGATUR WAKTU STATISTIK
- SET STATISTICS XML
- Pernyataan SHUTDOWN
- UPDATE STATISTICS
- UPDATETEXT
- Menggunakan EXECUTE untuk memanggil fungsi SQL
- VIEW... Klausa CHECK OPTION
- VIEW... Klausa VIEW_METADATA
- WAITFOR DELAY
- WAITFOR TIME
- WAITFOR, RECEIVE
- Konsep WITH XMLNAMESPACES
- WRITETEXT

- Ekspresi XPATH

Fungsionalitas yang didukung di Babelfish berdasarkan versi

Dalam tabel berikut, Anda dapat menemukan fungsionalitas T-SQL yang didukung oleh versi Babelfish yang berbeda. Untuk daftar fungsionalitas yang tidak didukung, lihat Fungsionalitas yang tidak didukung di Babelfish. Untuk informasi tentang berbagai rilis Babelfish, lihat Catatan Rilis untuk Aurora PostgreSQL.

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
4 parts object name references for SELECT statements	✓	✓	✓	✓	–	✓	–	–	–	–	–	–	–	–	–
ALTER AUTHORIZATION syntax to change database owner	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
ALTER ROLE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
ALTER USER...WI	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
TH LOGIN AT TIME ZONE clause	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
Babelfish instance as a linked server	✓	✓	✓	✓	–	✓	–	–	–	–	–	–	–	–	–
CREATE ROLE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
CREATE TRIGGER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Create unique indexes	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cross-database procedure execution	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
Cross-database references SELECT,SELECT..INTO, INSERT, UPDATE, DELETE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
Cursor typed parameters for input parameters only (not output)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Data migration using the bcp client utility	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
Database authentication with Kerberos using AWS Directory Service	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
Datatypes TIMESTAMP, ROWVERSION (for usage information, see Features with limited implementation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
DEFAULT keyword in calls to stored procedures and functions	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
DBCC CHECKIDENT	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
DROP DATABASE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–
DROP IF EXISTS (for SCHEMA, DATABASE, and USER objects)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
DROP ROLE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
Dump and restore	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
ENABLE/DISABLE TRIGGER	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
FULL TEXT SEARCH	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
GRANT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
GUC babelfish pg_tds.product_version	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
Identifiers with leading dot character	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
INSTEAD OF triggers on tables	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
INSTEAD OF triggers on views	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
KILL	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
pg_stat_statement extension	✓	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
PIVOT (supported from 3.4.0 and higher releases except when used in a view definition, a common table expression, or a join)	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
REVOKE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
SET BABELFISH_SHOWPLAN_ALL ON (and OFF)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
SET BABELFISH_STATISTICS PROFILE ON (OFF)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
SET CONTEXT_INFO	✓	✓	✓	✓	–	✓	–	–	–	–	–	–	–	–	–
SET LOCK_TIMEOUT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
SET NO_BROWSETABLE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
SET rowcount	✓	✓	✓	✓	–	✓	–	–	–	–	–	–	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
SET SHOWPLAN_ALL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
SET STATISTICS IO	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
SSMS: Connecting with the object explorer connection dialog	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
SSMS: Data migration with the Import/Export Wizard	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
SSMS: Partial support for the object explorer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
STDEV	✓	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–
STDEVP	✓	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
Triggers with multiple DML actions can reference transition tables	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
T-SQL hints (join methods, index usage, MAXDOP)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–
T-SQL square bracket syntax with the LIKE predicate	✓	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–
VAR	✓	✓	✓	✓	–	✓	–	–	–	–	–	–	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
VARP	✓	✓	✓	✓	–	✓	–	–	–	–	–	–	–	–	–
Zero-down time patching (ZDP)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–
Built-in functions:															
APP_NAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
ATN2	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
CHARINDEX	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
CHOOSE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
COL_LENGTH	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
COL_NAME	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
COLUMNS_UPDATED		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
COLUMNPROPERTY (CharMaxLen, AllowsNull only)		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
CONCAT_WS		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
CONTEXT_INFO	✓	✓	✓	–	✓	–	–	–	–	–	–	–	–	–	
CURSOR_STATUS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
DATABASE_PRINCIPAL_ID	✓	✓	✓	–	✓	–	–	–	–	–	–	–	–	–	
DATEADD	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	
DATEDIFF	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	
DATEDIFF_BIG	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–	
DATEFROMPARTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
DATENAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	
DATEPART	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	
DATETIMEFROMPARTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
DATETIME2FROMPARTS	✓	✓	✓	–	✓	–	–	–	–	–	–	–	–	–	
DATETIMEOFFSETFROMPARTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	
DATETRUNC	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
DATE_BUCKET	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
EOMONTH	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
EXECUTE AS CALLER	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–	–
fn_listextendedproperty	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–	–
FOR JSON	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–
FULLTEXTSERVICEPROPERTY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
HAS_DBACCESS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
HAS_PERMS_BY_NAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
HOST_NAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
HOST_ID	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–	–
IDENTITY	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
IS_MEMBER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
IS_ROLEMEMBER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
IS_SRVROLEMEMBER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
ISJSON	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
JSON_MODIFY	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
JSON_QUERY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
JSON_VALUE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
NEXT VALUE FOR	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
OBJECT_DEFINITION	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
OBJECT_SCHEMA_NAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
OPENJSON	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
OPENQUERY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
ORIGINAL_LOGIN	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
PARSENAME	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
PATINDEX	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
ROWCOUNT_BIG	✓	✓	✓	✓	–	✓	–	–	–	–	–	–	–	–	–
SCHEMA_NAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
SESSION_CONTEXT	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
SESSION_USER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
SID_BINARY (returns NULL always)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
SMALLDATETIMEFROMPARTS	✓	✓	✓	–	–	✓	✓	–	–	–	–	–	–	–	–
SQUARE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
STR	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
STRING_AGG	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
STRING_SPLIT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
SUSER_SID	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
SUSER_SNAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
SWITCHOFFSET	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
SYSTEM_USER	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–
TIMEFROMPARTS	✓	✓	✓	–	✓	–	–	–	–	–	–	–	–	–	–
TODATETIMEOFFSET	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
TO_CHAR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–
TRIGGER_NESTLEVEL (without arguments only)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
TRY_CONVERT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–
TYPE_ID	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
TYPE_NAME	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
UPDATE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–

INFORMATION_SCHEMA catalogs

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
CHECK_CONSTRAINTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
COLUMN_DOMAIN_USAGE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
COLUMNS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
CONSTRAINT_COLUMN_USAGE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
DOMAINS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
KEY_COLUMN_USAGE	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
ROUTINES	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
TABLES	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
TABLE_CONSTRAINTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
VIEWS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
System-defined @@ variables:															
@@CURSOR_ROWS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
@@DATEFIRST	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
@@DBTS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
@@ERROR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@ERROR=213	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
@@FETCH_STATUS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@IDENTITY	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@LANGUAGE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@LOCK_TIMEOUT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
@@MAX_CONNECTIONS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
@@MAX_PRECISION	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@MICROSOFTVERSION	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
@@NESTLEVEL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
@@PROCID	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
@@ROWCOUNT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
@@SERVERNAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@SERVICENAME	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
@@SPID	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@TRANCOUNT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
@@VERSION (note format difference as described in Perbedaan T-SQL di Babelfish.	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
System stored procedures:															
sp_addextendedproperty	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
sp_addlinkedserver	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
sp_addlinkedsrvlogin	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
sp_addrole	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–
sp_addrolemember	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–
sp_babelfish_volatility	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
sp_column_privileges	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sp_columns	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_columns_100	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_columns_managed	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_cursor	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_cursor_list	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
sp_cursorclose	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_cursorexecute	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_cursorfetch	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_cursoropen	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_cursoroption	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_cursorprepare	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_cursorprepexec	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_cursorunprepare	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_databases	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_datatype_info	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_datatype_info_100	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
sp_describe_cursor	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_describe_first_result_set	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_describe_undeclared_parameters	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sp_dropextendedproperty	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–	–
sp_droplinkedsrvlogin	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–	–
sp_droprole	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
sp_droprolemember	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
sp_dropserver	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–	–
sp_enumoledb_providers	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
sp_execute	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_execute_postgresql(CREATE, ALTER, DROP)	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–	–
sp_executesql	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_fkeys	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sp_getapplock	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_helpdb	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_helpdbfixedrole	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
sp_helplinkedsrvlogin	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
sp_helprole	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sp_helprolemember	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
sp_helpsrvrolemember	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
sp_helpuser	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sp_linkedservers	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
sp_oledb_ro_usrname	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_pkeys	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_prefix	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
sp_prepare	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_releaseapplock	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sp_rename	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
sp_serveroption(connect_timeout option)	✓	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
sp_set_session_context	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
sp_special_columns	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sp_sproc_columns	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sp_sproc_columns_100	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sp_statistics	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_statistics_100	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_stored_procedures	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sp_table_privileges	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sp_table_collations_100	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_tables	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
sp_testlinkedserver	✓	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–
sp_unprepare	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
sp_updateextendedproperty	✓	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–
sp_who	✓	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–
xp_qv	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
Properties supported on the CONNECTIONPROPERTY system function															
auth_scheme	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
client_net_address	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
local_net_address	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
local_tcp_port	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
net_transport	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
protocol_type	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
physical_net_transport	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
Properties supported on the OBJECTPROPERTY system function															
IsInlineFunction	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IsScalarFunction	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
IsTableFunction	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–
Properties supported on the SERVERPROPERTY function															
Babelfish	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Collation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CollationID	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Edition	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EditionID	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
EngineEdition	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
InstanceName	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
IsAdvancedAnalyticsInstalled	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
IsBigDataCluster	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
IsFullTextInstalled	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
IsIntegratedSecurityOnly	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
IsLocalDB	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
IsPolyBaseInstalled	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
IsSingleUser	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
IsXTPSupported	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
Japanese_CI_AI	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
Japanese_CI_AS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
Japanese_CS_AS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
LicenseType	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
MachineName	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
ProductLevel	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
ProductMajorVersion	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
ProductMinorVersion	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
ProductUpdateLevel	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
ProductVersion	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
ServerName	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

SQL Server views supported by Babelfish

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
information_schema.key_column_usage	✓	–	–	–	–	–	–	–	–	–	–	–	–	–	–
information_schema.routines	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–	–
information_schema.schemata	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–	–
information_schema.sequences	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–	–
sys.all_columns	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.all_objects	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.all_parameters	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
sys.all_sql_modules	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
sys.all_views	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
sys.columns	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.configurations	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sys.data_spaces	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sys.database_files	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sys.database_mirroring	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sys.database_principals	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sys.database_role_members	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sys.databases	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.dm_exec_connections	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sys.dm_exec_sessions	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
sys.dm_hadr_database_replica_states	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sys.dm_os_host_info	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sys.endpoints	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sys.extended_properties	✓	✓	–	–	✓	–	–	–	–	–	–	–	–	–	–
sys.indexes	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
sys.schemas	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.server_principals	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.server_role_members	✓	✓	–	–	–	–	–	–	–	–	–	–	–	–	–
sys.sql_modules	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–

Fungsionalitas atau sintaks	4.0.0	3.4.0	3.3.0	3.2.0	3.1.0	2.7.0	2.6.0	2.5.0	2.4.0	2.3.0	2.2.0	2.1.0	1.2.1	1.1.0	1.0.0
sys.syscon figures	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sys.syscur rconfigs	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sys.syslo gins	✓	✓	✓	–	✓	–	–	–	–	–	–	–	–	–	–
sys.syspr ocesses	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sys.sysus ers	✓	✓	✓	–	✓	–	–	–	–	–	–	–	–	–	–
sys.table _types	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–
sys.table s	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
sys.types	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–	–
sys.xml_s chema_col lections	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
syslanguages	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
sysobjects.crdate	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–

Referensi prosedur Babelfish for Aurora PostgreSQL

Gambaran Umum

Anda dapat menggunakan prosedur berikut untuk instans DB Amazon RDS yang menjalankan Babelfish for Aurora PostgreSQL untuk performa kueri yang lebih baik:

- sp_babelfish_volatility
- sp_execute_postgresql

sp_babelfish_volatility

Volatilitas fungsi PostgreSQL membantu pengoptimal untuk eksekusi kueri yang lebih baik yang bila digunakan pada bagian klausa tertentu memiliki dampak signifikan pada performa kueri.

Sintaksis

```
sp_babelfish_volatility ‘function_name’, ‘volatility’
```

Argumen

function_name (opsional)

Anda dapat menentukan nilai argumen ini dengan nama dua bagian sebagai `schema_name.function_name` atau hanya `function_name`. Jika hanya menentukan `function_name`, nama skemanya adalah skema default untuk pengguna saat ini.

volatilitas (opsional)

Nilai volatilitas PostgreSQL yang valid adalah `stable`, `volatile`, atau `immutable`. Untuk informasi selengkapnya, lihat https://www.postgresql.org/docs/current/xfunc-volatility.html

> **Note**
>
> Saat `sp_babelfish_volatility` dipanggil dengan `function_name` yang memiliki beberapa definisi, hal tersebut akan menimbulkan kesalahan.

Set hasil

Jika parameter tidak disebutkan, maka set hasil ditampilkan di bawah kolom berikut: `schemaname`, `functionname`, `volatility`.

Catatan penggunaan

Volatilitas fungsi PostgreSQL membantu pengoptimal untuk eksekusi kueri yang lebih baik yang bila digunakan pada bagian klausa tertentu memiliki dampak signifikan pada performa kueri.

Contoh

Contoh berikut menunjukkan cara membuat fungsi sederhana, lalu menjelaskan cara menggunakan `sp_babelfish_volatility` pada fungsi-fungsi ini menggunakan metode yang berbeda.

```
1> create function f1() returns int as begin return 0 end
2> go
```

```
1> create schema test_schema
2> go
```

```
1> create function test_schema.f1() returns int as begin return 0 end
2> go
```

Contoh berikut menunjukkan volatilitas fungsi tersebut:

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           volatile
test_schema f1           volatile
```

Contoh berikut menunjukkan cara mengubah volatilitas fungsi tersebut:

```
1> exec sp_babelfish_volatility 'f1','stable'
2> go
1> exec sp_babelfish_volatility 'test_schema.f1','immutable'
```

```
2> go
```

Saat Anda hanya menentukan function_name, hal tersebut menampilkan nama skema, nama fungsi, dan volatilitas fungsi tersebut. Contoh berikut menampilkan volatilitas fungsi setelah mengubah nilai:

```
1> exec sp_babelfish_volatility 'test_schema.f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
test_schema f1           immutable
```

```
1> exec sp_babelfish_volatility 'f1'
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
```

Saat Anda tidak menentukan argumen apa pun, daftar fungsi akan ditampilkan (nama skema, nama fungsi, volatilitas fungsi) yang ada dalam basis data saat ini:

```
1> exec sp_babelfish_volatility
2> go

schemaname  functionname volatility
----------- ------------ ----------
dbo         f1           stable
test_schema f1           immutable
```

sp_execute_postgresql

Anda dapat mengeksekusi pernyataan PostgreSQL dari titik akhir T-SQL. Hal ini menyederhanakan aplikasi karena Anda tidak perlu keluar dari port T-SQL untuk mengeksekusi pernyataan ini.

Sintaksis

```
sp_execute_postgresql [ @stmt = ] statement
```

Argumen

Pernyataan [@stmt]

Argumennya adalah dari datatype varchar. Argumen ini menerima pernyataan dialek PG.

> **Note**
>
> Anda hanya dapat meneruskan satu pernyataan dialek PG sebagai argumen, jika tidak, maka akan menimbulkan kesalahan berikut.

```
1>exec sp_execute_postgresql 'create extension pg_stat_statements; drop extension
 pg_stat_statements'
2>go
```

```
Msg 33557097, Level 16, State 1, Server BABELFISH, Line 1
expected 1 statement but got 2 statements after parsing
```

Catatan penggunaan

CREATE EXTENSION

Membuat dan memuat ekstensi baru ke dalam basis data saat ini.

```
1>EXEC sp_execute_postgresql 'create extension [ IF NOT EXISTS ] <extension name>
 [ WITH ] [SCHEMA schema_name] [VERSION version]';
2>go
```

Contoh berikut menunjukkan cara membuat ekstensi:

```
1>EXEC sp_execute_postgresql 'create extension pg_stat_statements with schema sys
 version "1.10"';
2>go
```

Gunakan perintah berikut untuk mengakses objek ekstensi:

```
1>select * from pg_stat_statements;
2>go
```

> **Note**
>
> Jika nama skema tidak diberikan secara eksplisit selama pembuatan ekstensi, secara default ekstensi diinstal di skema publik. Anda harus memberikan kualifikasi skema untuk mengakses objek ekstensi seperti yang disebutkan di bawah ini:

```
1>select * from [public].pg_stat_statements;
2>go
```

Ekstensi yang didukung

Ekstensi berikut yang tersedia dengan Aurora PostgreSQL bekerja dengan Babelfish.

- `pg_stat_statements`
- `tds_fdw`
- `fuzzystrmatch`

Batasan

- Pengguna harus memiliki peran sysadmin pada T-SQL dan rds_superuser di postgres untuk menginstal ekstensi.
- Ekstensi tidak dapat diinstal pada skema yang dibuat pengguna dan juga pada skema dbo dan tamu untuk basis data master, tempdb, dan msdb.
- Opsi CASCADE tidak didukung.

ALTER EXTENSION

Anda dapat meningkatkan ke versi ekstensi baru menggunakan ekstensi ALTER.

```
1>EXEC sp_execute_postgresql 'alter extension <extension name> UPDATE TO
 <new_version>';
2>go
```

Batasan

- Anda dapat meningkatkan versi ekstensi Anda hanya menggunakan pernyataan ALTER Extension. Opsi lain tidak didukung.

DROP EXTENSION

Membatalkan ekstensi yang ditentukan. Anda juga dapat menggunakan opsi `if exists` atau `restrict` untuk membatalkan ekstensi.

```
1>EXEC sp_execute_postgresql 'drop extension <extension name>';
2>go
```

Batasan

- Opsi CASCADE tidak didukung.

Mengelola Amazon Aurora PostgreSQL

Bagian berikut membahas pengelolaan performa dan penskalaan untuk klaster DB Amazon Aurora PostgreSQL. Bagian tersebut juga berisi informasi tentang tugas pemeliharaan lainnya.

Topik

- Penskalaan instans DB Aurora PostgreSQL
- Koneksi maksimum ke instans DB Aurora PostgreSQL
- Batas penyimpanan sementara untuk Aurora PostgreSQL
- Halaman besar untuk Aurora PostgreSQL
- Menguji Amazon Aurora PostgreSQL menggunakan kueri injeksi kesalahan
- Menampilkan status volume untuk klaster DB Aurora PostgreSQL
- Menentukan disk RAM untuk stats_temp_directory
- Mengelola file sementara dengan PostgreSQL

Penskalaan instans DB Aurora PostgreSQL

Anda dapat menskalakan instans DB Aurora PostgreSQL dengan dua cara, penskalaan instans dan penskalaan baca. Untuk informasi selengkapnya tentang penskalaan baca, lihat Penskalaan baca.

Anda dapat menskalakan klaster DB Aurora PostgreSQL Anda dengan mengubah kelas instans DB untuk setiap instans DB dalam klaster DB. Aurora PostgreSQL mendukung beberapa kelas instans DB yang dioptimalkan untuk Aurora. Jangan gunakan kelas instans db.t2 atau db.t3 untuk klaster Aurora yang lebih besar dengan ukuran lebih dari 40 terabyte (TB).

> **Note**
>
> Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk detail selengkapnya tentang kelas instans T, lihat Jenis kelas instans DB.

Penskalaan tidak terjadi secara instan. Diperlukan waktu 15 menit atau lebih untuk menyelesaikan perubahan ke kelas instans DB yang berbeda. Jika Anda menggunakan pendekatan ini untuk mengubah kelas instans DB, Anda perlu menerapkan perubahan selama periode pemeliharaan terjadwal berikutnya (bukan segera) agar tidak memengaruhi pengguna.

Sebagai alternatif untuk mengubah kelas instans DB secara langsung, Anda dapat meminimalkan waktu henti dengan menggunakan fitur ketersediaan tinggi Amazon Aurora. Pertama, tambahkan Replika Aurora ke klaster Anda. Saat membuat replika, pilih ukuran kelas instans DB yang ingin Anda gunakan untuk klaster Anda. Ketika Replika Aurora disinkronkan dengan klaster, maka Anda akan melakukan failover ke Replika yang baru ditambahkan. Untuk mempelajari selengkapnya, lihat Replika Aurora dan Failover cepat dengan Amazon Aurora PostgreSQL.

Untuk spesifikasi terperinci tentang kelas instans DB yang didukung oleh Aurora PostgreSQL, lihat Mesin DB yang didukung untuk kelas instans DB.

Koneksi maksimum ke instans DB Aurora PostgreSQL

klaster DB Aurora PostgreSQL mengalokasikan sumber daya berdasarkan kelas instans DB dan memori yang tersedia. Setiap koneksi ke klaster DB mengonsumsi sejumlah tambahan sumber daya ini, seperti memori dan CPU. Memori yang dikonsumsi per koneksi bervariasi berdasarkan jenis kueri, jumlah, dan apakah tabel sementara digunakan. Bahkan koneksi idle menghabiskan memori

dan CPU. Hal ini karena ketika kueri dijalankan pada koneksi, lebih banyak memori dialokasikan untuk setiap kueri dan tidak dirilis sepenuhnya, bahkan ketika pemrosesan berhenti. Oleh karena itu, kami menyarankan agar Anda memastikan aplikasi Anda tidak mempertahankan koneksi idle: setiap koneksi idle akan memboroskan sumber daya dan memengaruhi performa secara negatif. Untuk informasi selengkapnya, lihat Sumber daya yang dikonsumsi oleh koneksi PostgreSQL idle.

Jumlah maksimum koneksi yang diizinkan oleh instans DB Aurora PostgreSQL akan ditentukan berdasarkan nilai parameter `max_connections` yang ditentukan dalam grup parameter untuk instans DB tersebut. Pengaturan ideal untuk `max_connections` parameter adalah yang mendukung semua koneksi klien yang dibutuhkan aplikasi Anda, tanpa kelebihan koneksi yang tidak digunakan, ditambah setidaknya 3 koneksi lagi untuk mendukung AWS otomatisasi. Sebelum mengubah pengaturan parameter `max_connections`, sebaiknya pertimbangkan hal berikut:

- Jika nilai `max_connections` terlalu rendah, instans DB Aurora PostgreSQL mungkin tidak memiliki koneksi yang cukup saat klien mencoba terhubung. Jika hal ini terjadi, coba hubungkan menggunakan pesan "raise error" `psql` seperti berikut ini:

  ```
  psql: FATAL: remaining connection slots are reserved for non-replication superuser
   connections
  ```

- Jika nilai `max_connections` melebihi jumlah koneksi yang benar-benar dibutuhkan, koneksi yang tidak terpakai dapat menyebabkan performa menurun.

Nilai default `max_connections` berasal dari fungsi `LEAST` Aurora PostgreSQL berikut:

`LEAST({DBInstanceClassMemory/9531392},5000)`.

Jika Anda ingin mengubah nilai `max_connections`, Anda perlu membuat grup parameter klaster DB kustom dan mengubah nilainya di sana. Setelah menerapkan grup parameter DB kustom Anda ke klaster Anda, pastikan untuk mem-boot ulang instans primer agar nilai baru mulai berlaku. Lihat informasi yang lebih lengkap di Parameter Amazon Aurora PostgreSQL dan Membuat grup parameter klaster DB.

> **Tip**
>
> Jika aplikasi Anda sering membuka dan menutup koneksi, atau membiarkan sejumlah besar koneksi yang lama aktif tetap terbuka, kami menyarankan agar Anda menggunakan Proksi Amazon RDS. Proksi RDS adalah proksi basis data yang sepenuhnya terkelola dengan ketersediaan tinggi yang menggunakan pooling koneksi untuk berbagi koneksi basis data

dengan aman dan efisien. Untuk mempelajari tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS untuk Aurora.

Untuk detail tentang cara instans Aurora Serverless v2 menangani parameter ini, lihat Koneksi maksimum untuk Aurora Serverless v2.

Batas penyimpanan sementara untuk Aurora PostgreSQL

Aurora PostgreSQL menyimpan tabel dan indeks dalam subsistem penyimpanan Aurora. Aurora PostgreSQL menggunakan penyimpanan sementara terpisah untuk file sementara non-persisten. Hal ini termasuk file yang digunakan untuk tujuan seperti mengurutkan set data besar selama pemrosesan kueri atau untuk operasi pembuatan indeks. Untuk informasi selengkapnya, lihat artikel Bagaimana cara memecahkan masalah penyimpanan lokal di instans yang kompatibel dengan Aurora PostgreSQL?.

Volume penyimpanan lokal ini didukung oleh Amazon Elastic Block Store dan dapat diperluas dengan menggunakan kelas instans DB yang lebih besar. Untuk informasi selengkapnya tentang penyimpanan, lihat Penyimpanan dan keandalan Amazon Aurora. Anda juga dapat meningkatkan penyimpanan lokal untuk objek sementara dengan menggunakan jenis instans berkemampuan NVMe dan objek sementara yang diaktifkan oleh Aurora Optimized Read-enabled. Untuk informasi selengkapnya, lihat Meningkatkan performa kueri untuk Aurora PostgreSQL dengan Aurora Optimized Reads.

> **Note**
>
> Anda mungkin melihat peristiwa `storage-optimization` saat menskalakan instans DB, misalnya, dari db.r5.2xlarge ke db.r5.4xlarge.

Tabel berikut menunjukkan jumlah maksimum penyimpanan sementara yang tersedia untuk setiap kelas instans DB Aurora PostgreSQL. Untuk informasi selengkapnya tentang dukungan kelas instans DB untuk Aurora, lihat Kelas instans DB Aurora.

Kelas instans DB	Penyimpanan sementara maksimum yang tersedia (GiB)
db.x2g.16xlarge	1829

Kelas instans DB	Penyimpanan sementara maksimum yang tersedia (GiB)
db.x2g.12xlarge	1606
db.x2g.8xlarge	1071
db.x2g.4xlarge	535
db.x2g.2xlarge	268
db.x2g.xlarge	134
db.x2g.large	67
db.r7g.16xlarge	1008
db.r7g.12xlarge	756
db.r7g.8xlarge	504
db.r7g.4xlarge	252
db.r7g.2xlarge	126
db.r7g.xlarge	63
db.r7g.large	32
db.r6g.16xlarge	1008
db.r6g.12xlarge	756
db.r6g.8xlarge	504
db.r6g.4xlarge	252
db.r6g.2xlarge	126
db.r6g.xlarge	63
db.r6g.large	32

Kelas instans DB	Penyimpanan sementara maksimum yang tersedia (GiB)
db.r6i.32xlarge	1829
db.r6i.24xlarge	1500
db.r6i.16xlarge	1008
db.r6i.12xlarge	748
db.r6i.8xlarge	504
db.r6i.4xlarge	249
db.r6i.2xlarge	124
db.r6i.xlarge	62
db.r6i.large	31
db.r5.24xlarge	1500
db.r5.16xlarge	1008
db.r5.12xlarge	748
db.r5.8xlarge	504
db.r5.4xlarge	249
db.r5.2xlarge	124
db.r5.xlarge	62
db.r5.large	31
db.r4.16xlarge	960
db.r4.8xlarge	480
db.r4.4xlarge	240

Kelas instans DB	Penyimpanan sementara maksimum yang tersedia (GiB)
db.r4.2xlarge	120
db.r4.xlarge	60
db.r4.large	30
db.t4g.large	16.5
db.t4g.medium	8.13
db.t3.large	16
db.t3.medium	7.5

> **Note**
>
> Jenis instans yang diaktifkan NVMe dapat meningkatkan ruang sementara yang tersedia hingga ukuran NVMe total. Untuk informasi selengkapnya, lihat Meningkatkan performa kueri untuk Aurora PostgreSQL dengan Aurora Optimized Reads.

Anda dapat memantau penyimpanan sementara yang tersedia untuk instans DB dengan `FreeLocalStorage` CloudWatch metrik, -> dijelaskan dalam CloudWatch Metrik Amazon untuk Amazon Aurora. (Hal ini tidak berlaku untuk Aurora Serverless v2.)

Untuk beberapa beban kerja, Anda dapat mengurangi jumlah penyimpanan sementara dengan mengalokasikan lebih banyak memori ke proses yang melakukan operasi. Untuk meningkatkan ketersediaan memori bagi operasi, tingkatkan nilai parameter PostgreSQL work_mem atau maintenance_work_mem.

Halaman besar untuk Aurora PostgreSQL

Halaman besar adalah fitur manajemen memori yang mengurangi overhead saat instans DB menangani potongan besar memori yang berdekatan, seperti yang digunakan oleh buffer bersama. Fitur PostgreSQL ini didukung oleh semua versi Aurora PostgreSQL yang tersedia saat ini.

Parameter `Huge_pages` diaktifkan secara default untuk semua kelas instans DB selain kelas instans t3.medium, db.t3.large, db.t4g.medium, db.t4g.large. Anda tidak dapat mengubah nilai parameter `huge_pages` atau menonaktifkan fitur ini di kelas instans Aurora PostgreSQL yang didukung.

Menguji Amazon Aurora PostgreSQL menggunakan kueri injeksi kesalahan

Anda dapat menguji toleransi kesalahan klaster DB Aurora PostgreSQL Anda menggunakan kueri injeksi kesalahan. Kueri injeksi kesalahan dikeluarkan sebagai perintah SQL ke instans Amazon Aurora. Kueri injeksi kesalahan memungkinkan Anda menyebabkan crash pada instans sehingga Anda dapat menguji failover dan pemulihan. Anda juga dapat menyimulasikan kegagalan Replika Aurora, kegagalan disk, dan kepadatan disk. Kueri injeksi kesalahan didukung oleh semua versi Aurora PostgreSQL yang tersedia, sebagai berikut.

- Aurora PostgreSQL versi 12, 13, 14, dan lebih tinggi
- Aurora PostgreSQL versi 11.7 dan lebih tinggi
- Aurora PostgreSQL versi 10.11 dan lebih tinggi

Topik

- Menguji crash instans
- Menguji kegagalan Replika Aurora
- Menguji kegagalan disk
- Menguji kepadatan disk

Ketika kueri injeksi kesalahan menentukan suatu crash, kueri ini akan menyebabkan crash pada instans DB Aurora PostgreSQL. Kueri injeksi kesalahan lainnya menghasilkan simulasi peristiwa kegagalan, tetapi tidak menyebabkan peristiwa terjadi. Jika Anda mengirim kueri injeksi kesalahan, Anda juga menentukan durasi waktu untuk simulasi peristiwa kegagalan yang akan terjadi.

Anda dapat mengirim kueri injeksi kesalahan ke salah satu instans Replika Aurora dengan menghubungkan ke titik akhir untuk Replika Aurora. Untuk informasi selengkapnya, lihat Manajemen koneksi Amazon Aurora.

Menguji crash instans

Anda dapat memaksa crash instans Aurora PostgreSQL menggunakan fungsi kueri injeksi kesalahan `aurora_inject_crash()`.

Untuk kueri injeksi kesalahan ini, failover tidak akan terjadi. Jika Anda ingin menguji failover, maka Anda dapat memilih tindakan instance Failover untuk cluster DB Anda di konsol RDS, atau menggunakan failover-db-clusterAWS CLI perintah atau operasi FailOverdbCluster RDS API.

Sintaksis

```
SELECT aurora_inject_crash ('instance' | 'dispatcher' | 'node');
```

Opsi

Kueri injeksi kesalahan ini menggunakan salah satu dari jenis crash berikut. Jenis crash tidak peka huruf besar/kecil:

'instans'

Crash basis data yang kompatibel dengan PostgreSQL untuk instans Amazon Aurora disimulasikan.

'dispatcher'

Crash dispatcher pada instans primer untuk klaster DB Aurora disimulasikan. Dispatcher menulis pembaruan ke volume klaster untuk klaster DB Amazon Aurora.

'simpul'

Crash basis data yang kompatibel dengan PostgreSQL dan dispatcher untuk instans Amazon Aurora disimulasikan.

Menguji kegagalan Replika Aurora

Anda dapat menyimulasikan kegagalan Replika Aurora menggunakan fungsi kueri injeksi kesalahan `aurora_inject_replica_failure()`.

Kegagalan Replika Aurora akan memblokir replikasi ke Replika Aurora atau semua Replika Aurora dalam klaster DB untuk interval waktu yang ditentukan. Ketika interval waktu selesai, Replika Aurora yang terdampak akan secara otomatis tersinkronisasi dengan instans primer.

Sintaksis

```
SELECT aurora_inject_replica_failure(
   percentage_of_failure,
   time_interval,
   'replica_name'
```

```
);
```

Opsi

Kueri injeksi kesalahan ini mengambil parameter berikut:

percentage_of_failure

Persentase replikasi yang akan diblokir selama peristiwa kegagalan. Nilai ini dapat berupa double antara 0 dan 100. Jika Anda menetapkan 0, tidak ada replikasi yang akan diblokir. Jika Anda menetapkan 100, semua replikasi diblokir.

time_interval

Durasi waktu simulasi kegagalan Replika Aurora. Interval dalam detik. Misalnya, jika nilainya adalah 20, simulasi berjalan selama 20 detik.

> **Note**
>
> Berhati-hatilah saat menentukan interval waktu untuk peristiwa kegagalan Replika Aurora Anda. Jika Anda menentukan interval yang terlalu panjang, dan instans penulis Anda menulis data dalam jumlah besar selama peristiwa kegagalan, maka klaster DB Aurora Anda mungkin menganggap bahwa Replika Aurora telah crash lalu menggantinya.

replica_name

Replika Aurora tempat simulasi kegagalan akan diinjeksikan. Tentukan nama Replika Aurora untuk menyimulasikan kegagalan satu Replika Aurora. Tentukan string kosong untuk menyimulasikan kegagalan untuk semua Replika Aurora dalam klaster DB.

Untuk mengidentifikasi nama replika, lihat kolom `server_id` dari fungsi `aurora_replica_status()`. Sebagai contoh:

```
postgres=> SELECT server_id FROM aurora_replica_status();
```

Menguji kegagalan disk

Anda dapat menyimulasikan kegagalan disk untuk klaster DB Aurora PostgreSQL menggunakan fungsi kueri injeksi kesalahan `aurora_inject_disk_failure()`.

Selama simulasi kegagalan disk, klaster DB Aurora PostgreSQL menandai secara acak segmen disk sebagai mengalami kesalahan. Permintaan untuk segmen tersebut diblokir selama durasi simulasi.

Sintaksis

```
SELECT aurora_inject_disk_failure(
   percentage_of_failure,
   index,
   is_disk,
   time_interval
);
```

Opsi

Kueri injeksi kesalahan ini mengambil parameter berikut:

percentage_of_failure

Persentase disk yang akan ditandai sebagai mengalami kesalahan selama peristiwa kegagalan. Nilai ini dapat berupa double antara 0 dan 100. Jika Anda menentukan 0, tidak ada disk yang akan ditandai sebagai mengalami kesalahan. Jika Anda menentukan 100, seluruh disk akan ditandai sebagai mengalami kesalahan.

indeks

Blok data logis tertentu tempat peristiwa kegagalan akan disimulasikan. Jika Anda melampaui rentang blok logis atau data simpul penyimpanan yang tersedia, Anda akan menerima pesan kesalahan yang memberitahukan nilai indeks maksimum yang dapat Anda tentukan. Untuk menghindari kesalahan ini, lihat Menampilkan status volume untuk klaster DB Aurora PostgreSQL.

is_disk

Mengindikasikan apakah kegagalan injeksi terjadi pada blok logis atau simpul penyimpanan. Menentukan benar berarti kegagalan injeksi terjadi pada blok logis. Menentukan salah berarti kegagalan injeksi terjadi pada simpul penyimpanan.

time_interval

Jumlah waktu untuk mensimulasikan kegagalan disk. Interval dalam detik. Misalnya, jika nilainya adalah 20, simulasi berjalan selama 20 detik.

Menguji kepadatan disk

Anda dapat mensimulasikan kemacetan disk untuk cluster Aurora PostgreSQL DB dengan menggunakan fungsi query injeksi kesalahan. `aurora_inject_disk_congestion()`

Selama simulasi kepadatan disk, klaster DB Aurora PostgreSQL secara acak menandai segmen disk sebagai padat. Permintaan untuk segmen tersebut ditunda antara waktu penundaan minimum dan maksimum yang ditentukan untuk durasi simulasi.

Sintaksis

```
SELECT aurora_inject_disk_congestion(
   percentage_of_failure,
   index,
   is_disk,
   time_interval,
   minimum,
   maximum
);
```

Opsi

Kueri injeksi kesalahan ini mengambil parameter berikut:

percentage_of_failure

Persentase disk yang akan ditandai sebagai padat selama peristiwa kegagalan. Ini adalah nilai double antara 0 dan 100. Jika Anda menetapkan 0, tidak ada disk yang ditandai sebagai padat. Jika Anda menetapkan 100, seluruh disk akan ditandai sebagai padat.

indeks

Blok logis data atau simpul penyimpanan tertentu yang akan digunakan dalam simulasi peristiwa kegagalan.

Jika Anda melampaui rentang blok logis atau simpul penyimpanan data yang tersedia, Anda akan menerima pesan kesalahan yang memberitahukan nilai indeks maksimum yang dapat Anda tentukan. Untuk menghindari kesalahan ini, lihat Menampilkan status volume untuk klaster DB Aurora PostgreSQL.

is_disk

Mengindikasikan apakah kegagalan injeksi terjadi pada blok logis atau simpul penyimpanan. Menentukan benar berarti kegagalan injeksi terjadi pada blok logis. Menentukan salah berarti kegagalan injeksi terjadi pada simpul penyimpanan.

time_interval

Jumlah waktu untuk mensimulasikan kemacetan disk. Interval dalam detik. Misalnya, jika nilainya adalah 20, simulasi berjalan selama 20 detik.

minimum, maximum

Penundaan kepadatan minimum dan maksimum, dalam milidetik. Nilai yang valid berkisar dari 0,0 hingga 100,0 milidetik. Segmen disk yang ditandai padat akan ditunda selama jangka waktu acak dalam rentang durasi minimum dan maksimum untuk simulasi. Nilai maksimum harus lebih besar dari nilai minimum.

Menampilkan status volume untuk klaster DB Aurora PostgreSQL

Di Amazon Aurora, volume klaster DB terdiri dari kumpulan blok logis. Masing-masing blok ini merepresentasikan 10 gigabyte penyimpanan yang dialokasikan. Blok-blok ini disebut grup perlindungan.

Data di setiap grup perlindungan direplikasi di enam perangkat penyimpanan fisik, yang disebut simpul penyimpanan. Simpul penyimpanan ini dialokasikan di tiga Zona Ketersediaan (AZ) di wilayah tempat klaster DB berada. Pada gilirannya, setiap simpul penyimpanan berisi satu atau beberapa blok logis data untuk volume klaster DB. Untuk informasi selengkapnya tentang grup perlindungan dan simpul penyimpanan, lihat Memperkenalkan mesin penyimpanan Aurora dalam Blog Basis Data AWS. Untuk mempelajari selengkapnya tentang volume klaster Aurora secara umum, lihat Penyimpanan dan keandalan Amazon Aurora.

Gunakan fungsi `aurora_show_volume_status()` untuk menghasilkan variabel status server berikut:

- `Disks` — Jumlah total blok logis data untuk volume klaster DB.
- `Nodes` — Jumlah total simpul penyimpanan untuk volume klaster DB.

Anda dapat menggunakan fungsi `aurora_show_volume_status()` untuk membantu menghindari kesalahan saat menggunakan fungsi injeksi kesalahan `aurora_inject_disk_failure()`.

Fungsi injeksi kesalahan `aurora_inject_disk_failure()` menyimulasikan kegagalan seluruh simpul penyimpanan atau satu blok logis data dalam simpul penyimpanan. Dalam fungsi tersebut, tentukan nilai indeks blok logis spesifik data atau simpul penyimpanan. Namun, pernyataan tersebut akan menghasilkan kesalahan jika Anda menentukan nilai indeks yang lebih besar dari jumlah blok logis data atau simpul penyimpanan yang digunakan oleh volume klaster DB. Untuk informasi selengkapnya tentang kueri injeksi kesalahan, lihat Menguji Amazon Aurora PostgreSQL menggunakan kueri injeksi kesalahan.

> **Note**
>
> Fungsi `aurora_show_volume_status()` tersedia untuk Aurora PostgreSQL versi 10.11. Untuk informasi selengkapnya tentang versi Aurora PostgreSQL, lihat Versi rilis dan mesin Amazon Aurora PostgreSQL.

Sintaksis

```
SELECT * FROM aurora_show_volume_status();
```

Contoh

```
customer_database=> SELECT * FROM aurora_show_volume_status();
 disks | nodes
-------+-------
    96 |    45
```

Menentukan disk RAM untuk stats_temp_directory

Anda dapat menggunakan parameter Aurora PostgreSQL, `rds.pg_stat_ramdisk_size`, untuk menentukan memori sistem yang dialokasikan ke disk RAM untuk menyimpan `stats_temp_directory` PostgreSQL. Parameter disk RAM hanya tersedia di Aurora PostgreSQL 14 dan versi lebih rendah.

Di bawah beban kerja tertentu, pengaturan parameter ini dapat meningkatkan performa dan menurunkan kebutuhan IO. Untuk informasi selengkapnya tentang `stats_temp_directory`, lihat Run-time Statistics dalam dokumentasi PostgreSQL. Mulai PostgreSQL versi 15, komunitas PostgreSQL beralih menggunakan memori bersama dinamis. Jadi, `stats_temp_directory` tidak perlu diatur.

Untuk mengaktifkan disk RAM untuk `stats_temp_directory` Anda, setel parameter `rds.pg_stat_ramdisk_size` ke nilai bukan nol di grup parameter klaster DB yang digunakan oleh klaster DB Anda. Parameter ini menunjukkan MB, jadi Anda harus menggunakan nilai bilangan bulat. Ekspresi, rumus, dan fungsi tidak valid untuk parameter `rds.pg_stat_ramdisk_size`. Pastikan untuk memulai ulang klaster DB agar perubahan diterapkan. Untuk informasi tentang mengatur parameter, lihat Bekerja dengan grup parameter. Untuk informasi selengkapnya tentang memulai ulang klaster DB, lihat Mem-boot ulang klaster DB Amazon Aurora atau instans DB Amazon Aurora.

Misalnya, perintah AWS CLI berikut mengatur parameter disk RAM menjadi 256 MB.

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name db-cl-pg-ramdisk-testing \
    --parameters "ParameterName=rds.pg_stat_ramdisk_size, ParameterValue=256,
 ApplyMethod=pending-reboot"
```

Setelah Anda memulai ulang klaster DB, jalankan perintah berikut untuk melihat status `stats_temp_directory`:

```
postgres=> SHOW stats_temp_directory;
```

Perintah tersebut akan menghasilkan hal berikut:

```
stats_temp_directory
----------------------------
/rdsdbramdisk/pg_stat_tmp
(1 row)
```

Mengelola file sementara dengan PostgreSQL

Di PostgreSQL, kueri yang melakukan operasi pengurutan dan hash menggunakan memori instans untuk menyimpan hasil hingga nilai yang ditentukan dalam parameter `work_mem`. Jika memori instans tidak cukup, file sementara akan dibuat untuk menyimpan hasil. Hasil ini ditulis ke disk untuk menyelesaikan eksekusi kueri. Kemudian, file-file ini secara otomatis dihapus setelah kueri selesai. Dalam Aurora PostgreSQL, file ini berbagi penyimpanan lokal dengan file log lainnya. Anda dapat memantau ruang penyimpanan lokal klaster DB Aurora PostgreSQL dengan melihat metrik Amazon CloudWatch untuk `FreeLocalStorage`. Untuk informasi selengkapnya, lihat Memecahkan masalah penyimpanan lokal.

Anda dapat menggunakan parameter dan fungsi berikut untuk mengelola file sementara dalam instans Anda.

- **temp_file_limit** – Parameter ini membatalkan kueri apa pun yang melebihi ukuran temp_files dalam KB. Batas ini mencegah kueri apa pun berjalan tanpa henti dan menghabiskan ruang disk dengan file sementara. Anda dapat memperkirakan nilai menggunakan hasil dari parameter `log_temp_files`. Sebagai praktik terbaik, periksa perilaku beban kerja dan tetapkan batas sesuai dengan estimasi. Contoh berikut menunjukkan bagaimana kueri dibatalkan saat melampaui batas.

```
postgres=> select * from pgbench_accounts, pg_class, big_table;
```

```
ERROR: temporary file size exceeds temp_file_limit (64kB)
```

- **log_temp_files** – Parameter ini mengirimkan pesan ke postgresql.log ketika file sementara dari sebuah sesi dihapus. Parameter ini menghasilkan log setelah kueri berhasil diselesaikan. Oleh karena itu, ini mungkin tidak membantu dalam memecahkan masalah kueri yang aktif dan berjalan lama.

  Contoh berikut menunjukkan bahwa ketika kueri berhasil diselesaikan, entri dicatat dalam file postgresql.log, sedangkan file sementara dibersihkan.

```
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.5", size 140353536
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:LOG:
 temporary file: path "base/pgsql_tmp/pgsql_tmp31236.4", size 180428800
2023-02-06 23:48:35 UTC:205.251.233.182(12456):adminuser@postgres:[31236]:STATEMENT:
  select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid limit 10;
```

- **pg_ls_tmpdir** – Fungsi yang tersedia dari RDS untuk PostgreSQL 13 dan versi yang lebih baru ini memberikan visibilitas terhadap penggunaan file sementara saat ini. Kueri yang sudah selesai tidak muncul di hasil fungsi. Dalam contoh berikut, Anda dapat melihat hasil dari fungsi ini.

```
postgres=> select * from pg_ls_tmpdir();
```

```
      name       |    size    |      modification
-----------------+------------+------------------------
 pgsql_tmp8355.1 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.0 | 1072250880 | 2023-02-06 22:54:43+00
 pgsql_tmp8327.0 | 1072250880 | 2023-02-06 22:54:56+00
 pgsql_tmp8351.1 |  703168512 | 2023-02-06 22:54:56+00
 pgsql_tmp8355.0 | 1072250880 | 2023-02-06 22:54:00+00
 pgsql_tmp8328.1 |  835031040 | 2023-02-06 22:54:56+00
 pgsql_tmp8328.0 | 1072250880 | 2023-02-06 22:54:40+00
(7 rows)
```

```
postgres=> select query from pg_stat_activity where pid = 8355;

query
-------------------------------------------------------------------------------------
select a.aid from pgbench_accounts a, pgbench_accounts b where a.bid=b.bid order by
 a.bid
(1 row)
```

Nama file mencakup ID pemrosesan (PID) dari sesi yang menghasilkan file sementara. Kueri yang lebih maju, seperti pada contoh berikut, melakukan penjumlahan file sementara untuk setiap PID.

```
postgres=> select replace(left(name, strpos(name, '.')-1),'pgsql_tmp','') as pid,
 count(*), sum(size) from pg_ls_tmpdir() group by pid;
```

```
 pid  | count |   sum
------+-------+------------
 8355 |     2 | 2144501760
 8351 |     2 | 2090770432
 8327 |     1 | 1072250880
 8328 |     2 | 2144501760
```

```
(4 rows)
```

- **pg_stat_statements** – Jika Anda mengaktifkan parameter pg_stat_statement, Anda dapat melihat rata-rata penggunaan file sementara per panggilan. Anda dapat mengidentifikasi query_id dari kueri dan menggunakannya untuk memeriksa penggunaan file sementara seperti yang ditunjukkan pada contoh berikut.

```
postgres=> select queryid from pg_stat_statements where query like 'select a.aid from
 pgbench%';
```

```
       queryid
----------------------
 -7170349228837045701
(1 row)
```

```
postgres=> select queryid, substr(query,1,25), calls, temp_blks_read/calls
 temp_blks_read_per_call, temp_blks_written/calls temp_blks_written_per_call from
 pg_stat_statements where queryid = -7170349228837045701;
```

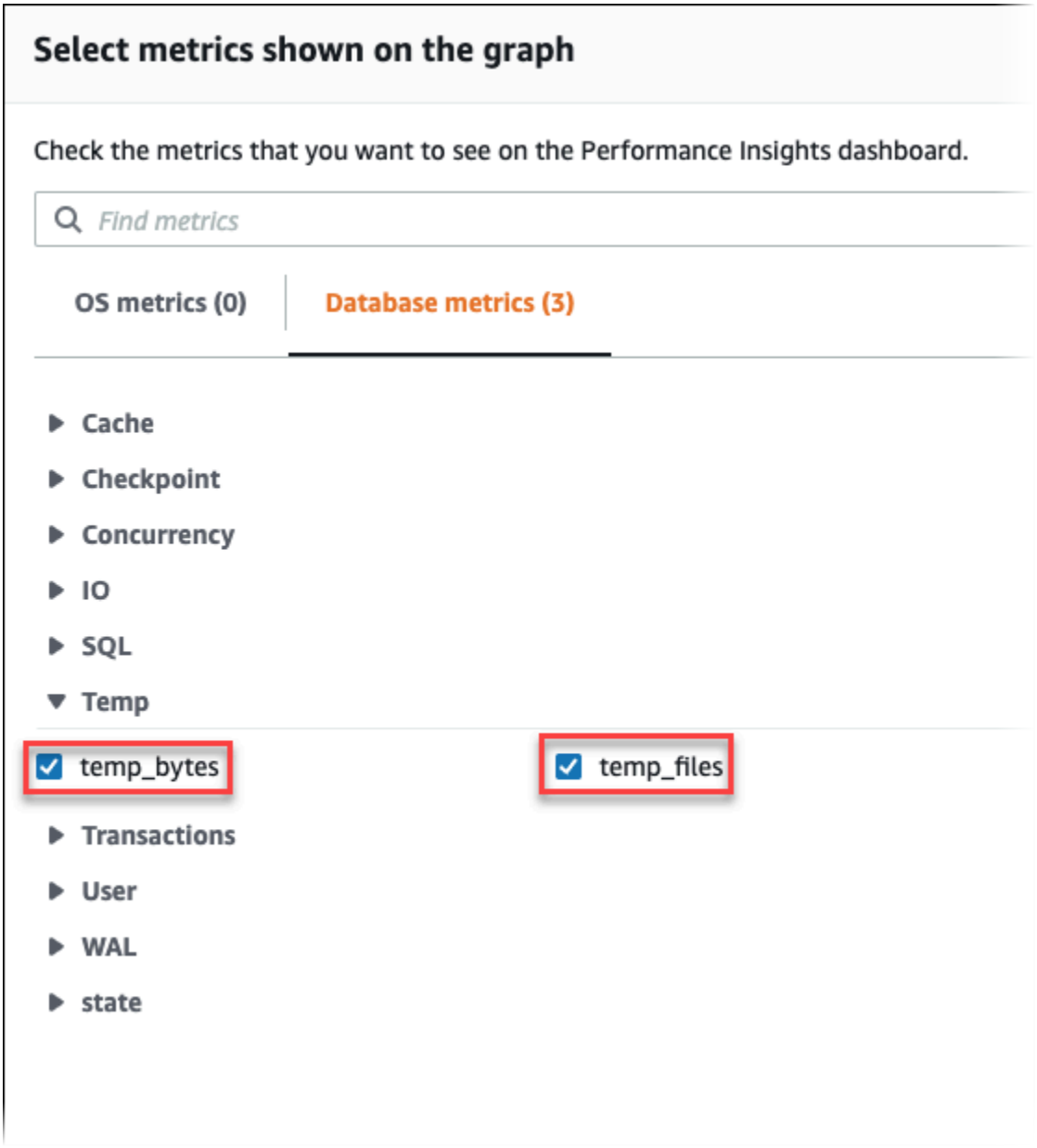
```
       queryid        |          substr           | calls | temp_blks_read_per_call |
 temp_blks_written_per_call
----------------------+---------------------------+-------+-------------------------
+----------------------------
 -7170349228837045701 | select a.aid from pgbench |    50 |                  239226 |
                     388678
(1 row)
```

- **Performance Insights** – Di dasbor Wawasan Performa, Anda dapat melihat penggunaan file sementara dengan mengaktifkan metrik temp_bytes dan temp_files. Kemudian, Anda dapat melihat rata-rata kedua metrik ini dan melihat sejauh mana kesesuaiannya dengan beban kerja kueri. Tampilan dalam Wawasan Performa tidak secara khusus menampilkan kueri yang menghasilkan file sementara. Namun, jika Anda menggabungkan Wawasan Performa dengan kueri yang ditampilkan untuk `pg_ls_tmpdir`, Anda dapat memecahkan masalah, menganalisis, dan menentukan perubahan dalam beban kerja kueri.

Untuk informasi selengkapnya tentang cara menganalisis metrik dan kueri dengan Wawasan Performa, lihat Menganalisis metrik dengan dasbor Wawasan Performa

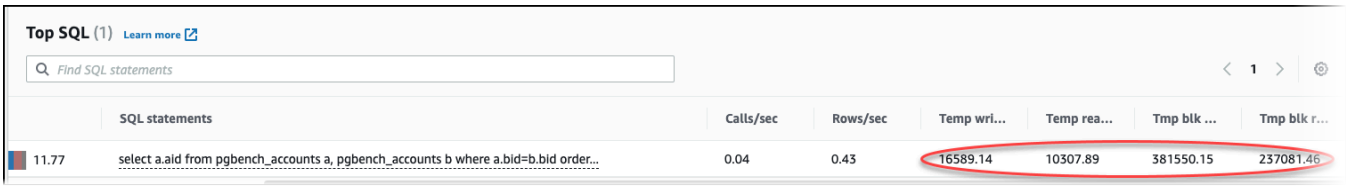
Untuk melihat penggunaan file sementara dengan Wawasan Performa

1. Di dasbor Wawasan Performa, pilih Kelola Metrik.
2. Pilih Metrik basis data, lalu pilih metrik temp_bytes dan temp_files seperti yang ditunjukkan pada gambar berikut.

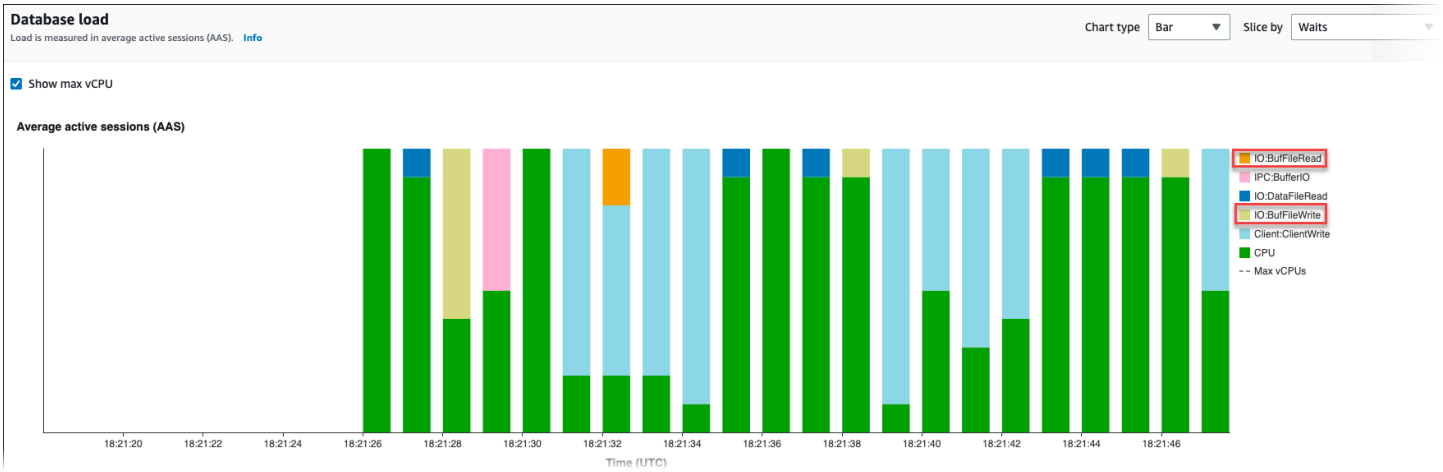


3. Di tab SQL Teratas, pilih ikon Preferensi.

4. Di jendela Preferensi, aktifkan statistik berikut agar muncul di tab SQL Teratas dan pilih Lanjutkan.
   - Temp writes/detik
   - Temp reads/detik
   - Tmp blk write/panggilan
   - Tmp blk read/panggilan
5. File sementara rusak saat digabungkan kueri yang ditampilkan untuk `pg_ls_tmpdir`, seperti yang ditunjukkan pada contoh berikut.



Peristiwa `IO:BufFileRead` dan `IO:BufFileWrite` terjadi ketika kueri teratas di beban kerja Anda sering membuat file sementara. Anda dapat menggunakan Wawasan Performa untuk mengidentifikasi kueri teratas yang menunggu pada `IO:BufFileRead` dan `IO:BufFileWrite` dengan meninjau Sesi Aktif Rata-rata (AAS) di bagian Muatan Basis Data dan SQL Teratas.



Untuk informasi selengkapnya tentang cara menganalisis kueri teratas dan muatan berdasarkan peristiwa tunggu dengan Wawasan Performa, lihat Ikhtisar tab SQL Teratas. Anda harus mengidentifikasi dan menyetel kueri yang menyebabkan peningkatan penggunaan file sementara dan peristiwa tunggu terkait. Untuk informasi selengkapnya tentang peristiwa tunggu dan remediasi ini, lihat IO:BufFileRead dan IO:BufFileWrite.

> **Note**
>
> Parameter `work_mem` mengontrol ketika operasi pengurutan kehabisan memori dan hasilnya ditulis ke dalam file sementara. Sebaiknya Anda tidak mengubah pengaturan parameter ini lebih tinggi dari nilai default karena akan memungkinkan setiap sesi basis data mengonsumsi lebih banyak memori. Selain itu, satu sesi yang melakukan penggabungan dan pengurutan kompleks dapat melakukan operasi paralel di mana setiap operasi mengonsumsi memori. Sebagai praktik terbaik, ketika Anda memiliki laporan besar dengan beberapa penggabungan dan pengurutan, atur parameter ini pada tingkat sesi dengan menggunakan perintah `SET work_mem`. Kemudian, perubahan hanya diterapkan pada sesi saat ini dan tidak mengubah nilai secara global.

Menyetel dengan peristiwa tunggu di Aurora PostgreSQL

Peristiwa tunggu adalah alat penyetelan penting untuk Aurora PostgreSQL. Ketika Anda dapat mengetahui mengapa sesi menunggu sumber daya dan apa yang sedang dilakukan, Anda lebih mampu mengurangi kemacetan. Anda dapat menggunakan informasi di bagian ini untuk menemukan kemungkinan penyebab dan tindakan korektif. Sebelum mempelajari bagian ini, kami sangat menyarankan agar Anda memahami konsep dasar Aurora, terutama topik-topik berikut:

- Penyimpanan dan keandalan Amazon Aurora
- Mengelola performa dan penskalaan untuk klaster DB Aurora

> **Important**
>
> Peristiwa tunggu di bagian ini khusus untuk Aurora PostgreSQL. Gunakan informasi di bagian ini untuk menyetel Amazon Aurora saja, bukan RDS for PostgreSQL.
> Beberapa peristiwa tunggu di bagian ini tidak memiliki analog dalam versi sumber terbuka dari mesin basis data ini. Peristiwa tunggu lainnya memiliki nama yang sama dengan peristiwa di mesin sumber terbuka, tetapi berperilaku berbeda. Misalnya, penyimpanan Amazon Aurora bekerja secara berbeda dari penyimpanan sumber terbuka, sehingga peristiwa tunggu terkait penyimpanan menunjukkan kondisi sumber daya yang berbeda.

Topik

- Konsep penting untuk penyetelan Aurora PostgreSQL
- Peristiwa tunggu Aurora PostgreSQL
- Client:ClientRead
- Client:ClientWrite
- CPU
- IO:BufFileRead dan IO:BufFileWrite
- IO:DataFileRead
- IO:XactSync
- IPC:DamRecordTxAck
- Lock:advisory
- Lock:extend
- Lock:Relation
- Lock:transactionid
- Lock:tuple
- LWLock:buffer_content (BufferContent)
- LWLock:buffer_mapping
- LWLock:BufferIO (IPC:BufferIO)
- LWLock:lock_manager
- LWLock: MultiXact
- Timeout:PgSleep

Konsep penting untuk penyetelan Aurora PostgreSQL

Sebelum Anda menyetel basis data Aurora PostgreSQL, pastikan untuk mempelajari apa itu peristiwa tunggu dan mengapa itu terjadi. Juga tinjau memori dasar dan arsitektur disk Aurora PostgreSQL. Untuk diagram arsitektur yang bermanfaat, lihat wikibook PostgreSQL.

Topik

- Peristiwa tunggu Aurora PostgreSQL
- Memori Aurora PostgreSQL

- Proses Aurora PostgreSQL

Peristiwa tunggu Aurora PostgreSQL

Peristiwa tunggu menunjukkan sumber daya yang sedang menunggu sesi. Misalnya, peristiwa tunggu `Client:ClientRead` terjadi ketika Aurora PostgreSQL menunggu untuk menerima data dari klien. Sumber daya khas yang ditunggu sesi meliputi yang berikut:

- Akses single-threaded ke buffer, misalnya, saat sesi mencoba mengubah buffer
- Baris yang saat ini dikunci oleh sesi lain
- Pembacaan file data
- Penulisan file log

Misalnya, untuk memenuhi kueri, sesi mungkin melakukan pemindaian tabel lengkap. Jika data belum ada dalam memori, sesi menunggu disk I/O selesai. Ketika buffer dibaca ke dalam memori, sesi mungkin perlu menunggu karena sesi lain mengakses buffer yang sama. Basis data mencatat kejadian tunggu dengan menggunakan peristiwa tunggu yang telah ditentukan sebelumnya. Peristiwa tersebut dikelompokkan ke dalam kategori.

Peristiwa tunggu tidak dengan sendirinya menunjukkan masalah kinerja. Misalnya, jika data yang diminta tidak ada dalam memori, membaca data dari disk diperlukan. Jika satu sesi mengunci baris untuk pembaruan, sesi lain menunggu baris dibuka sehingga dapat memperbaruinya. Commit perlu menunggu penulisan ke file log selesai. Peristiwa tunggu merupakan bagian integral dari fungsi normal basis data.

Sejumlah besar peristiwa tunggu biasanya menunjukkan masalah kinerja. Dalam kasus seperti itu, Anda dapat menggunakan data peristiwa tunggu untuk menentukan tempat sesi menghabiskan waktu. Misalnya, jika laporan yang biasanya berjalan dalam hitungan menit sekarang berjalan selama berjam-jam, Anda dapat mengidentifikasi peristiwa tunggu yang berkontribusi paling besar terhadap total waktu tunggu. Jika Anda dapat menentukan penyebab peristiwa tunggu teratas, terkadang Anda dapat membuat perubahan yang meningkatkan kinerja. Misalnya, jika sesi Anda menunggu pada baris yang telah dikunci oleh sesi lain, Anda dapat mengakhiri sesi penguncian.

Memori Aurora PostgreSQL

Memori Aurora PostgreSQL dibagi menjadi bersama dan lokal.

Topik

- Memori bersama di Aurora PostgreSQL
- Memori lokal di Aurora PostgreSQL

Memori bersama di Aurora PostgreSQL

Aurora PostgreSQL mengalokasikan memori bersama saat instans dimulai. Memori bersama dibagi menjadi beberapa subarea. Setelah itu, Anda dapat menemukan deskripsi yang paling penting.

Topik

- Buffer bersama
- Buffer log write ahead (WAL)

Buffer bersama

Kolam buffer bersama adalah area memori Aurora PostgreSQL yang menampung semua halaman yang sedang atau telah digunakan oleh koneksi aplikasi. Halaman adalah versi memori dari blok disk. Kolam buffer bersama menyimpan blok data yang dibaca dari disk. Kumpulan tersebut mengurangi kebutuhan untuk membaca ulang data dari disk, membuat basis data beroperasi lebih efisien.

Setiap tabel dan indeks disimpan sebagai array halaman dengan ukuran tetap. Setiap blok berisi beberapa tuple, yang sesuai dengan baris. Tuple dapat disimpan di halaman mana pun.

Kolam buffer bersama memiliki memori terbatas. Jika permintaan baru memerlukan halaman yang tidak ada dalam memori, dan tidak ada lagi memori, Aurora PostgreSQL mengosongkan halaman yang jarang digunakan untuk mengakomodasi permintaan tersebut. Kebijakan pengosongan diimplementasikan oleh algoritma clock sweep.

Parameter `shared_buffers` menentukan berapa banyak memori server mendedikasikan untuk caching data.

Buffer log write ahead (WAL)

Buffer log write-ahead (WAL) menyimpan data transaksi yang kemudian ditulis Aurora PostgreSQL ke penyimpanan persisten. Menggunakan mekanisme WAL, Aurora PostgreSQL dapat melakukan hal berikut:

- Memulihkan data setelah kegagalan
- Mengurangi disk I/O dengan menghindari penulisan ke disk yang sering

Ketika klien mengubah data, Aurora PostgreSQL menulis perubahan ke buffer WAL. Ketika klien mengeluarkan `COMMIT`, proses penulis WAL menulis data transaksi ke file WAL.

Parameter `wal_level` menentukan berapa banyak informasi yang ditulis ke WAL.

Memori lokal di Aurora PostgreSQL

Setiap proses backend mengalokasikan memori lokal untuk pemrosesan kueri.

Topik

- Area memori kerja
- Area memori kerja pemeliharaan
- Area buffer sementara

Area memori kerja

Area memori kerja menyimpan data sementara untuk kueri yang melakukan pengurutan dan hash. Misalnya, kueri dengan klausa `ORDER BY` melakukan pengurutan. Kueri menggunakan tabel hash dalam gabungan dan agregasi hash.

Parameter `work_mem` jumlah memori yang akan digunakan oleh operasi pengurutan internal dan tabel hash sebelum menulis ke file disk sementara. Nilai default-nya adalah 4 MB. Beberapa sesi dapat berjalan secara bersamaan, dan setiap sesi dapat menjalankan operasi pemeliharaan secara paralel. Karena alasan ini, total memori kerja yang digunakan dapat menjadi kelipatan dari pengaturan `work_mem`.

Area memori kerja pemeliharaan

Area memori kerja pemeliharaan menyimpan data untuk operasi pemeliharaan. Operasi ini termasuk memvakum, membuat indeks, dan menambahkan kunci asing.

Parameter `maintenance_work_mem` menentukan jumlah maksimum memori yang akan digunakan oleh operasi pemeliharaan. Nilai default-nya adalah 64 MB. Sebuah sesi basis data hanya dapat menjalankan satu operasi pemeliharaan dalam satu waktu.

Area buffer sementara

Area buffer sementara menyimpan tabel sementara untuk setiap sesi basis data.

Setiap sesi mengalokasikan buffer sementara sesuai kebutuhan hingga batas yang Anda tentukan. Saat sesi berakhir, server menghapus buffer.

Parameter `temp_buffers` mengatur jumlah maksimum buffer sementara yang digunakan oleh setiap sesi. Sebelum penggunaan pertama tabel sementara dalam sesi, Anda dapat mengubah nilai `temp_buffers`.

Proses Aurora PostgreSQL

Aurora PostgreSQL menggunakan beberapa proses.

Topik

- Proses postmaster
- Proses backend
- Proses latar belakang

Proses postmaster

Proses postmaster adalah proses pertama yang dimulai ketika Anda memulai Aurora PostgreSQL. Proses postmaster memiliki tanggung jawab utama berikut:

- Membagi dan memantau proses latar belakang
- Menerima permintaan autentikasi dari proses klien, dan mengautentikasi mereka sebelum mengizinkan basis data untuk melayani permintaan

Proses backend

Jika postmaster mengautentikasi permintaan klien, postmaster akan melakukan proses backend baru, juga disebut proses postgres. Satu proses klien terhubung ke persis satu proses backend. Proses klien dan proses backend berkomunikasi secara langsung tanpa intervensi oleh proses postmaster.

Proses latar belakang

Proses postmaster membagi beberapa proses yang melakukan tugas backend yang berbeda. Beberapa hal yang lebih penting termasuk berikut ini:

- Penulis WAL

  Aurora PostgreSQL menulis data dalam buffer WAL (log write ahead) ke file log. Prinsip log write ahead adalah bahwa basis data tidak dapat menulis perubahan pada file data sampai setelah basis data menulis catatan log yang menjelaskan perubahan tersebut ke disk. Mekanisme WAL

mengurangi disk I/O, dan memungkinkan Aurora PostgreSQL untuk menggunakan log untuk memulihkan basis data setelah kegagalan.

- Penulis latar belakang

  Proses ini secara berkala menulis halaman kotor (diubah) dari buffer memori ke file data. Sebuah halaman menjadi kotor ketika proses backend memodifikasinya dalam memori.

- Daemon autovacuum

  Daemon terdiri dari hal-hal berikut:

  - Peluncur autovacuum
  - Proses pekerja autovacuum

  Saat autovacuum diaktifkan, autovacuum tersebut memeriksa tabel yang memiliki banyak tuple yang dimasukkan, diperbarui, atau dihapus. Daemon memiliki tanggung jawab sebagai berikut:

  - Memulihkan atau menggunakan kembali ruang disk yang ditempati oleh baris yang diperbarui atau dihapus
  - Memperbarui statistik yang digunakan oleh perencana
  - Melindungi dari kehilangan data lama karena wraparound ID transaksi

  Fitur autovacuum mengotomatiskan eksekusi `VACUUM` dan perintah `ANALYZE`. `VACUUM` memiliki varian berikut: standar dan penuh. Vakum standar berjalan secara paralel dengan operasi basis data lainnya. `VACUUM FULL` membutuhkan kunci eksklusif di tabel yang sedang dikerjakannya. Dengan demikian, tidak dapat berjalan secara paralel dengan operasi yang mengakses tabel yang sama. `VACUUM` membuat sejumlah besar lalu lintas I/O, yang dapat menyebabkan kinerja buruk untuk sesi aktif lainnya.

Peristiwa tunggu Aurora PostgreSQL

Tabel berikut mencantumkan peristiwa tunggu untuk Aurora PostgreSQL yang paling sering menunjukkan masalah kinerja, dan meringkas penyebab paling umum dan tindakan korektif. Peristiwa tunggu berikut adalah subset dari daftar di Peristiwa tunggu Amazon Aurora PostgreSQL.

Peristiwa tunggu	Definisi
Client:ClientRead	Peristiwa ini terjadi ketika Aurora PostgreSQL menunggu untuk menerima data dari klien.

Peristiwa tunggu	Definisi
Client:ClientWrite	Peristiwa ini terjadi ketika Aurora PostgreSQL menunggu untuk menulis data ke klien.
CPU	Peristiwa ini terjadi ketika thread aktif di CPU atau sedang menunggu CPU.
IO:BufFileRead dan IO:BufFileWrite	Peristiwa ini terjadi ketika Aurora PostgreSQL membuat file sementara.
IO:DataFileRead	Peristiwa ini terjadi ketika koneksi menunggu pada proses backend untuk membaca halaman yang diperlukan dari penyimpanan karena halaman tidak tersedia dalam memori bersama.
IO:XactSync	Peristiwa ini terjadi ketika basis data menunggu subsistem penyimpanan Aurora untuk mengakui commit transaksi reguler, atau commit atau pengembalian transaksi yang disiapkan.
IPC:DamRecordTxAck	Peristiwa ini terjadi ketika Aurora PostgreSQL dalam sesi menggunakan aliran aktivitas basis data menghasilkan peristiwa aliran aktivitas, lalu menunggu peristiwa tersebut menjadi tahan lama.
Lock:advisory	Peristiwa ini terjadi ketika aplikasi PostgreSQL menggunakan kunci untuk mengoordinasikan aktivitas di beberapa sesi.
Lock:extend	Peristiwa ini terjadi ketika proses backend menunggu untuk mengunci relasi untuk memperpanjangnya selagi proses lain memiliki kunci pada relasi itu untuk tujuan yang sama.
Lock:Relation	Peristiwa ini terjadi ketika kueri menunggu untuk memperoleh kunci pada tabel atau tampilan yang saat ini dikunci oleh transaksi lain.

Peristiwa tunggu	Definisi
Lock:transactionid	Peristiwa ini terjadi ketika transaksi sedang menunggu kunci tingkat baris.
Lock:tuple	Peristiwa ini terjadi ketika proses backend menunggu untuk mendapatkan kunci pada tuple.
LWLock:buffer_content (BufferContent)	Peristiwa ini terjadi ketika suatu sesi menunggu untuk membaca atau menulis halaman data di memori selagi sesi lain mengunci halaman tersebut untuk penulisan.
LWLock:buffer_mapping	Peristiwa ini terjadi ketika sesi menunggu untuk mengaitkan blok data dengan buffer di kolam buffer bersama.
LWLock:BufferIO (IPC:BufferIO)	Peristiwa ini terjadi ketika Aurora PostgreSQL atau RDS for PostgreSQL sedang menunggu proses lain untuk menyelesaikan operasi input/output (I/O) mereka ketika secara bersamaan mencoba mengakses halaman.
LWLock:lock_manager	Peristiwa ini terjadi ketika mesin Aurora PostgreSQL mempertahankan area memori kunci bersama untuk mengalokasikan, memeriksa, dan mendealokasikan kunci ketika kunci jalur cepat tidak memungkinkan.
LWLock: MultiXact	Jenis peristiwa ini terjadi ketika Aurora PostgreSQL menjaga sesi terbuka untuk menyelesaikan beberapa transaksi yang melibatkan baris yang sama dalam tabel. Peristiwa tunggu menunjukkan aspek mana dari beberapa pemrosesan transaksi yang menghasilkan peristiwa tunggu, yaitu, LWLock:MultiXactOffsetSLRU, LWLock:MultiXactOffsetBuffer, LWLock:MultiXactMemberSLRU, atau LWLock:MultiXactMemberBuffer.

Peristiwa tunggu	Definisi
Timeout:PgSleep	Peristiwa ini terjadi ketika proses server telah memanggil fungsi `pg_sleep` dan menunggu batas waktu tidur berakhir.

Client:ClientRead

Peristiwa `Client:ClientRead` terjadi saat Aurora PostgreSQL menunggu untuk menerima data dari klien.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk Aurora PostgreSQL versi 10 dan yang lebih tinggi.

Konteks

Klaster DB Aurora PostgreSQL menunggu untuk menerima data dari klien. Klaster DB Aurora PostgreSQL harus menerima data dari klien agar dapat mengirim lebih banyak data ke klien. Waktu saat klaster menunggu sebelum menerima data dari klien adalah peristiwa `Client:ClientRead`.

Kemungkinan penyebab peningkatan peristiwa tunggu

Penyebab umum peristiwa `Client:ClientRead` muncul dalam peristiwa tunggu teratas mencakup yang berikut:

Peningkatan latensi jaringan

Mungkin terdapat peningkatan latensi jaringan antara klaster DB Aurora PostgreSQL dan klien. Latensi jaringan yang lebih tinggi akan menambah waktu yang diperlukan agar klaster DB dapat menerima data dari klien.

Peningkatan beban pada klien

Mungkin terdapat tekanan CPU atau saturasi jaringan pada klien. Peningkatan beban pada klien dapat menunda transmisi data dari klien ke klaster DB Aurora PostgreSQL.

Round trip jaringan yang berlebihan

Sejumlah besar round trip jaringan antara klaster DB Aurora PostgreSQL dan klien dapat menunda transmisi data dari klien ke klaster DB Aurora PostgreSQL.

Operasi penyalinan besar

Selama operasi penyalinan, data ditransfer dari sistem file klien ke klaster DB Aurora PostgreSQL. Mengirim data dalam jumlah besar ke klaster DB dapat menunda transmisi data dari klien ke klaster DB.

Koneksi klien idle

Koneksi ke instans DB Aurora PostgreSQL berada dalam status idle dalam transaksi dan sedang menunggu klien mengirim lebih banyak data atau mengeluarkan perintah. Status ini dapat menyebabkan peningkatan peristiwa `Client:ClientRead`.

PgBouncer digunakan untuk penggabungan koneksi

PgBouncer memiliki pengaturan konfigurasi jaringan tingkat rendah `pkt_buf`, yang diatur ke 4.096 secara default. Jika beban kerja mengirimkan paket kueri lebih besar dari 4.096 byte melalui PgBouncer, sebaiknya tingkatkan pengaturan `pkt_buf` ke 8.192. Jika pengaturan baru tidak mengurangi jumlah peristiwa `Client:ClientRead`, sebaiknya tingkatkan pengaturan `pkt_buf` ke nilai yang lebih besar, seperti 16.384 atau 32.768. Jika teks kueri berukuran besar, pengaturan yang lebih besar dapat sangat membantu.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Menempatkan klien di subnet VPC dan Zona Ketersediaan yang sama seperti klaster
- Menskalakan klien
- Menggunakan instans generasi saat ini
- Meningkatkan bandwidth jaringan
- Memantau maksimum untuk performa jaringan

- Memantau transaksi dalam status "idle dalam transaksi"

Menempatkan klien di subnet VPC dan Zona Ketersediaan yang sama seperti klaster

Untuk mengurangi latensi jaringan dan meningkatkan throughput jaringan, tempatkan klien di subnet cloud privat virtual (VPC) dan Zona Ketersediaan yang sama seperti klaster DB Aurora PostgreSQL. Pastikan klien secara geografis berada sedekat mungkin dengan klaster DB.

Menskalakan klien

Dengan menggunakan Amazon CloudWatch atau metrik host lainnya, ketahui apakah klien Anda saat ini dibatasi oleh CPU atau bandwidth jaringan, atau keduanya. Jika klien dibatasi, skalakan klien sesuai yang diperlukan.

Menggunakan instans generasi saat ini

Dalam kasus tertentu, Anda mungkin tidak menggunakan kelas instans DB yang mendukung frame jumbo. Jika Anda menjalankan aplikasi di Amazon EC2, coba gunakan instans generasi saat ini untuk klien. Selain itu, konfigurasikan maximum transmission unit (MTU) pada sistem operasi klien. Teknik ini dapat mengurangi jumlah round trip jaringan dan meningkatkan throughput jaringan. Untuk informasi selengkapnya, lihat Frame jumbo (9001 MTU) dalam Panduan Pengguna Amazon EC2 untuk Instans Linux.

Untuk informasi tentang kelas instans DB, lihat Kelas instans DB Aurora. Untuk mengetahui kelas instans DB yang setara dengan jenis instans Amazon EC2, tempatkan `db.` sebelum nama jenis instans Amazon EC2. Misalnya, instans Amazon EC2 `r5.8xlarge` setara dengan kelas instans DB `db.r5.8xlarge`.

Meningkatkan bandwidth jaringan

Gunakan metrik Amazon CloudWatch `NetworkReceiveThroughput` dan `NetworkTransmitThroughput` untuk memantau lalu lintas jaringan masuk dan keluar pada klaster DB. Metrik ini dapat membantu mengetahui apakah bandwidth jaringan cukup untuk beban kerja Anda.

Jika tidak, tingkatkan bandwidth jaringan. Jika klien AWS atau instans DB Anda mencapai batas bandwidth jaringan, satu-satunya cara untuk meningkatkan bandwidth adalah dengan meningkatkan ukuran instans DB Anda.

Untuk informasi selengkapnya tentang metrik CloudWatch, lihat CloudWatch Metrik Amazon untuk Amazon Aurora.

Memantau maksimum untuk performa jaringan

Jika Anda menggunakan klien Amazon EC2, Amazon EC2 akan menyediakan maksimum untuk metrik performa jaringan, termasuk bandwidth jaringan masuk dan keluar agregat. Amazon EC2 juga menyediakan pelacakan koneksi untuk memastikan paket dikembalikan sebagaimana diharapkan dan akses layanan tautan-lokal untuk layanan seperti Sistem Nama Domain (DNS). Untuk memantau maksimum ini, gunakan driver jaringan yang ditingkatkan saat ini dan pantau performa jaringan untuk klien Anda.

Untuk informasi selengkapnya, lihat Memantau performa jaringan untuk instans Amazon EC2 dalam Panduan Pengguna Amazon EC2 untuk Instans Linux dan Memantau performa jaringan untuk instans Amazon EC2 dalam Panduan Pengguna Amazon EC2 untuk Instans Windows.

Memantau transaksi dalam status "idle dalam transaksi"

Periksa apakah Anda memiliki peningkatan jumlah koneksi `idle in transaction`. Untuk melakukannya, pantau kolom `state` dalam tabel `pg_stat_activity`. Anda mungkin dapat mengidentifikasi sumber koneksi dengan menjalankan kueri yang mirip dengan yang berikut.

```
select client_addr, state, count(1) from pg_stat_activity
where state like 'idle in transaction%'
group by 1,2
order by 3 desc
```

Client:ClientWrite

Peristiwa `Client:ClientWrite` terjadi saat Aurora PostgreSQL menunggu untuk menulis data ke klien.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk Aurora PostgreSQL versi 10 dan yang lebih tinggi.

Konteks

Proses klien harus membaca semua data yang diterima dari klaster DB Aurora PostgreSQL sebelum klaster tersebut dapat mengirim lebih banyak data. Waktu saat klaster menunggu sebelum mengirim lebih banyak data ke klien adalah peristiwa `Client:ClientWrite`.

Throughput jaringan yang berkurang antara klaster DB Aurora PostgreSQL dan klien dapat menyebabkan peristiwa ini. Tekanan CPU dan saturasi jaringan pada klien juga dapat menyebabkan peristiwa ini. Tekanan CPU adalah saat CPU sepenuhnya digunakan dan ada tugas yang menunggu waktu CPU. Saturasi jaringan adalah saat jaringan antara basis data dan klien membawa lebih banyak data dari yang dapat ditanganinya.

Kemungkinan penyebab peningkatan peristiwa tunggu

Penyebab umum peristiwa `Client:ClientWrite` muncul dalam peristiwa tunggu teratas mencakup yang berikut:

Peningkatan latensi jaringan

Mungkin terdapat peningkatan latensi jaringan antara klaster DB Aurora PostgreSQL dan klien. Latensi jaringan yang lebih tinggi akan menambah waktu yang diperlukan klien untuk menerima data.

Peningkatan beban pada klien

Mungkin terdapat tekanan CPU atau saturasi jaringan pada klien. Peningkatan beban pada klien menunda penerimaan data dari klaster DB Aurora PostgreSQL.

Data dalam volume besar yang dikirim ke klien

Klaster DB Aurora PostgreSQL mungkin mengirimkan data dalam jumlah besar ke klien. Klien mungkin tidak dapat menerima data secepat klaster mengirimkannya. Aktivitas seperti penyalinan tabel besar dapat mengakibatkan peningkatan peristiwa `Client:ClientWrite`.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Menempatkan klien di subnet VPC dan Zona Ketersediaan yang sama seperti klaster
- Menggunakan instans generasi saat ini

- Mengurangi jumlah data yang dikirim ke klien
- Menskalakan klien

Menempatkan klien di subnet VPC dan Zona Ketersediaan yang sama seperti klaster

Untuk mengurangi latensi jaringan dan meningkatkan throughput jaringan, tempatkan klien di subnet cloud privat virtual (VPC) dan Zona Ketersediaan yang sama seperti klaster DB Aurora PostgreSQL.

Menggunakan instans generasi saat ini

Dalam kasus tertentu, Anda mungkin tidak menggunakan kelas instans DB yang mendukung frame jumbo. Jika Anda menjalankan aplikasi di Amazon EC2, coba gunakan instans generasi saat ini untuk klien. Selain itu, konfigurasikan maximum transmission unit (MTU) pada sistem operasi klien. Teknik ini dapat mengurangi jumlah round trip jaringan dan meningkatkan throughput jaringan. Untuk informasi selengkapnya, lihat Frame jumbo (9001 MTU) dalam Panduan Pengguna Amazon EC2 untuk Instans Linux.

Untuk informasi tentang kelas instans DB, lihat Kelas instans DB Aurora. Untuk mengetahui kelas instans DB yang setara dengan jenis instans Amazon EC2, tempatkan `db.` sebelum nama jenis instans Amazon EC2. Misalnya, instans Amazon EC2 `r5.8xlarge` setara dengan kelas instans DB `db.r5.8xlarge`.

Mengurangi jumlah data yang dikirim ke klien

Jika memungkinkan, sesuaikan aplikasi Anda untuk mengurangi jumlah data yang dikirim klaster DB Aurora PostgreSQL ke klien. Penyesuaian ini akan mengurangi pertentangan CPU dan jaringan pada klien.

Menskalakan klien

Dengan menggunakan Amazon CloudWatch atau metrik host lainnya, ketahui apakah klien Anda saat ini dibatasi oleh CPU atau bandwidth jaringan, atau keduanya. Jika klien dibatasi, skalakan klien sesuai yang diperlukan.

CPU

Peristiwa ini terjadi saat thread aktif di CPU atau sedang menunggu CPU.

Topik

- Versi mesin yang didukung

- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini relevan untuk Aurora PostgreSQL versi 9.6 dan yang lebih tinggi.

Konteks

CPU (Central Processing Unit) adalah komponen komputer yang menjalankan instruksi. Misalnya, instruksi CPU melakukan operasi aritmetika dan bertukar data dalam memori. Jika kueri meningkatkan jumlah instruksi yang dilakukannya melalui mesin basis data, waktu yang dihabiskan untuk menjalankan kueri akan meningkat. Penjadwalan CPU memberikan waktu CPU untuk suatu proses. Penjadwalan diatur oleh kernel sistem operasi.

Topik

- Bagaimana cara mengetahui kapan peristiwa tunggu ini terjadi?
- Metrik DBLoadCPU
- Metrik os.cpuUtilization
- Kemungkinan penyebab penjadwalan CPU

Bagaimana cara mengetahui kapan peristiwa tunggu ini terjadi?

Peristiwa tunggu `CPU` ini menunjukkan bahwa proses backend aktif di CPU atau sedang menunggu CPU. Anda mengetahuinya terjadi saat kueri menunjukkan informasi berikut:

- Kolom `pg_stat_activity.state` memiliki nilai `active`.
- Kolom `wait_event_type` dan `wait_event` di `pg_stat_activity` adalah `null`.

Untuk melihat proses backend yang menggunakan atau menunggu CPU, jalankan kueri berikut.

```
SELECT *
FROM   pg_stat_activity
WHERE  state = 'active'
AND    wait_event_type IS NULL
AND    wait_event IS NULL;
```

Metrik DBLoadCPU

Metrik Wawasan Performa untuk CPU adalah `DBLoadCPU`. Nilai untuk `DBLoadCPU` dapat berbeda dengan nilai untuk metrik Amazon CloudWatch `CPUUtilization`. Metrik CloudWatch dikumpulkan dari HyperVisor untuk instans basis data.

Metrik os.cpuUtilization

Metrik sistem operasi Wawasan Performa memberikan informasi terperinci tentang pemanfaatan CPU. Misalnya, Anda dapat menampilkan metrik berikut:

- `os.cpuUtilization.nice.avg`
- `os.cpuUtilization.total.avg`
- `os.cpuUtilization.wait.avg`
- `os.cpuUtilization.idle.avg`

Wawasan Performa melaporkan penggunaan CPU oleh mesin basis data sebagai `os.cpuUtilization.nice.avg`.

Kemungkinan penyebab penjadwalan CPU

Dari perspektif sistem operasi, CPU aktif saat tidak menjalankan thread idle. CPU aktif saat menjalankan komputasi dan saat menunggu pada memori I/O. Jenis I/O ini mendominasi beban kerja basis data pada umumnya.

Proses cenderung menunggu untuk dijadwalkan pada CPU saat kondisi berikut terpenuhi:

- Metrik `CPUUtilization` CloudWatch mendekati 100%.
- Beban rata-rata lebih besar dari jumlah vCPU, yang menunjukkan beban berat. Anda dapat menemukan metrik `loadAverageMinute` di bagian metrik OS dalam Wawasan Performa.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa tunggu CPU terjadi lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

Topik

- Kemungkinan penyebab lonjakan mendadak
- Kemungkinan penyebab frekuensi tinggi jangka panjang

- Corner case

Kemungkinan penyebab lonjakan mendadak

Penyebab lonjakan mendadak yang paling memungkinkan adalah sebagai berikut:

- Aplikasi Anda membuka terlalu banyak koneksi bersamaan ke basis data. Skenario ini dikenal sebagai "storm koneksi".
- Beban kerja aplikasi berubah dengan salah satu cara berikut:
  - Kueri baru
  - Peningkatan ukuran set data
  - Pemeliharaan atau pembuatan indeks
  - Fungsi baru
  - Operator baru
  - Peningkatan eksekusi kueri paralel
- Rencana eksekusi kueri Anda telah berubah. Dalam kasus tertentu, perubahan dapat menyebabkan peningkatan buffer. Misalnya, kueri sekarang menggunakan pemindaian berurutan saat sebelumnya menggunakan indeks. Dalam hal ini, kueri membutuhkan lebih banyak CPU untuk mencapai tujuan yang sama.

Kemungkinan penyebab frekuensi tinggi jangka panjang

Berikut adalah penyebab paling memungkinkan dari peristiwa yang berulang dalam jangka waktu lama:

- Terlalu banyak proses backend yang berjalan bersamaan pada CPU. Proses-proses ini dapat menjadi pekerja paralel.
- Kueri berperforma suboptimal karena membutuhkan buffer dalam jumlah besar.

Corner case

Jika tidak ada kemungkinan penyebab yang merupakan penyebab sebenarnya, situasi berikut mungkin terjadi:

- CPU menukar proses masuk dan keluar.
- Peralihan konteks CPU telah meningkat.

- Kode Aurora PostgreSQL tidak memiliki peristiwa tunggu.

Tindakan

Jika peristiwa tunggu CPU mendominasi aktivitas basis data, hal tersebut tidak selalu menunjukkan adanya masalah performa. Tanggapi peristiwa ini hanya saat performa menurun.

Topik

- Menyelidiki apakah basis data menyebabkan peningkatan CPU
- Mengetahui apakah jumlah koneksi meningkat
- Merespons perubahan beban kerja

Menyelidiki apakah basis data menyebabkan peningkatan CPU

Periksa metrik `os.cpuUtilization.nice.avg` dalam Wawasan Performa. Jika nilai ini jauh lebih kecil daripada penggunaan CPU, proses non-basis data adalah kontributor utama ke CPU.

Mengetahui apakah jumlah koneksi meningkat

Periksa metrik `DatabaseConnections` di Amazon CloudWatch. Tindakan Anda bergantung pada apakah jumlahnya meningkat atau menurun selama periode peningkatan peristiwa tunggu CPU.

Koneksi meningkat

Jika jumlah koneksi meningkat, bandingkan jumlah proses backend yang mengonsumsi CPU terhadap jumlah vCPU. Skenario berikut mungkin terjadi:

- Jumlah proses backend yang mengonsumsi CPU lebih kecil dari jumlah vCPU.

  Dalam hal ini, jumlah koneksi tidak menjadi masalah. Namun, Anda mungkin masih mencoba mengurangi pemanfaatan CPU.
- Jumlah proses backend yang mengonsumsi CPU lebih besar dari jumlah vCPU.

  Jika demikian, pertimbangkan opsi berikut:
  - Kurangi jumlah proses backend yang terhubung ke basis data Anda. Misalnya, terapkan solusi penggabungan koneksi seperti Proksi RDS. Untuk mempelajari selengkapnya, lihat Menggunakan Proksi Amazon RDS untuk Aurora.
  - Tingkatkan ukuran instans untuk mendapatkan jumlah vCPU yang lebih tinggi.

- Jika berlaku, arahkan ulang beberapa beban kerja hanya-baca ke simpul pembaca.

Koneksi tidak meningkat

Periksa metrik `blks_hit` dalam Wawasan Performa. Cari korelasi antara peningkatan `blks_hit` dan penggunaan CPU. Skenario berikut mungkin terjadi:

- Penggunaan CPU dan `blks_hit` berkorelasi.

  Dalam hal ini, temukan pernyataan SQL teratas yang terkait dengan penggunaan CPU, lalu cari perubahan rencana. Anda dapat menggunakan salah satu teknik berikut:

  - Jelaskan rencana secara manual, lalu bandingkan dengan rencana eksekusi yang diperkirakan.
  - Cari peningkatan hit blokir per detik dan hit blokir lokal per detik. Di bagian SQL Teratas pada dasbor Wawasan Performa, pilih Preferensi.
- Penggunaan CPU dan `blks_hit` tidak berkorelasi.

  Jika demikian, ketahui apakah salah satu hal berikut terjadi:

  - Aplikasi ini dengan cepat terhubung ke dan memutuskan sambungan dari basis data.

    Jalankan diagnosis perilaku ini dengan mengaktifkan `log_connections` dan `log_disconnections`, lalu menganalisis log PostgreSQL. Pertimbangkan untuk menggunakan penganalisis log `pgbadger`. Untuk informasi selengkapnya, lihat https://github.com/darold/pgbadger.
  - OS kelebihan beban.

    Dalam hal ini, Wawasan Performa menunjukkan bahwa proses backend menggunakan CPU untuk waktu yang lebih lama dari biasanya. Cari bukti di metrik `os.cpuUtilization` Wawasan Performa atau metrik `CPUUtilization` CloudWatch. Jika sistem operasi kelebihan beban, lihat metrik Peningkatan Pemantauan untuk mendiagnosis lebih lanjut. Secara khusus, lihat daftar proses dan persentase CPU yang dikonsumsi oleh setiap proses.
  - Pernyataan SQL teratas mengonsumsi terlalu banyak CPU.

    Periksa pernyataan yang terkait dengan penggunaan CPU untuk mengetahui apakah pernyataan tersebut dapat menggunakan lebih sedikit CPU. Jalankan perintah `EXPLAIN`, lalu fokus pada simpul rencana yang memiliki dampak terbesar. Pertimbangkan untuk menggunakan visualizer rencana eksekusi PostgreSQL. Untuk mencoba alat bantu ini, lihat http://explain.dalibo.com/.

Merespons perubahan beban kerja

Jika beban kerja Anda berubah, cari jenis perubahan berikut:

Kueri baru

Periksa apakah kueri baru diharapkan. Jika ya, pastikan rencana eksekusinya dan jumlah eksekusi per detik diharapkan.

Peningkatan ukuran set data

Ketahui apakah partisi, jika belum diterapkan, dapat membantu. Strategi ini dapat mengurangi jumlah halaman yang perlu diambil oleh kueri.

Pemeliharaan atau pembuatan indeks

Periksa apakah jadwal pemeliharaan diharapkan. Praktik terbaik adalah menjadwalkan aktivitas pemeliharaan di luar aktivitas puncak.

Fungsi baru

Periksa apakah fungsi ini berjalan seperti yang diharapkan selama pengujian. Secara khusus, periksa apakah jumlah eksekusi per detik diharapkan.

Operator baru

Periksa apakah operator baru berfungsi seperti yang diharapkan selama pengujian.

Peningkatan dalam menjalankan kueri paralel

Ketahui apakah salah satu situasi berikut telah terjadi:

- Relasi atau indeks yang terlibat secara tiba-tiba berkembang ukurannya, sehingga sangat berbeda dengan `min_parallel_table_scan_size` atau `min_parallel_index_scan_size`.
- Perubahan terkini telah dilakukan pada `parallel_setup_cost` atau `parallel_tuple_cost`.
- Perubahan terkini telah dilakukan pada `max_parallel_workers` atau `max_parallel_workers_per_gather`.

IO:BufFileRead dan IO:BufFileWrite

Peristiwa `IO:BufFileRead` dan `IO:BufFileWrite` terjadi saat Aurora PostgreSQL membuat file sementara. Saat operasi membutuhkan lebih banyak memori daripada yang saat ini ditentukan oleh

parameter memori kerja, operasi akan menulis data sementara ke penyimpanan persisten. Operasi ini terkadang disebut "spilling to disk".

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

Konteks

`IO:BufFileRead` dan `IO:BufFileWrite` berkaitan dengan area memori kerja dan area memori kerja pemeliharaan. Untuk informasi selengkapnya tentang area memori lokal ini, lihat Area memori kerja dan Area memori kerja pemeliharaan.

Nilai default untuk `work_mem` adalah 4 MB. Jika satu sesi melakukan operasi secara paralel, setiap pekerja yang menangani paralelisme menggunakan memori 4 MB. Untuk alasan ini, atur `work_mem` dengan hati-hati. Jika Anda meningkatkan nilai terlalu banyak, basis data yang menjalankan banyak sesi mungkin mengonsumsi terlalu banyak memori. Jika Anda menetapkan nilai terlalu rendah, Aurora PostgreSQL akan membuat file sementara di penyimpanan lokal. Disk I/O untuk file sementara ini dapat mengurangi performa.

Jika Anda mengamati urutan peristiwa berikut, basis data mungkin menghasilkan file sementara:

1. Penurunan ketersediaan secara tiba-tiba dan tajam
2. Pemulihan cepat untuk ruang kosong

Anda mungkin juga melihat pola "gergaji". Pola ini dapat menunjukkan bahwa basis data membuat file kecil secara terus-menerus.

Kemungkinan penyebab peningkatan peristiwa tunggu

Secara umum, peristiwa tunggu ini disebabkan oleh operasi yang mengonsumsi lebih banyak memori daripada yang dialokasikan oleh parameter `work_mem` atau `maintenance_work_mem`. Untuk

mengompensasi, operasi menulis ke file sementara. Penyebab umum peristiwa `IO:BufFileRead` dan `IO:BufFileWrite` mencakup hal berikut:

Kueri yang membutuhkan lebih banyak memori daripada yang ada di area memori kerja

Kueri dengan karakteristik berikut menggunakan area memori kerja:

- Sambungan hash
- Klausa `ORDER BY`
- Klausa `GROUP BY`
- `DISTINCT`
- Fungsi jendela
- `CREATE TABLE AS SELECT`
- Penyegaran tampilan terwujud

Pernyataan yang membutuhkan lebih banyak memori daripada yang ada di area memori kerja pemeliharaan

Pernyataan berikut menggunakan area memori kerja pemeliharaan:

- `CREATE INDEX`
- `CLUSTER`

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengidentifikasi masalah
- Memeriksa kueri gabungan
- Memeriksa kueri ORDER BY dan GROUP BY
- Mencegah penggunaan operasi DISTINCT
- Mempertimbangkan untuk menggunakan fungsi jendela alih-alih fungsi GROUP BY
- Menyelidiki tampilan terwujud dan pernyataan CTAS
- Menggunakan pg_repack saat Anda membuat indeks

- Meningkatkan maintenance_work_mem saat Anda mengelompokkan tabel
- Menyetel memori untuk mencegah IO:BufFileRead dan IO:BufFileWrite

Mengidentifikasi masalah

Asumsikan situasi saat Wawasan Performa tidak diaktifkan dan Anda menduga bahwa `IO:BufFileRead` dan `IO:BufFileWrite` terjadi lebih sering daripada biasanya. Lakukan hal berikut:

1. Periksa metrik `FreeLocalStorage` di Amazon CloudWatch.
2. Cari pola gergaji, yang merupakan serangkaian paku bergerigi.

Pola gergaji menunjukkan konsumsi cepat dan pelepasan penyimpanan, yang sering dikaitkan dengan file sementara. Jika Anda melihat pola ini, aktifkan Wawasan Performa. Saat menggunakan Wawasan Performa, Anda dapat mengidentifikasi waktu terjadinya peristiwa tunggu dan kueri yang terkait dengannya. Solusi Anda bergantung pada kueri spesifik yang menyebabkan peristiwa.

Sebagai alternatif, atur parameter `log_temp_files`. Parameter ini mencatat semua kueri yang menghasilkan lebih dari ambang batas KB file sementara. Jika nilainya `0`, Aurora PostgreSQL mencatat semua file sementara. Jika nilainya `1024`, Aurora PostgreSQL mencatat semua kueri yang menghasilkan file sementara berukuran lebih besar dari 1 MB. Untuk informasi selengkapnya tentang `log_temp_files`, lihat Pencatatan dan Pelaporan Kesalahan dalam dokumentasi PostgreSQL.

Memeriksa kueri gabungan

Aplikasi Anda mungkin menggunakan gabungan. Misalnya, kueri berikut menggabungkan empat tabel.

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = order.customer_id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

Kemungkinan penyebab lonjakan penggunaan file sementara adalah masalah dalam kueri itu sendiri. Misalnya, klausa yang rusak mungkin tidak memfilter gabungan dengan benar. Pertimbangkan gabungan inti kedua dalam contoh berikut.

```
SELECT *
       FROM order
 INNER JOIN order_item
       ON (order.id = order_item.order_id)
 INNER JOIN customer
       ON (customer.id = customer.id)
 INNER JOIN customer_address
       ON (customer_address.customer_id = customer.id AND
           order.customer_address_id = customer_address.id)
 WHERE customer.id = 1234567890;
```

Kueri sebelumnya secara keliru menggabungkan `customer.id` ke `customer.id`, sehingga memberikan hasil perkalian Cartesian antara setiap pelanggan dan setiap pesanan. Jenis gabungan yang tak terduga ini menghasilkan file sementara yang besar. Tergantung pada ukuran tabel, kueri Cartesian bahkan dapat mengisi penyimpanan. Aplikasi Anda dapat memiliki gabungan Cartesian jika kondisi berikut terpenuhi:

- Anda melihat penurunan besar dan tajam dalam ketersediaan penyimpanan, yang diikuti oleh pemulihan cepat.
- Tidak ada indeks yang dibuat.
- Tidak ada pernyataan `CREATE TABLE FROM SELECT` yang dikeluarkan.
- Tidak ada tampilan terwujud yang disegarkan.

Untuk melihat apakah tabel sedang digabungkan menggunakan kunci yang tepat, periksa kueri dan petunjuk pemetaan relasional objek Anda. Perlu diperhatikan bahwa kueri tertentu dari aplikasi Anda tidak dipanggil sepanjang waktu, dan beberapa kueri dihasilkan secara dinamis.

Memeriksa kueri ORDER BY dan GROUP BY

Dalam beberapa kasus, klausa `ORDER BY` dapat menghasilkan file sementara yang berlebihan. Pertimbangkan panduan berikut:

- Hanya sertakan kolom dalam klausa `ORDER BY` saat perlu diurutkan. Pedoman ini sangat penting untuk kueri yang menampilkan ribuan baris dan menentukan banyak kolom dalam klausa `ORDER BY`.

- Pertimbangkan untuk membuat indeks guna mempercepat klausa ORDER BY saat klausa cocok dengan kolom yang memiliki urutan naik atau turun yang sama. Indeks sebagian lebih disukai karena lebih kecil. Indeks yang lebih kecil lebih cepat untuk dibaca dan dilewati.
- Jika Anda membuat indeks untuk kolom yang dapat menerima nilai kosong, pertimbangkan apakah Anda ingin nilai kosong disimpan di akhir atau di awal indeks.

  Jika memungkinkan, kurangi jumlah baris yang perlu diurutkan dengan memfilter set hasil. Jika Anda menggunakan pernyataan klausa atau subkueri WITH, perlu diperhatikan bahwa kueri dalam menghasilkan kumpulan hasil, lalu meneruskannya ke kueri luar. Semakin banyak baris yang dapat difilter kueri, semakin sedikit urutan yang perlu dilakukan kueri.
- Jika Anda tidak perlu mendapatkan set hasil lengkap, gunakan klausa LIMIT. Misalnya, jika Anda hanya menginginkan lima baris teratas, kueri yang menggunakan klausa LIMIT akan berhenti memberikan hasil. Dengan cara ini, kueri membutuhkan lebih sedikit memori dan file sementara.

Kueri yang menggunakan klausa GROUP BY juga dapat memerlukan file sementara. Kueri GROUP BY meringkas nilai dengan menggunakan fungsi seperti berikut:

- COUNT
- AVG
- MIN
- MAX
- SUM
- STDDEV

Untuk menyetel kueri GROUP BY, ikuti rekomendasi untuk kueri ORDER BY.

Mencegah penggunaan operasi DISTINCT

Jika memungkinkan, jangan gunakan operasi DISTINCT untuk menghapus baris duplikat. Semakin banyak baris duplikat yang tidak perlu, yang ditampilkan oleh kueri Anda, operasi DISTINCT menjadi semakin mahal. Jika memungkinkan, tambahkan filter dalam klausa WHERE meskipun Anda menggunakan filter yang sama untuk tabel yang berbeda. Memfilter kueri dan menggabungkan dengan benar akan meningkatkan performa Anda serta mengurangi penggunaan sumber daya. Hal tersebut juga mencegah laporan dan hasil yang salah.

Jika Anda perlu menggunakan DISTINCT untuk beberapa baris dari tabel yang sama, pertimbangkan untuk membuat indeks komposit. Mengelompokkan beberapa kolom dalam indeks dapat meningkatkan waktu untuk mengevaluasi baris yang berbeda. Selain itu, jika menggunakan Amazon Aurora PostgreSQL versi 10 atau yang lebih tinggi, Anda dapat menghubungkan statistik di antara beberapa kolom dengan menggunakan perintah CREATE STATISTICS.

Mempertimbangkan untuk menggunakan fungsi jendela alih-alih fungsi GROUP BY

Dengan menggunakan GROUP BY, Anda mengubah set hasil, lalu mengambil hasil agregat. Dengan menggunakan fungsi jendela, Anda mengumpulkan data tanpa mengubah set hasil. Fungsi jendela menggunakan klausa OVER untuk melakukan perhitungan di seluruh set yang ditentukan oleh kueri, dengan menghubungkan satu baris dengan yang lain. Anda dapat menggunakan semua fungsi GROUP BY dalam fungsi jendela, tetapi juga menggunakan fungsi seperti berikut:

- RANK
- ARRAY_AGG
- ROW_NUMBER
- LAG
- LEAD

Untuk meminimalkan jumlah file sementara yang dihasilkan oleh fungsi jendela, hapus duplikasi untuk set hasil yang sama saat Anda membutuhkan dua agregasi yang berbeda. Pertimbangkan kueri berikut.

```
SELECT sum(salary) OVER (PARTITION BY dept ORDER BY salary DESC) as sum_salary
     , avg(salary) OVER (PARTITION BY dept ORDER BY salary ASC) as avg_salary
  FROM empsalary;
```

Anda dapat menulis ulang kueri dengan klausa WINDOW sebagai berikut:

```
SELECT sum(salary) OVER w as sum_salary
         , avg(salary) OVER w as_avg_salary
    FROM empsalary
  WINDOW w AS (PARTITION BY dept ORDER BY salary DESC);
```

Secara default, perencana eksekusi Aurora PostgreSQL menggabungkan simpul serupa, sehingga tidak menggandakan operasi. Namun, dengan menggunakan deklarasi eksplisit untuk blok jendela,

Anda dapat mempertahankan kueri dengan lebih mudah. Anda juga dapat meningkatkan performa dengan mencegah duplikasi.

Menyelidiki tampilan terwujud dan pernyataan CTAS

Saat tampilan terwujud disegarkan, kueri akan dijalankan. Kueri ini dapat berisi operasi seperti `GROUP BY`, `ORDER BY`, atau `DISTINCT`. Selama penyegaran, Anda mungkin mengamati sejumlah besar file sementara dan peristiwa tunggu `IO:BufFileWrite` dan `IO:BufFileRead`. Demikian pula, saat Anda membuat tabel berdasarkan pernyataan `SELECT`, pernyataan `CREATE TABLE` tersebut menjalankan kueri. Untuk mengurangi file sementara yang dibutuhkan, optimalkan kueri.

Menggunakan pg_repack saat Anda membuat indeks

Saat Anda membuat indeks, mesin mengurutkan set hasil. Saat tabel bertambah besar, dan karena nilai di kolom yang diindeks menjadi lebih beragam, file sementara membutuhkan lebih banyak ruang. Dalam kebanyakan kasus, Anda tidak dapat mencegah pembuatan file sementara untuk tabel besar tanpa memodifikasi area memori kerja pemeliharaan. Untuk informasi selengkapnya, lihat Area memori kerja pemeliharaan.

Solusi yang memungkinkan saat membuat ulang indeks besar adalah menggunakan alat pg_repack. Untuk informasi selengkapnya, lihat Mengatur ulang tabel di basis data PostgreSQL dengan kunci minimal dalam dokumentasi pg_repack.

Meningkatkan maintenance_work_mem saat Anda mengelompokkan tabel

Perintah `CLUSTER` mengelompokkan tabel yang ditentukan oleh table_name berdasarkan indeks yang ada yang ditentukan oleh index_name. Aurora PostgreSQL secara fisik membuat ulang tabel agar sesuai dengan urutan indeks yang diberikan.

Saat penyimpanan magnetik lazim, klaster menjadi umum karena throughput penyimpanan terbatas. Sekarang penyimpanan berbasis SSD sudah umum, klaster menjadi kurang populer. Namun, jika mengelompokkan tabel, Anda masih dapat meningkatkan sedikit performa tergantung pada ukuran tabel, indeks, kueri, dan banyak lagi.

Jika Anda menjalankan perintah `CLUSTER` dan mengamati peristiwa tunggu `IO:BufFileWrite` dan `IO:BufFileRead`, sesuaikan `maintenance_work_mem`. Tingkatkan ukuran memori ke jumlah yang cukup besar. Nilai tinggi berarti mesin dapat menggunakan lebih banyak memori untuk operasi klaster.

Menyetel memori untuk mencegah IO:BufFileRead dan IO:BufFileWrite

Dalam situasi tertentu, Anda perlu menyetel memori. Tujuannya adalah menyeimbangkan persyaratan berikut:

- Nilai `work_mem` (lihat Area memori kerja)
- Memori yang tersisa setelah memangkas nilai `shared_buffers` (lihat Pool buffer)
- Koneksi maksimum yang dibuka dan digunakan, yang dibatasi oleh `max_connections`

Meningkatkan ukuran area memori kerja

Dalam beberapa situasi, satu-satunya pilihan adalah menambah memori yang digunakan oleh sesi Anda. Jika kueri Anda ditulis dengan benar dan menggunakan kunci yang benar untuk bergabung, pertimbangkan untuk meningkatkan nilai `work_mem`. Untuk informasi selengkapnya, lihat Area memori kerja.

Untuk mengetahui jumlah file sementara yang dihasilkan kueri, atur `log_temp_files` ke `0`. Jika meningkatkan nilai `work_mem` ke nilai maksimum yang diidentifikasi dalam log, Anda mencegah kueri menghasilkan file sementara. Namun, `work_mem` menetapkan nilai maksimum per simpul rencana untuk setiap koneksi atau pekerja paralel. Jika basis data memiliki 5.000 koneksi, dan jika masing-masing menggunakan memori 256 MiB, mesin akan membutuhkan RAM 1,2 TiB. Karena itu, instans Anda dapat kehabisan memori.

Mencadangkan memori yang cukup untuk pool buffer bersama

Basis data Anda menggunakan area memori seperti pool buffer bersama, bukan hanya area memori kerja. Pertimbangkan persyaratan area memori tambahan ini sebelum Anda meningkatkan `work_mem`. Untuk informasi selengkapnya tentang pool buffer, lihat Pool buffer.

Misalnya, asumsikan bahwa kelas instans Aurora PostgreSQL Anda adalah db.r5.2xlarge. Kelas ini memiliki memori 64 GiB. Secara default, 75% memori dicadangkan untuk pool buffer bersama. Setelah Anda mengurangi jumlah yang dialokasikan ke area memori bersama, 16.384 MB tetap ada. Jangan alokasikan memori yang tersisa hanya ke area memori kerja karena sistem operasi dan mesin juga memerlukan memori.

Memori yang dapat dialokasikan ke `work_mem` bergantung pada kelas instans. Jika Anda menggunakan kelas instans yang lebih besar, memori yang tersedia akan lebih banyak. Namun, dalam contoh sebelumnya, Anda tidak dapat menggunakan lebih dari 16 GiB. Jika melakukannya,

instans Anda menjadi tidak tersedia saat kehabisan memori. Untuk memulihkan instans dari status tidak tersedia, layanan otomatisasi Aurora PostgreSQL secara otomatis dimulai ulang.

Mengelola jumlah koneksi

Misalnya, instans basis data Anda memiliki 5.000 koneksi bersama. Setiap koneksi menggunakan setidaknya 4 MiB `work_mem`. Konsumsi memori yang tinggi dari koneksi cenderung menurunkan performa. Untuk itu, Anda memiliki opsi berikut:

- Tingkatkan ke kelas instans yang lebih besar.
- Kurangi jumlah koneksi basis data bersama dengan menggunakan proksi koneksi atau pooler.

Untuk proksi, pertimbangkan Proksi Amazon RDS, pgBouncer, atau pooler koneksi berdasarkan aplikasi Anda. Solusi ini mengurangi beban CPU. Solusi ini juga mengurangi risiko saat semua koneksi memerlukan area memori kerja. Saat koneksi basis data tersedia lebih sedikit, Anda dapat meningkatkan nilai `work_mem`. Dengan cara ini, Anda mengurangi munculnya peristiwa tunggu `IO:BufFileRead` dan `IO:BufFileWrite`. Selain itu, kueri yang menunggu area memori kerja dipercepat secara signifikan.

IO:DataFileRead

Peristiwa `IO:DataFileRead` terjadi saat koneksi menunggu pada proses backend untuk membaca halaman yang diperlukan dari penyimpanan karena halaman tidak tersedia dalam memori bersama.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

Konteks

Semua kueri dan operasi manipulasi data (DML) mengakses halaman di pool buffer. Pernyataan yang dapat menimbulkan pembacaan mencakup `SELECT`, `UPDATE`, dan `DELETE`. Misalnya, `UPDATE`

dapat membaca halaman dari tabel atau indeks. Jika halaman yang diminta atau diperbarui tidak berada dalam pool buffer bersama, pembacaan ini dapat mengarah ke peristiwa `IO:DataFileRead`.

Karena bersifat terbatas, pool buffer bersama dapat diisi. Dalam hal ini, permintaan untuk halaman yang tidak berada dalam memori memaksa basis data untuk membaca blok dari disk. Jika peristiwa `IO:DataFileRead` sering terjadi, pool buffer bersama mungkin terlalu kecil untuk mengakomodasi beban kerja Anda. Masalah ini bersifat akut untuk kueri `SELECT` yang membaca sejumlah besar baris yang tidak sesuai dengan pool buffer. Untuk informasi selengkapnya tentang pool buffer, lihat Pool buffer.

Kemungkinan penyebab peningkatan peristiwa tunggu

Penyebab umum peristiwa `IO:DataFileRead` tersebut mencakup:

Lonjakan koneksi

Anda mungkin menemukan beberapa koneksi yang menghasilkan jumlah peristiwa tunggu IO:DataFileRead yang sama. Dalam hal ini, lonjakan (peningkatan tiba-tiba dan besar) dalam peristiwa `IO:DataFileRead` dapat terjadi.

Pernyataan SELECT dan DML yang melakukan pemindaian berurutan

Aplikasi Anda mungkin melakukan operasi baru. Operasi yang ada mungkin juga berubah karena rencana eksekusi baru. Dalam kasus ini, cari tabel (terutama tabel besar) yang memiliki nilai `seq_scan` yang lebih besar. Temukan tabel dengan membuat kueri `pg_stat_user_tables`. Untuk melacak kueri yang menghasilkan lebih banyak operasi baca, gunakan ekstensi `pg_stat_statements`.

CTAS dan CREATE INDEX untuk set data besar

CTAS adalah sebuah pernyataan `CREATE TABLE AS SELECT`. Jika Anda menjalankan CTAS menggunakan set data besar sebagai sumber, atau membuat indeks pada tabel besar, maka peristiwa `IO:DataFileRead` dapat terjadi. Saat Anda membuat indeks, basis data mungkin perlu membaca seluruh objek menggunakan pemindaian berurutan. CTAS menghasilkan pembacaan `IO:DataFile` saat halaman tidak ada dalam memori.

Beberapa pekerja vakum berjalan pada waktu yang sama

Pekerja vakum dapat dipicu secara manual atau otomatis. Sebaiknya adopsi strategi vakum yang agresif. Namun, saat tabel memiliki banyak baris yang diperbarui atau dihapus, peristiwa tunggu `IO:DataFileRead` bertambah. Setelah ruang direklamasi, waktu vakum yang dihabiskan untuk `IO:DataFileRead` akan berkurang.

Menyerap data dalam jumlah besar

Saat aplikasi Anda menyerap data dalam jumlah besar, operasi `ANALYZE` mungkin terjadi lebih sering. Proses `ANALYZE` dapat dipicu oleh peluncur autovacuum atau diinvokasi secara manual.

Operasi `ANALYZE` membaca subset dari tabel. Jumlah halaman yang harus dipindai dihitung menggunakan perkalian 30 dengan nilai `default_statistics_target`. Untuk informasi selengkapnya, lihat Dokumentasi PostgreSQL. Parameter `default_statistics_target` menerima nilai antara 1 hingga 10.000, dengan nilai default adalah 100.

Kekurangan sumber daya

Jika bandwidth jaringan instans atau CPU dikonsumsi, peristiwa `IO:DataFileRead` mungkin terjadi lebih sering.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Memeriksa filter predikat untuk kueri yang menghasilkan peristiwa tunggu
- Meminimalkan efek operasi pemeliharaan
- Merespons jumlah koneksi yang tinggi

Memeriksa filter predikat untuk kueri yang menghasilkan peristiwa tunggu

Asumsikan bahwa Anda mengidentifikasi kueri spesifik yang menghasilkan peristiwa tunggu `IO:DataFileRead`. Anda dapat mengidentifikasinya menggunakan teknik berikut:

- Wawasan Performa
- Tampilan katalog seperti yang disediakan oleh ekstensi `pg_stat_statements`
- Tampilan katalog `pg_stat_all_tables`, jika secara berkala menunjukkan peningkatan jumlah pembacaan fisik
- Tampilan `pg_statio_all_tables`, jika menunjukkan bahwa penghitung `_read` meningkat

Sebaiknya Anda menentukan filter yang akan digunakan dalam predikat (klausa `WHERE`) kueri ini. Ikuti pedoman berikut:

- Jalankan perintah `EXPLAIN`. Pada output, identifikasi jenis pemindaian yang digunakan. Pemindaian berurutan tidak selalu menunjukkan adanya masalah. Kueri yang menggunakan pemindaian berurutan secara alami menghasilkan lebih banyak peristiwa `IO:DataFileRead` jika dibandingkan dengan kueri yang menggunakan filter.

  Cari tahu apakah kolom yang tercantum dalam klausa `WHERE` telah diindeks. Jika tidak, coba buat indeks untuk kolom ini. Pendekatan ini mencegah pemindaian berurutan dan mengurangi peristiwa `IO:DataFileRead`. Jika kueri memiliki filter yang ketat dan masih menghasilkan pemindaian berurutan, evaluasi apakah indeks yang tepat sedang digunakan.
- Cari tahu apakah kueri mengakses tabel yang sangat besar. Dalam beberapa kasus, partisi tabel dapat meningkatkan performa, dengan memungkinkan kueri hanya membaca partisi yang diperlukan.
- Periksa kardinalitas (jumlah total baris) dari operasi gabungan Anda. Perhatikan seberapa ketat nilai yang Anda teruskan di filter untuk klausa `WHERE` Anda. Jika memungkinkan, setel kueri Anda untuk mengurangi jumlah baris yang diteruskan di setiap langkah rencana.

Meminimalkan efek operasi pemeliharaan

Operasi pemeliharaan seperti `VACUUM` dan `ANALYZE` bersifat penting. Sebaiknya jangan dinonaktifkan karena peristiwa tunggu `IO:DataFileRead` berkaitan dengan operasi pemeliharaan ini. Pendekatan berikut dapat meminimalkan efek operasi ini:

- Jalankan operasi pemeliharaan secara manual selama di luar jam sibuk. Teknik ini mencegah basis data mencapai ambang batas untuk operasi otomatis.
- Untuk tabel yang sangat besar, pertimbangkan untuk mempartisi tabel. Teknik ini mengurangi overhead operasi pemeliharaan. Basis data hanya mengakses partisi yang membutuhkan pemeliharaan.
- Saat Anda menyerap data dalam jumlah besar, coba nonaktifkan fitur analisis otomatis.

Fitur autovacuum secara otomatis dipicu untuk tabel saat rumus berikut benar.

```
pg_stat_user_tables.n_dead_tup > (pg_class.reltuples x autovacuum_vacuum_scale_factor)
 + autovacuum_vacuum_threshold
```

Tampilan `pg_stat_user_tables` dan katalog `pg_class` berisi beberapa baris. Satu baris dapat sesuai dengan satu baris di tabel Anda. Rumus ini mengasumsikan bahwa `reltuples` adalah

untuk tabel tertentu. Parameter `autovacuum_vacuum_scale_factor` (0,20 secara default) dan `autovacuum_vacuum_threshold` (50 tupel secara default) biasanya diatur secara global untuk seluruh instans. Namun, Anda dapat mengatur nilai yang berbeda untuk tabel tertentu.

Topik

- Menemukan tabel yang mengonsumsi ruang secara tidak perlu
- Menemukan indeks yang mengonsumsi ruang yang tidak perlu
- Menemukan tabel yang memenuhi syarat untuk autovacuum

Menemukan tabel yang mengonsumsi ruang secara tidak perlu

Untuk menemukan tabel yang mengonsumsi ruang lebih dari yang diperlukan, jalankan kueri berikut. Saat dijalankan oleh peran pengguna basis data yang tidak memiliki peran `rds_superuser`, kueri ini hanya menampilkan informasi tentang tabel yang diizinkan dibaca oleh peran pengguna. Kueri ini didukung oleh PostgreSQL versi 12 dan versi yang lebih baru.

```
WITH report AS (
   SELECT   schemaname
           ,tblname
           ,n_dead_tup
           ,n_live_tup
           ,block_size*tblpages AS real_size
           ,(tblpages-est_tblpages)*block_size AS extra_size
           ,CASE WHEN tblpages - est_tblpages > 0
              THEN 100 * (tblpages - est_tblpages)/tblpages::float
              ELSE 0
            END AS extra_ratio, fillfactor, (tblpages-est_tblpages_ff)*block_size AS
 bloat_size
           ,CASE WHEN tblpages - est_tblpages_ff > 0
              THEN 100 * (tblpages - est_tblpages_ff)/tblpages::float
              ELSE 0
            END AS bloat_ratio
           ,is_na
    FROM (
          SELECT  ceil( reltuples / ( (block_size-page_hdr)/tpl_size ) ) +
 ceil( toasttuples / 4 ) AS est_tblpages
                 ,ceil( reltuples / ( (block_size-page_hdr)*fillfactor/
(tpl_size*100) ) ) + ceil( toasttuples / 4 ) AS est_tblpages_ff
                 ,tblpages
                 ,fillfactor
                 ,block_size
```

```
                  ,tblid
                  ,schemaname
                  ,tblname
                  ,n_dead_tup
                  ,n_live_tup
                  ,heappages
                  ,toastpages
                  ,is_na
              FROM (
                     SELECT ( 4 + tpl_hdr_size + tpl_data_size + (2*ma)
                                - CASE WHEN tpl_hdr_size%ma = 0 THEN ma ELSE
tpl_hdr_size%ma END
                                - CASE WHEN ceil(tpl_data_size)::int%ma = 0 THEN ma ELSE
ceil(tpl_data_size)::int%ma END
                            ) AS tpl_size
                            ,block_size - page_hdr AS size_per_block
                            ,(heappages + toastpages) AS tblpages
                            ,heappages
                            ,toastpages
                            ,reltuples
                            ,toasttuples
                            ,block_size
                            ,page_hdr
                            ,tblid
                            ,schemaname
                            ,tblname
                            ,fillfactor
                            ,is_na
                            ,n_dead_tup
                            ,n_live_tup
                           FROM (
                                SELECT  tbl.oid                       AS tblid
                                       ,ns.nspname                    AS schemaname
                                       ,tbl.relname                   AS tblname
                                       ,tbl.reltuples                 AS reltuples
                                       ,tbl.relpages                  AS heappages
                                       ,coalesce(toast.relpages, 0)   AS toastpages
                                       ,coalesce(toast.reltuples, 0)  AS toasttuples
                                       ,psat.n_dead_tup               AS n_dead_tup
                                       ,psat.n_live_tup               AS n_live_tup
                                       ,24                            AS page_hdr
                                       ,current_setting('block_size')::numeric AS
block_size
```

```
 ,coalesce(substring( array_to_string(tbl.reloptions, ' ') FROM
 'fillfactor=([0-9]+)')::smallint, 100) AS fillfactor
                                       ,CASE WHEN version()~'mingw32' OR version()~'64-
bit|x86_64|ppc64|ia64|amd64' THEN 8 ELSE 4 END        AS ma
                                       ,23 + CASE WHEN MAX(coalesce(null_frac,0)) > 0
 THEN ( 7 + count(*) ) / 8 ELSE 0::int END              AS tpl_hdr_size
                                       ,sum( (1-coalesce(s.null_frac, 0)) *
 coalesce(s.avg_width, 1024) )                                  AS tpl_data_size
                                       ,bool_or(att.atttypid =
 'pg_catalog.name'::regtype) OR count(att.attname) <> count(s.attname)         AS is_na
                                  FROM  pg_attribute       AS att
                                  JOIN  pg_class           AS tbl    ON (att.attrelid =
 tbl.oid)
                                  JOIN  pg_stat_all_tables AS psat   ON (tbl.oid =
 psat.relid)
                                  JOIN  pg_namespace       AS ns     ON (ns.oid =
 tbl.relnamespace)
                               LEFT JOIN  pg_stats         AS s      ON
 (s.schemaname=ns.nspname AND s.tablename = tbl.relname AND s.inherited=false AND
 s.attname=att.attname)
                               LEFT JOIN  pg_class         AS toast  ON
 (tbl.reltoastrelid = toast.oid)
                                   WHERE  att.attnum > 0
                                     AND  NOT att.attisdropped
                                     AND  tbl.relkind = 'r'
                                GROUP BY  tbl.oid, ns.nspname, tbl.relname,
 tbl.reltuples, tbl.relpages, toastpages, toasttuples, fillfactor, block_size, ma,
 n_dead_tup, n_live_tup
                                ORDER BY  schemaname, tblname
                              ) AS s
                  ) AS s2
       ) AS s3
 ORDER BY bloat_size DESC
)
  SELECT *
    FROM report
   WHERE bloat_ratio != 0
 -- AND schemaname = 'public'
 -- AND tblname = 'pgbench_accounts'
;

-- WHERE NOT is_na
```

```
--   AND tblpages*((pst).free_percent + (pst).dead_tuple_percent)::float4/100 >= 1
```

Anda dapat memeriksa bloat indeks dan tabel di aplikasi Anda. Untuk informasi selengkapnya, lihat

Anda dapat menggunakan Multiversion Concurrency Control (MVCC) PostgreSQL untuk membantu menjaga integritas data. MVCC PostgreSQL beroperasi dengan menyimpan salinan internal baris yang diperbarui atau dihapus (juga disebut tuple) sampai transaksi dikomit atau di-rollback. Salinan internal yang disimpan ini tidak terlihat oleh pengguna. Namun, bloat tabel dapat terjadi ketika salinan yang tidak terlihat tersebut tidak dibersihkan secara teratur oleh utilitas VACUUM atau AUTOVACUUM. Bloat tabel yang tidak terpantau dapat menimbulkan peningkatan biaya penyimpanan dan memperlambat kecepatan pemrosesan Anda.

Dalam banyak kasus, pengaturan default untuk VACUUM atau AUTOVACUUM di Aurora cukup untuk menangani bloat tabel yang tidak diinginkan. Namun, Anda sebaiknya memeriksa bloat jika aplikasi Anda mengalami kondisi berikut:

- Memproses sejumlah besar transaksi dalam waktu yang relatif singkat di antara proses VACUUM.

- Beperforma buruk dan kehabisan penyimpanan.

Untuk memulai, kumpulkan informasi paling akurat tentang jumlah ruang yang digunakan oleh tuple nonaktif dan jumlah yang dapat Anda pulihkan dengan membersihkan bloat tabel dan indeks. Untuk melakukannya, gunakan ekstensi `pgstattuple` untuk mengumpulkan statistik tentang klaster Aurora Anda. Untuk informasi selengkapnya, lihat pgstattuple. Hak akses untuk menggunakan ekstensi `pgstattuple` dibatasi pada peran `pg_stat_scan_tables` dan superuser basis data.

Untuk membuat ekstensi `pgstattuple` di Aurora, hubungkan sesi klien ke klaster, misalnya, psql atau pgAdmin, lalu gunakan perintah berikut:

```
CREATE EXTENSION pgstattuple;
```

Buat ekstensi di setiap basis data yang ingin Anda buat profilnya. Setelah membuat ekstensi, gunakan antarmuka baris perintah (CLI) untuk mengukur jumlah ruang yang tidak dapat digunakan yang dapat Anda klaim kembali. Sebelum mengumpulkan statistik, ubah grup parameter klaster dengan menetapkan AUTOVACUUM ke 0. Pengaturan 0 akan mencegah Aurora membersihkan tuple nonaktif secara otomatis yang ditinggalkan oleh aplikasi Anda, yang dapat memengaruhi keakuratan hasil. Masukkan perintah berikut untuk membuat tabel sederhana:

```
postgres=> CREATE TABLE lab AS SELECT generate_series (0,100000);
```

```
SELECT 100001
```

Dalam contoh berikut, kita menjalankan kueri dengan AUTOVACUUM diaktifkan untuk klaster DB. `dead_tuple_count` adalah 0, yang menunjukkan bahwa AUTOVACUUM telah menghapus data atau tuple usang dari basis data PostgreSQL.

Untuk menggunakan `pgstattuple` dalam mengumpulkan informasi tentang tabel, tentukan nama tabel atau pengidentifikasi objek (OID) dalam kueri:

```
postgres=> SELECT * FROM pgstattuple('lab');
```

```

```

```
table_len  | tuple_count | tuple_len | tuple_percent | dead_tuple_count |
 dead_tuple_len | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056    | 100001      | 2800028   | 77.16         | 0                | 0
  | 0                 | 16616      | 0.46
(1 row)
```

```

```

Dalam kueri berikut, kita menonaktifkan AUTOVACUUM dan memasukkan perintah yang menghapus 25.000 baris dari tabel. Akibatnya, `dead_tuple_count` meningkat menjadi 25000.

```

```

```
postgres=> DELETE FROM lab WHERE generate_series < 25000;
```

```

```



```
DELETE 25000
```

```

```

```
----------+-------------+----------+--------------+-----------------
```

```
+---------------+--------------------+------------+-------------
```

```
3629056 | 75001 | 2100028 | 57.87 | 25000 | 700000 | 19.29 | 16616 | 0.46
```

```
(1 row)
```

Untuk mengklaim kembali tuple nonaktif, mulailah proses VACUUM.

Mengamati bloat tanpa menghentikan aplikasi Anda

Pengaturan pada klaster Aurora dioptimalkan untuk memberikan praktik terbaik untuk sebagian besar beban kerja. Namun, Anda sebaiknya mengoptimalkan klaster agar lebih sesuai dengan aplikasi dan pola penggunaan Anda. Dalam hal ini, Anda dapat menggunakan ekstensi `pgstattuple` tanpa menghentikan aplikasi yang sibuk. Untuk melakukannya, lakukan langkah-langkah berikut ini:

1. Kloning instans Aurora Anda.
2. Ubah file parameter untuk menonaktifkan AUTOVACUUM di klon.
3. Lakukan kueri `pgstattuple` saat menguji klon dengan beban kerja sampel atau dengan pgbench, yang merupakan program untuk menjalankan pengujian tolok ukur di PostgreSQL. Untuk informasi selengkapnya, lihat pgbench.

Setelah menjalankan aplikasi Anda dan melihat hasilnya, gunakan pg_repack atau VACUUM FULL pada salinan yang dipulihkan dan bandingkan perbedaannya. Jika Anda melihat penurunan yang signifikan pada dead_tuple_count, dead_tuple_len, atau dead_tuple_percent, sesuaikan jadwal vacuum pada klaster produksi Anda untuk meminimalkan bloat.

Menghindari bloat di tabel sementara

Jika aplikasi Anda membuat tabel sementara, pastikan bahwa aplikasi Anda menghapus tabel sementara ketika sudah tidak lagi diperlukan. Proses autovacuum tidak menemukan tabel sementara. Jika tidak terpantau, tabel sementara dapat dengan cepat membuat bloat basis data. Selain itu, bloat dapat meluas ke tabel sistem, yang merupakan tabel internal yang melacak objek dan atribut PostgreSQL, seperti pg_attribute dan pg_depend.

Ketika tabel sementara tidak lagi diperlukan, Anda dapat menggunakan pernyataan TRUNCATE untuk mengosongkan tabel dan membebaskan ruang. Kemudian, lakukan vacuum manual pada tabel pg_attribute dan pg_depend. Dengan melakukan vacuum pada tabel ini, akan dipastikan

bahwa pembuatan dan pemotongan/penghapusan tabel sementara secara terus-menerus tidak akan menambahkan tuple dan berkontribusi pada bloat sistem.

Anda dapat menghindari masalah ini saat membuat tabel sementara dengan menyertakan sintaksis berikut yang menghapus baris baru saat konten dikomit:

```
CREATE TEMP TABLE IF NOT EXISTS table_name(table_description) ON COMMIT DELETE ROWS;
```

Klausa `ON COMMIT DELETE ROWS` memotong tabel sementara ketika transaksi dikomit.

Menghindari bloat dalam indeks

Saat Anda mengubah bidang yang diindeks dalam tabel, pembaruan indeks akan menghasilkan satu atau beberapa tuple nonaktif dalam indeks tersebut. Secara default, proses autovacuum membersihkan bloat dalam indeks, tetapi pembersihan tersebut menggunakan sejumlah besar waktu dan sumber daya. Untuk menentukan preferensi pembersihan indeks saat Anda membuat tabel, sertakan klausa vacuum_index_cleanup. Secara default, pada waktu pembuatan tabel, klausa ini diatur ke AUTO, yang berarti bahwa server akan memutuskan apakah indeks Anda memerlukan pembersihan saat melakukan vaccum pada tabel. Anda dapat mengatur klausa ini ke ON untuk mengaktifkan pembersihan indeks untuk tabel tertentu, atau OFF untuk menonaktifkan pembersihan indeks untuk tabel tersebut. Ingat, menonaktifkan pembersihan indeks mungkin menghemat waktu, tetapi berpotensi menyebabkan indeks mengalami bloat.

Anda dapat mengontrol pembersihan indeks secara manual saat Anda melakukan VACUUM pada tabel di baris perintah. Untuk melakukan vacuum pada tabel dan menghapus tuple nonaktif dari indeks, sertakan klausa INDEX_CLEANUP dengan nilai ON dan nama tabel:

```
acctg=> VACUUM (INDEX_CLEANUP ON) receivables;
```

```

```

```
INFO: aggressively vacuuming "public.receivables"
```

```
VACUUM
```

Untuk melakukan vacuum pada tabel tanpa membersihkan indeks, tentukan nilai OFF:

```
acctg=> VACUUM (INDEX_CLEANUP OFF) receivables;
```

```

```



```
INFO: aggressively vacuuming "public.receivables"
```

```
VACUUM
```

Menemukan indeks yang mengonsumsi ruang yang tidak perlu

Untuk menemukan indeks yang mengonsumsi ruang yang tidak perlu, jalankan kueri berikut.

```
-- WARNING: run with a nonsuperuser role, the query inspects
-- only indexes on tables you have permissions to read.
-- WARNING: rows with is_na = 't' are known to have bad statistics ("name" type is not
 supported).
-- This query is compatible with PostgreSQL 8.2 and later.

SELECT current_database(), nspname AS schemaname, tblname, idxname,
 bs*(relpages)::bigint AS real_size,
  bs*(relpages-est_pages)::bigint AS extra_size,
  100 * (relpages-est_pages)::float / relpages AS extra_ratio,
  fillfactor, bs*(relpages-est_pages_ff) AS bloat_size,
  100 * (relpages-est_pages_ff)::float / relpages AS bloat_ratio,
  is_na
  -- , 100-(sub.pst).avg_leaf_density, est_pages, index_tuple_hdr_bm,
  -- maxalign, pagehdr, nulldatawidth, nulldatahdrwidth, sub.reltuples, sub.relpages
  -- (DEBUG INFO)
FROM (
  SELECT coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)/(4+nulldatahdrwidth)::float)), 0
       -- ItemIdData size + computed avg size of a tuple (nulldatahdrwidth)
    ) AS est_pages,
    coalesce(1 +
       ceil(reltuples/floor((bs-pageopqdata-pagehdr)*fillfactor/
(100*(4+nulldatahdrwidth)::float))), 0
    ) AS est_pages_ff,
    bs, nspname, table_oid, tblname, idxname, relpages, fillfactor, is_na
    -- , stattuple.pgstatindex(quote_ident(nspname)||'.'||quote_ident(idxname)) AS
 pst,
    -- index_tuple_hdr_bm, maxalign, pagehdr, nulldatawidth, nulldatahdrwidth,
 reltuples
    -- (DEBUG INFO)
  FROM (
    SELECT maxalign, bs, nspname, tblname, idxname, reltuples, relpages, relam,
 table_oid, fillfactor,
      ( index_tuple_hdr_bm +
          maxalign - CASE -- Add padding to the index tuple header to align on MAXALIGN
            WHEN index_tuple_hdr_bm%maxalign = 0 THEN maxalign
            ELSE index_tuple_hdr_bm%maxalign
          END
```

```
       + nulldatawidth + maxalign - CASE -- Add padding to the data to align on
MAXALIGN
           WHEN nulldatawidth = 0 THEN 0
           WHEN nulldatawidth::integer%maxalign = 0 THEN maxalign
           ELSE nulldatawidth::integer%maxalign
         END
     )::numeric AS nulldatahdrwidth, pagehdr, pageopqdata, is_na
     -- , index_tuple_hdr_bm, nulldatawidth -- (DEBUG INFO)
   FROM (
     SELECT
       i.nspname, i.tblname, i.idxname, i.reltuples, i.relpages, i.relam, a.attrelid
AS table_oid,
       current_setting('block_size')::numeric AS bs, fillfactor,
       CASE -- MAXALIGN: 4 on 32bits, 8 on 64bits (and mingw32 ?)
         WHEN version() ~ 'mingw32' OR version() ~ '64-bit|x86_64|ppc64|ia64|amd64'
THEN 8
         ELSE 4
       END AS maxalign,
       /* per page header, fixed size: 20 for 7.X, 24 for others */
       24 AS pagehdr,
       /* per page btree opaque data */
       16 AS pageopqdata,
       /* per tuple header: add IndexAttributeBitMapData if some cols are null-able */
       CASE WHEN max(coalesce(s.null_frac,0)) = 0
         THEN 2 -- IndexTupleData size
         ELSE 2 + (( 32 + 8 - 1 ) / 8)
         -- IndexTupleData size + IndexAttributeBitMapData size ( max num filed per
index + 8 - 1 /8)
       END AS index_tuple_hdr_bm,
       /* data len: we remove null values save space using it fractionnal part from
stats */
       sum( (1-coalesce(s.null_frac, 0)) * coalesce(s.avg_width, 1024)) AS
nulldatawidth,
       max( CASE WHEN a.atttypid = 'pg_catalog.name'::regtype THEN 1 ELSE 0 END ) > 0
AS is_na
     FROM pg_attribute AS a
       JOIN (
         SELECT nspname, tbl.relname AS tblname, idx.relname AS idxname,
           idx.reltuples, idx.relpages, idx.relam,
           indrelid, indexrelid, indkey::smallint[] AS attnum,
           coalesce(substring(
             array_to_string(idx.reloptions, ' ')
              from 'fillfactor=([0-9]+)')::smallint, 90) AS fillfactor
         FROM pg_index
```

```
            JOIN pg_class idx ON idx.oid=pg_index.indexrelid
            JOIN pg_class tbl ON tbl.oid=pg_index.indrelid
            JOIN pg_namespace ON pg_namespace.oid = idx.relnamespace
          WHERE pg_index.indisvalid AND tbl.relkind = 'r' AND idx.relpages > 0
        ) AS i ON a.attrelid = i.indexrelid
        JOIN pg_stats AS s ON s.schemaname = i.nspname
          AND ((s.tablename = i.tblname AND s.attname =
 pg_catalog.pg_get_indexdef(a.attrelid, a.attnum, TRUE))
          -- stats from tbl
          OR  (s.tablename = i.idxname AND s.attname = a.attname))
          -- stats from functionnal cols
        JOIN pg_type AS t ON a.atttypid = t.oid
      WHERE a.attnum > 0
      GROUP BY 1, 2, 3, 4, 5, 6, 7, 8, 9
    ) AS s1
  ) AS s2
    JOIN pg_am am ON s2.relam = am.oid WHERE am.amname = 'btree'
) AS sub
-- WHERE NOT is_na
ORDER BY 2,3,4;
```

Menemukan tabel yang memenuhi syarat untuk autovacuum

Untuk menemukan tabel yang memenuhi syarat untuk autovacuum, jalankan kueri berikut.

```
--This query shows tables that need vacuuming and are eligible candidates.
--The following query lists all tables that are due to be processed by autovacuum.
-- During normal operation, this query should return very little.
WITH  vbt AS (SELECT setting AS autovacuum_vacuum_threshold
              FROM pg_settings WHERE name = 'autovacuum_vacuum_threshold')
    , vsf AS (SELECT setting AS autovacuum_vacuum_scale_factor
              FROM pg_settings WHERE name = 'autovacuum_vacuum_scale_factor')
    , fma AS (SELECT setting AS autovacuum_freeze_max_age
              FROM pg_settings WHERE name = 'autovacuum_freeze_max_age')
    , sto AS (SELECT opt_oid, split_part(setting, '=', 1) as param,
                split_part(setting, '=', 2) as value
              FROM (SELECT oid opt_oid, unnest(reloptions) setting FROM pg_class) opt)
SELECT
    '"'||ns.nspname||'"."'||c.relname||'"' as relation
    , pg_size_pretty(pg_table_size(c.oid)) as table_size
    , age(relfrozenxid) as xid_age
    , coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
 autovacuum_freeze_max_age
    , (coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
```

```
        coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) *
 c.reltuples)
        as autovacuum_vacuum_tuples
    , n_dead_tup as dead_tuples
FROM pg_class c
JOIN pg_namespace ns ON ns.oid = c.relnamespace
JOIN pg_stat_all_tables stat ON stat.relid = c.oid
JOIN vbt on (1=1)
JOIN vsf ON (1=1)
JOIN fma on (1=1)
LEFT JOIN sto cvbt ON cvbt.param = 'autovacuum_vacuum_threshold' AND c.oid =
 cvbt.opt_oid
LEFT JOIN sto cvsf ON cvsf.param = 'autovacuum_vacuum_scale_factor' AND c.oid =
 cvsf.opt_oid
LEFT JOIN sto cfma ON cfma.param = 'autovacuum_freeze_max_age' AND c.oid = cfma.opt_oid
WHERE c.relkind = 'r'
AND nspname <> 'pg_catalog'
AND (
    age(relfrozenxid) >= coalesce(cfma.value::float, autovacuum_freeze_max_age::float)
    or
    coalesce(cvbt.value::float, autovacuum_vacuum_threshold::float) +
      coalesce(cvsf.value::float,autovacuum_vacuum_scale_factor::float) * c.reltuples
 <= n_dead_tup
    -- or 1 = 1
)
ORDER BY age(relfrozenxid) DESC;
```

Merespons jumlah koneksi yang tinggi

Saat memantau Amazon CloudWatch, Anda mungkin menemukan bahwa metrik `DatabaseConnections` meningkat. Peningkatan ini menunjukkan bertambahnya jumlah koneksi ke basis data Anda. Sebaiknya lakukan pendekatan berikut:

- Membatasi jumlah koneksi yang dapat dibuka aplikasi dengan setiap instans. Jika aplikasi Anda memiliki fitur kumpulan koneksi tertanam, tetapkan jumlah koneksi yang wajar. Tentukan angka berdasarkan yang dapat diparalelkan oleh vCPU dalam instans Anda secara efektif.

  Jika aplikasi Anda tidak menggunakan fitur kumpulan koneksi, coba gunakan Proksi Amazon RDS atau alternatifnya. Pendekatan ini memungkinkan aplikasi Anda membuka beberapa koneksi dengan penyeimbang beban. Penyeimbang selanjutnya dapat membuka sejumlah koneksi terbatas dengan basis data. Karena lebih sedikit koneksi yang berjalan secara paralel, instans DB Anda melakukan lebih sedikit peralihan konteks di kernel. Kueri harus berkembang lebih cepat, yang

mengarah ke lebih sedikit peristiwa tunggu. Untuk informasi selengkapnya, lihat Menggunakan Proksi Amazon RDS untuk Aurora.

- Jika memungkinkan, manfaatkan simpul pembaca untuk Aurora PostgreSQL dan replika baca untuk RDS PostgreSQL. Saat aplikasi Anda menjalankan operasi hanya-baca, kirim permintaan ini ke titik akhir khusus pembaca. Teknik ini menyebarkan permintaan aplikasi di semua simpul pembaca, sehingga mengurangi tekanan I/O pada simpul penulis.
- Coba naikkan skala instans DB Anda. Kelas instans berkapasitas lebih tinggi memberikan memori lebih banyak, yang memberi Aurora PostgreSQL pool buffer bersama yang lebih besar untuk menampung halaman. Ukuran lebih besar juga memberikan instans DB lebih banyak vCPU untuk menangani koneksi. Lebih banyak vCPU akan sangat membantu saat operasi yang menghasilkan peristiwa tunggu `IO:DataFileRead` adalah operasi tulis.

IO:XactSync

Peristiwa `IO:XactSync` terjadi saat basis data menunggu subsistem penyimpanan Aurora mengakui commit transaksi reguler, atau commit ataupun rollback transaksi yang disiapkan. Transaksi yang disiapkan adalah bagian dari dukungan PostgreSQL untuk commit dua fase.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

Konteks

Peristiwa `IO:XactSync` menunjukkan bahwa instans menghabiskan waktu menunggu subsistem penyimpanan Aurora untuk mengkonfirmasi bahwa data transaksi telah diproses.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `IO:XactSync` ditampilkan lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

Saturasi jaringan

Lalu lintas antara klien dan instans DB atau lalu lintas ke subsistem penyimpanan mungkin terlalu berat untuk bandwidth jaringan.

Tekanan CPU

Beban kerja yang berat mungkin mencegah daemon penyimpanan Aurora mendapatkan waktu CPU yang cukup.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Memantau sumber daya
- Menaikkan skala CPU
- Meningkatkan bandwidth jaringan
- Mengurangi jumlah commit

Memantau sumber daya

Untuk menentukan penyebab peristiwa `IO:XactSync` yang meningkat, periksa metrik berikut:

- `WriteThroughput` dan `CommitThroughput` – Perubahan throughput tulis atau throughput commit dapat menunjukkan peningkatan beban kerja.
- `WriteLatency` dan `CommitLatency` – Perubahan latensi tulis atau latensi commit dapat menunjukkan bahwa subsistem penyimpanan diminta untuk melakukan lebih banyak pekerjaan.
- `CPUUtilization` – Jika penggunaan CPU instans di atas 90%, daemon penyimpanan Aurora mungkin tidak mendapatkan cukup waktu pada CPU. Dalam hal ini, performa I/O menurun.

Untuk informasi selengkapnya tentang metrik ini, lihat Metrik tingkat instans untuk Amazon Aurora.

Menaikkan skala CPU

Untuk mengatasi masalah kekurangan CPU, coba ubah ke jenis instans dengan kapasitas CPU yang lebih besar. Untuk informasi tentang kapasitas CPU untuk kelas instans DB, lihat Spesifikasi perangkat keras kelas instans DB untuk Aurora.

Meningkatkan bandwidth jaringan

Untuk menentukan apakah instans mencapai batas bandwidth jaringannya, periksa peristiwa tunggu lainnya berikut:

- `IO:DataFileRead`, `IO:BufferRead`, `IO:BufferWrite`, dan `IO:XactWrite` – Kueri yang menggunakan I/O dalam jumlah besar dapat membuat lebih banyak peristiwa tunggu ini.
- `Client:ClientRead` dan `Client:ClientWrite` – Kueri dengan komunikasi klien dalam jumlah besar dapat membuat lebih banyak peristiwa tunggu ini.

Jika masalahnya ada pada bandwidth jaringan, coba ubah ke jenis instans dengan bandwidth jaringan yang lebih banyak. Untuk informasi tentang performa jaringan untuk kelas instans DB, lihat Spesifikasi perangkat keras kelas instans DB untuk Aurora.

Mengurangi jumlah commit

Untuk mengurangi jumlah commit, gabungkan pernyataan ke dalam blok transaksi.

IPC:DamRecordTxAck

Peristiwa `IPC:DamRecordTxAck` terjadi saat Aurora PostgreSQL dalam sesi yang menggunakan aliran aktivitas basis data menghasilkan peristiwa aliran aktivitas, lalu menunggu peristiwa tersebut menjadi tahan lama.

Topik

- Versi mesin yang relevan
- Konteks
- Penyebab
- Tindakan

Versi mesin yang relevan

Informasi peristiwa tunggu ini relevan untuk semua Aurora PostgreSQL 10.7 dan versi 10 yang lebih tinggi, 11.4 dan versi 11 yang lebih tinggi, serta semua versi 12 dan 13.

Konteks

Dalam mode sinkron, daya tahan peristiwa aliran aktivitas lebih disukai daripada performa basis data. Saat menunggu penulisan peristiwa yang tahan lama, sesi akan memblokir aktivitas basis data lainnya, sehingga menyebabkan peristiwa tunggu `IPC:DamRecordTxAck`.

Penyebab

Penyebab paling umum munculnya peristiwa `IPC:DamRecordTxAck` dalam peristiwa tunggu teratas adalah bahwa fitur DAS (Aliran Aktivitas Basis Data) merupakan audit holistik. Aktivitas SQL yang lebih tinggi menghasilkan peristiwa aliran aktivitas yang perlu dicatat.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda:

- Kurangi jumlah pernyataan SQL atau nonaktifkan aliran aktivitas basis data. Tindakan ini akan mengurangi jumlah peristiwa yang membutuhkan penulisan yang tahan lama.
- Ubah ke mode asinkron. Tindakan ini akan membantu mengurangi pertentangan pada peristiwa tunggu `IPC:DamRecordTxAck`.

  Namun, fitur DAS tidak dapat menjamin daya tahan setiap peristiwa dalam mode asinkron.

Lock:advisory

Peristiwa `Lock:advisory` terjadi saat aplikasi PostgreSQL menggunakan kunci untuk mengoordinasi aktivitas di beberapa sesi.

Topik

- Versi mesin yang relevan
- Konteks
- Penyebab
- Tindakan

Versi mesin yang relevan

Informasi peristiwa tunggu ini relevan untuk Aurora PostgreSQL versi 9.6 dan yang lebih tinggi.

Konteks

Kunci penasihat PostgreSQL adalah kunci kooperatif tingkat aplikasi yang secara eksplisit dikunci dan dibuka oleh kode aplikasi pengguna. Aplikasi dapat menggunakan kunci penasihat PostgreSQL untuk mengoordinasi aktivitas di beberapa sesi. Tidak seperti kunci biasa tingkat objek atau baris, aplikasi memiliki kontrol penuh atas masa pakai kunci. Untuk informasi selengkapnya, lihat Kunci Penasihat dalam dokumentasi PostgreSQL.

Kunci penasihat dapat dilepaskan sebelum transaksi berakhir atau disimpan oleh sesi di seluruh transaksi. Ini tidak berlaku untuk kunci implisit yang diberlakukan sistem, seperti kunci eksklusif akses pada tabel yang diperoleh dari pernyataan `CREATE INDEX`.

Untuk deskripsi fungsi yang digunakan untuk memperoleh (mengunci) dan melepaskan (membuka kunci) kunci penasihat, lihat Fungsi Kunci Penasihat dalam dokumentasi PostgreSQL.

Kunci penasihat diterapkan di atas sistem penguncian PostgreSQL biasa dan terlihat dalam tampilan sistem `pg_locks`.

Penyebab

Jenis kunci ini secara eksklusif dikendalikan oleh aplikasi yang menggunakannya secara eksplisit. Kunci penasihat yang diperoleh untuk setiap baris sebagai bagian dari kueri dapat menyebabkan lonjakan kunci atau penumpukan jangka panjang.

Efek ini terjadi saat kueri dijalankan dengan cara yang memperoleh kunci pada lebih banyak baris daripada yang ditampilkan oleh kueri. Aplikasi pada akhirnya harus melepaskan setiap kunci, tetapi jika kunci diperoleh pada baris yang tidak ditampilkan, maka aplikasi tidak dapat menemukan semua kunci.

Contoh berikut berasal dari Kunci Penasihat dalam dokumentasi PostgreSQL.

```
SELECT pg_advisory_lock(id) FROM foo WHERE id > 12345 LIMIT 100;
```

Dalam contoh ini, klausa `LIMIT` hanya dapat menghentikan output kueri setelah baris dipilih secara internal dan nilai ID-nya dikunci. Hal ini dapat terjadi secara tiba-tiba saat volume data yang berkembang menyebabkan perencana memilih rencana eksekusi lain yang tidak diuji selama pengembangan. Penumpukan dalam kasus ini terjadi karena aplikasi secara eksplisit memanggil `pg_advisory_unlock` untuk setiap nilai ID yang terkunci. Namun, dalam kasus ini, aplikasi tidak dapat menemukan set kunci yang diperoleh pada baris yang tidak ditampilkan. Karena diperoleh pada tingkat sesi, kunci tidak dilepaskan secara otomatis pada akhir transaksi.

Kemungkinan penyebab lain untuk lonjakan upaya kunci yang diblokir adalah konflik yang tidak diinginkan. Dalam konflik ini, bagian aplikasi yang tidak terkait berbagi ruang ID kunci yang sama secara tidak sengaja.

Tindakan

Tinjau penggunaan kunci penasihat oleh aplikasi dan jelaskan secara detail di mana dan kapan dalam alur aplikasi setiap jenis kunci penasihat diperoleh dan dilepaskan.

Ketahui apakah sesi memperoleh terlalu banyak kunci atau sesi yang berjalan lama tidak melepaskan kunci cukup awal, sehingga mengarah ke penumpukan kunci yang lambat. Anda dapat memperbaiki penumpukan kunci tingkat sesi yang lambat dengan mengakhiri sesi menggunakan `pg_terminate_backend(pid)`.

Klien yang menunggu kunci penasihat muncul dalam `pg_stat_activity` dengan `wait_event_type=Lock` dan `wait_event=advisory`. Anda dapat memperoleh nilai kunci tertentu dengan membuat kueri tampilan sistem `pg_locks` untuk `pid` yang sama, dengan mencari `locktype=advisory` dan `granted=f`.

Anda kemudian dapat mengidentifikasi sesi pemblokiran dengan membuat kueri `pg_locks` untuk kunci penasihat serupa yang memiliki `granted=t`, seperti ditunjukkan dalam contoh berikut.

```
SELECT blocked_locks.pid AS blocked_pid,
         blocking_locks.pid AS blocking_pid,
         blocked_activity.usename AS blocked_user,
         blocking_activity.usename AS blocking_user,
         now() - blocked_activity.xact_start AS blocked_transaction_duration,
         now() - blocking_activity.xact_start AS blocking_transaction_duration,
         concat(blocked_activity.wait_event_type,':',blocked_activity.wait_event) AS
 blocked_wait_event,
         concat(blocking_activity.wait_event_type,':',blocking_activity.wait_event) AS
 blocking_wait_event,
         blocked_activity.state AS blocked_state,
         blocking_activity.state AS blocking_state,
         blocked_locks.locktype AS blocked_locktype,
         blocking_locks.locktype AS blocking_locktype,
         blocked_activity.query AS blocked_statement,
         blocking_activity.query AS blocking_statement
    FROM pg_catalog.pg_locks blocked_locks
    JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid =
 blocked_locks.pid
```

```
    JOIN pg_catalog.pg_locks blocking_locks
        ON blocking_locks.locktype = blocked_locks.locktype
        AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
        AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
        AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
        AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
        AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
        AND blocking_locks.transactionid IS NOT DISTINCT FROM
blocked_locks.transactionid
        AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
        AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
        AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
        AND blocking_locks.pid != blocked_locks.pid
    JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid =
blocking_locks.pid
    WHERE NOT blocked_locks.GRANTED;
```

Semua fungsi API kunci penasihat memiliki dua set argumen, baik satu argumen `bigint` atau dua argumen `integer`:

- Untuk fungsi API dengan satu argumen `bigint`, 32 bit atas masuk dalam `pg_locks.classid` dan 32 bit bawah masuk dalam `pg_locks.objid`.
- Untuk fungsi API dengan dua argumen `integer`, argumen pertama adalah `pg_locks.classid` dan argumen kedua adalah `pg_locks.objid`.

Nilai `pg_locks.objsubid` menunjukkan formulir API yang digunakan: 1 berarti satu argumen `bigint`; 2 berarti dua argumen `integer`.

Lock:extend

Peristiwa `Lock:extend` terjadi saat suatu proses backend sedang menunggu untuk mengunci relasi agar dapat diperluas, sementara proses lain mengunci relasi tersebut untuk tujuan yang sama.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

Konteks

Peristiwa `Lock:extend` menunjukkan bahwa suatu proses backend sedang menunggu untuk memperluas relasi yang dikunci oleh proses backend lain saat relasi tersebut diperluas. Karena hanya satu proses pada satu waktu yang dapat memperluas relasi, sistem membuat peristiwa tunggu `Lock:extend`. Operasi `INSERT`, `COPY`, dan `UPDATE` dapat membuat peristiwa ini.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `Lock:extend` muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

Lonjakan sisipan atau pembaruan bersamaan ke tabel yang sama

Mungkin terdapat peningkatan jumlah sesi bersamaan dengan kueri yang disisipkan ke atau memperbarui tabel yang sama.

Bandwidth jaringan tidak cukup

Bandwidth jaringan pada instans DB mungkin tidak cukup untuk kebutuhan komunikasi penyimpanan dari beban kerja saat ini. Hal ini dapat berkontribusi pada latensi penyimpanan yang menyebabkan peningkatan peristiwa `Lock:extend`.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengurangi sisipan dan pembaruan bersamaan ke relasi yang sama
- Meningkatkan bandwidth jaringan

Mengurangi sisipan dan pembaruan bersamaan ke relasi yang sama

Pertama, ketahui apakah terdapat peningkatan pada metrik `tup_inserted` dan `tup_updated` dengan disertai peningkatan pada peristiwa tunggu ini. Jika demikian, periksa relasi yang memiliki

pertentangan tinggi untuk operasi penyisipan dan pembaruan. Untuk menentukan hal ini, buat kueri tampilan `pg_stat_all_tables` untuk nilai pada bidang `n_tup_ins` dan `n_tup_upd`. Untuk informasi tentang tampilan `pg_stat_all_tables`, lihat pg_stat_statements pada dokumentasi PostgreSQL.

Untuk mendapatkan informasi selengkapnya tentang pemblokiran dan kueri yang diblokir, buat kueri `pg_stat_activity` seperti contoh berikut:

```
SELECT
    blocked.pid,
    blocked.usename,
    blocked.query,
    blocking.pid AS blocking_id,
    blocking.query AS blocking_query,
    blocking.wait_event AS blocking_wait_event,
    blocking.wait_event_type AS blocking_wait_event_type
FROM pg_stat_activity AS blocked
JOIN pg_stat_activity AS blocking ON blocking.pid = ANY(pg_blocking_pids(blocked.pid))
where
blocked.wait_event = 'extend'
and blocked.wait_event_type = 'Lock';

   pid  | usename  |            query             | blocking_id |
            blocking_query                          | blocking_wait_event |
 blocking_wait_event_type
  ------+----------+------------------------------+-------------
+-------------------------------------------------------------------
+---------------------+--------------------------
   7143 |  myuser  | insert into tab1 values (1); |        4600 | INSERT INTO tab1 (a)
 SELECT s FROM generate_series(1,1000000) s; | DataFileExtend      | IO
```

Setelah Anda mengidentifikasi relasi yang berkontribusi untuk meningkatkan peristiwa `Lock:extend`, gunakan teknik berikut untuk mengurangi pertentangan:

- Cari tahu apakah Anda dapat menggunakan partisi untuk mengurangi pertentangan pada tabel yang sama. Memisahkan tuple yang disisipkan atau diperbarui ke dalam partisi yang berbeda dapat mengurangi pertentangan. Untuk informasi tentang partisi, lihat Mengelola partisi PostgreSQL dengan ekstensi pg_partman.
- Jika peristiwa tunggu terutama disebabkan oleh aktivitas pembaruan, pertimbangkan untuk mengurangi nilai faktor isi relasi. Ini dapat mengurangi permintaan untuk blok baru selama pembaruan. Faktor isi adalah parameter penyimpanan untuk tabel yang menentukan jumlah

maksimum ruang untuk mengemas halaman tabel. Ini dinyatakan sebagai persentase dari total ruang untuk sebuah halaman. Untuk informasi selengkapnya tentang parameter faktor isi, lihat BUAT TABEL pada dokumentasi PostgreSQL.

> **Important**
>
> Kami sangat merekomendasikan untuk menguji sistem jika Anda mengubah faktor isi karena mengubah nilai ini dapat berdampak negatif pada performa, tergantung pada beban kerja Anda.

Meningkatkan bandwidth jaringan

Untuk melihat apakah terdapat peningkatan latensi tulis, periksa metrik `WriteLatency` di CloudWatch. Jika ada, gunakan metrik `WriteThroughput` dan `ReadThroughput` Amazon CloudWatch untuk memantau lalu lintas terkait penyimpanan pada klaster DB. Metrik ini dapat membantu Anda menentukan apakah bandwidth jaringan cukup untuk aktivitas penyimpanan beban kerja Anda.

Jika tidak cukup, tingkatkan bandwidth jaringan Anda. Jika instans DB Anda mencapai batas bandwidth jaringan, satu-satunya cara untuk meningkatkan bandwidth adalah meningkatkan ukuran instans DB Anda.

Untuk informasi selengkapnya tentang melihat metrik CloudWatch, lihat CloudWatch Metrik Amazon untuk Amazon Aurora. Untuk informasi tentang performa jaringan untuk setiap kelas instans DB, lihat Spesifikasi perangkat keras kelas instans DB untuk Aurora.

Lock:Relation

Peristiwa `Lock:Relation` terjadi saat kueri menunggu untuk memperoleh kunci pada tabel atau tampilan (relasi) yang saat ini dikunci oleh transaksi lain.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

Konteks

Sebagian besar perintah PostgreSQL secara implisit menggunakan kunci untuk mengontrol akses bersamaan ke data dalam tabel. Anda juga dapat menggunakan kunci ini secara eksplisit pada kode aplikasi Anda dengan perintah `LOCK`. Banyak mode kunci yang tidak kompatibel satu sama lain, dan mereka dapat memblokir transaksi saat mencoba mengakses objek yang sama. Apabila hal ini terjadi, Aurora PostgreSQL membuat peristiwa `Lock:Relation`. Berikut adalah beberapa contoh umum:

- Kunci eksklusif seperti `ACCESS EXCLUSIVE` dapat memblokir semua akses bersamaan. Operasi data definition language (DDL) seperti `DROP TABLE`, `TRUNCATE`, `VACUUM FULL`, dan `CLUSTER` memperoleh kunci `ACCESS EXCLUSIVE` secara implisit. `ACCESS EXCLUSIVE` juga merupakan mode kunci default untuk pernyataan `LOCK TABLE` yang tidak menentukan mode secara eksplisit.
- Menggunakan `CREATE INDEX (without CONCURRENT)` pada tabel bertentangan dengan `UPDATE`, `DELETE`, dan `INSERT` pernyataan data manipulation language (DML), yang memperoleh kunci `ROW EXCLUSIVE`.

Untuk informasi selengkapnya tentang kunci tingkat tabel dan mode kunci yang bertentangan, lihat Penguncian Eksplisit pada dokumentasi PostgreSQL.

Memblokir kueri dan transaksi biasanya membuka blokir dengan salah satu cara berikut:

- Memblokir kueri — Aplikasi dapat membatalkan kueri atau pengguna dapat mengakhiri proses. Mesin juga dapat memaksa kueri untuk berakhir karena batas waktu pernyataan sesi atau mekanisme deteksi kebuntuan.
- Memblokir transaksi — Transaksi berhenti memblokir saat menjalankan pernyataan `ROLLBACK` atau `COMMIT`. Rollback juga terjadi secara otomatis saat sesi terputus dari klien atau masalah jaringan, atau berakhir. Sesi dapat berakhir saat mesin basis data dimatikan, sistem kehabisan memori, dan sebagainya.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `Lock:Relation` terjadi lebih sering dari biasanya, hal tersebut dapat menunjukkan masalah performa. Penyebab umumnya meliputi yang berikut:

Peningkatan sesi bersamaan dengan kunci tabel yang bertentangan

Mungkin ada peningkatan jumlah sesi bersamaan dengan kueri yang mengunci tabel yang sama dengan mode kunci yang bertentangan.

Operasi pemeliharaan

Operasi pemeliharaan kondisi seperti `VACUUM` dan `ANALYZE` secara signifikan dapat meningkatkan jumlah kunci yang bertentangan. `VACUUM FULL` memperoleh kunci `ACCESS EXCLUSIVE`, dan `ANALYSE` memperoleh kunci `SHARE UPDATE EXCLUSIVE`. Kedua jenis kunci tersebut dapat menyebabkan peristiwa tunggu `Lock:Relation`. Operasi pemeliharaan data aplikasi seperti menyegarkan tampilan terwujud juga dapat meningkatkan kueri dan transaksi yang diblokir.

Kunci pada instans pembaca

Mungkin ada pertentangan antara kunci relasi yang dipegang oleh penulis dan pembaca. Saat ini, hanya kunci relasi `ACCESS EXCLUSIVE` yang direplikasi ke instans pembaca. Namun, kunci relasi `ACCESS EXCLUSIVE` akan bertentangan dengan kunci relasi `ACCESS SHARE` yang dipegang oleh pembaca. Hal ini dapat menyebabkan peningkatan peristiwa tunggu relasi kunci pada pembaca.

Tindakan

Kami merekomendasikan berbagai tindakan tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Mengurangi dampak pemblokiran pernyataan SQL
- Minimalkan efek operasi pemeliharaan
- Periksa kunci pembaca

Mengurangi dampak pemblokiran pernyataan SQL

Untuk mengurangi dampak pemblokiran pernyataan SQL, ubah kode aplikasi Anda jika memungkinkan. Berikut adalah dua teknik umum untuk mengurangi pemblokiran:

- Gunakan pilihan `NOWAIT` — Beberapa perintah SQL, seperti pernyataan `SELECT` dan `LOCK`, mendukung pilihan ini. Arahan `NOWAIT` membatalkan kueri permintaan kunci jika kunci tidak dapat segera diperoleh. Teknik ini dapat membantu mencegah tumpukan sesi yang diblokir di belakang yang disebabkan oleh pemblokiran sesi.

Misalnya: Transaksi A sedang menunggu kunci yang dipegang oleh transaksi B. Jika B meminta kunci pada tabel yang dikunci oleh transaksi C, transaksi A mungkin diblokir hingga transaksi C selesai. Namun, jika transaksi B menggunakan `NOWAIT` saat meminta kunci pada C, hal ini dapat memicu gagal cepat (fail fast) dan memastikan bahwa transaksi A tidak harus menunggu tanpa batas waktu.

- Gunakan `SET lock_timeout` - Tetapkan nilai `lock_timeout` untuk membatasi waktu tunggu pernyataan SQL untuk memperoleh kunci pada relasi. Jika kunci tidak diperoleh dalam batas waktu yang ditentukan, transaksi yang meminta kunci akan dibatalkan. Tetapkan nilai ini pada tingkat sesi.

Minimalkan efek operasi pemeliharaan

Penting untuk memelihara operasi seperti `VACUUM` dan `ANALYZE`. Kami merekomendasikan Anda untuk tidak menonaktifkannya karena Anda menemukan peristiwa tunggu `Lock:Relation` yang terkait dengan operasi pemeliharaan ini. Pendekatan berikut dapat meminimalkan efek operasi ini:

- Jalankan pemeliharaan operasi secara manual di luar jam sibuk.
- Untuk mengurangi waktu tunggu `Lock:Relation` yang disebabkan oleh tugas autovacuum, lakukan penyetelan autovacuum yang diperlukan. Untuk informasi tentang menyetel autovacuum, lihat Bekerja dengan autovacuum PostgreSQL di Amazon RDS pada Panduan Pengguna Amazon RDS.

Periksa kunci pembaca

Anda dapat melihat bagaimana sesi bersamaan pada kunci yang mungkin dipegang oleh penulis dan pembaca yang memblokir satu sama lain. Salah satu cara untuk melakukannya adalah dengan menjalankan kueri yang mengembalikan jenis kunci dan relasi. Pada tabel, Anda dapat menemukan urutan kueri untuk dua sesi bersamaan tersebut, sesi penulis (kolom kiri) dan sesi pembaca (kolom kanan).

Proses pemutaran ulang menunggu durasi `max_standby_streaming_delay` sebelum membatalkan kueri pembaca. Seperti yang ditunjukkan pada contoh, batas waktu kunci 100 md jauh di bawah default `max_standby_streaming_delay` 30 detik. Waktu kunci habis sebelum menjadi masalah.

Sesi penulis

```
export WRITER=aurorapg1.1234567891
0.us-west-1.rds.amazonaws.com

psql -h $WRITER
psql (15devel, server 10.14)
Type "help" for help.
```

Sesi pembaca

```
export READER=aurorapg2.1234567891
0.us-west-1.rds.amazonaws.com

psql -h $READER
psql (15devel, server 10.14)
Type "help" for help.
```

Sesi penulis membuat tabel `t1` pada instans penulis. Kunci `ACCESS EXCLUSIVE` segera diperoleh penulis, dengan asumsi bahwa tidak ada kueri yang bertentangan pada penulis.

```
postgres=> CREATE TABLE t1(b
 integer);
CREATE TABLE
```

Sesi pembaca menetapkan interval batas waktu kunci 100 milidetik.

```
postgres=> SET lock_timeout=100;
SET
```

Sesi pembaca mencoba membaca data dari tabel `t1` pada instans pembaca.

```
postgres=> SELECT * FROM t1;
 b
---
(0 rows)
```

Sesi penulis membatalkan `t1`.

```
postgres=> BEGIN;
BEGIN
postgres=> DROP TABLE t1;
DROP TABLE
postgres=>
```

Waktu kueri habis dan dibatalkan pada pembaca.

Sesi penulis

Sesi pembaca

```
postgres=> SELECT * FROM t1;
ERROR:  canceling statement due to
 lock timeout
LINE 1: SELECT * FROM t1;
                      ^
```

Sesi pembaca membuat kueri `pg_locks` dan `pg_stat_activity` untuk menentukan penyebab kesalahan. Hasilnya menunjukkan bahwa proses `aurora wal replay` memegang kunci `ACCESS EXCLUSIVE` di atas tabel `t1`.

```
postgres=> SELECT locktype, relation,
 mode, backend_type
postgres-> FROM pg_locks l, pg_stat_a
ctivity t1
postgres-> WHERE l.pid=t1.pid AND
 relation = 't1'::regclass::oid;
 locktype | relation |        mode
    |   backend_type
----------+----------+----------
-----------+------------------
 relation | 68628525 | AccessExc
lusiveLock | aurora wal replay
(1 row)
```

Lock:transactionid

Peristiwa `Lock:transactionid` terjadi saat transaksi sedang menunggu kunci tingkat baris.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

Konteks

Peristiwa `Lock:transactionid` terjadi saat transaksi mencoba memperoleh kunci tingkat baris yang telah diberikan untuk transaksi yang sedang berlangsung pada saat bersamaan. Sesi yang menunjukkan peristiwa tunggu `Lock:transactionid` diblokir karena kunci ini. Setelah transaksi pemblokiran berakhir dengan pernyataan `COMMIT` atau `ROLLBACK`, transaksi yang diblokir dapat dilanjutkan.

Semantik kontrol bersamaan multiversi dari Aurora PostgreSQL menjamin bahwa pembaca tidak memblokir penulis dan penulis tidak memblokir pembaca. Agar pertentangan tingkat baris terjadi, transaksi yang memblokir dan diblokir harus mengeluarkan pernyataan yang bertentangan dari jenis berikut:

- `UPDATE`
- `SELECT … FOR UPDATE`
- `SELECT … FOR KEY SHARE`

Pernyataan `SELECT … FOR KEY SHARE` adalah kasus khusus. Basis data menggunakan klausa `FOR KEY SHARE` untuk mengoptimalkan performa integritas referensial. Kunci tingkat baris pada baris dapat memblokir perintah `INSERT`, `UPDATE`, dan `DELETE` pada tabel lain yang mereferensikan baris tersebut.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa ini ditampilkan lebih dari biasanya, penyebabnya biasanya pernyataan `UPDATE`, `SELECT … FOR UPDATE`, atau `SELECT … FOR KEY SHARE` yang dikombinasikan dengan kondisi berikut.

Topik

- Konkurensi tinggi
- Idle pada transaksi
- Transaksi jangka panjang

Konkurensi tinggi

Aurora PostgreSQL dapat menggunakan semantik penguncian tingkat baris terperinci. Probabilitas pertentangan tingkat baris meningkat saat kondisi berikut terpenuhi:

- Beban kerja bersamaan tingkat tinggi berselisih untuk baris yang sama.
- Konkurensi meningkat.

Idle pada transaksi

Terkadang kolom `pg_stat_activity.state` menunjukkan nilai `idle in transaction`. Nilai ini ditampilkan untuk sesi yang telah memulai transaksi, tetapi belum mengeluarkan `COMMIT` atau `ROLLBACK`. Jika nilai `pg_stat_activity.state` tidak `active`, kueri yang ditampilkan pada `pg_stat_activity` adalah yang terbaru untuk diselesaikan. Sesi pemblokiran tidak secara aktif memproses kueri karena transaksi yang terbuka menahan kunci.

Jika transaksi idle memperoleh kunci tingkat baris, hal tersebut mungkin mencegah sesi lain memperolehnya. Kondisi ini menyebabkan peristiwa tunggu `Lock:transactionid` yang sering terjadi. Untuk mendiagnosis masalah, periksa output dari `pg_stat_activity` dan `pg_locks`.

Transaksi jangka panjang

Transaksi yang berjalan dalam jangka panjang mendapatkan kunci untuk jangka panjang. Kunci yang sudah lama dipegang ini dapat memblokir transaksi lain agar tidak berlangsung.

Tindakan

Penguncian baris adalah pertentangan antara pernyataan `UPDATE`, `SELECT … FOR UPDATE`, atau `SELECT … FOR KEY SHARE`. Sebelum mencoba solusi, cari tahu kapan pernyataan ini berjalan pada baris yang sama. Gunakan informasi ini untuk memilih strategi yang dijelaskan pada bagian berikut.

Topik

- Menanggapi konkurensi tinggi
- Menanggapi transaksi idle
- Menanggapi transaksi jangka panjang

Menanggapi konkurensi tinggi

Jika masalahnya pada konkurensi, coba salah satu teknik berikut:

- Menurunkan konkurensi pada aplikasi. Misalnya, mengurangi jumlah sesi aktif.
- Menerapkan pool koneksi. Untuk mempelajari cara mengumpulkan koneksi dengan Proksi RDS, lihat Menggunakan Proksi Amazon RDS untuk Aurora.
- Desain aplikasi atau model data untuk menghindari perselisihan pernyataan `UPDATE` dan `SELECT … FOR UPDATE`. Anda juga dapat mengurangi jumlah kunci asing yang diakses oleh pernyataan `SELECT … FOR KEY SHARE`.

Menanggapi transaksi idle

Jika `pg_stat_activity.state` menampilkan `idle in transaction`, gunakan strategi berikut:

- Aktifkan autocommit saat memungkinkan. Pendekatan ini mencegah transaksi memblokir transaksi lain sambil menunggu `COMMIT` atau `ROLLBACK`.
- Cari jalur kode yang tidak memiliki `COMMIT`, `ROLLBACK`, atau `END`.
- Pastikan logika penanganan pengecualian pada aplikasi Anda selalu memiliki jalur ke `end of transaction` yang valid.
- Pastikan bahwa aplikasi Anda memproses hasil kueri setelah mengakhiri transaksi dengan `COMMIT` atau `ROLLBACK`.

Menanggapi transaksi jangka panjang

Jika transaksi jangka panjang menyebabkan `Lock:transactionid` yang sering terjadi, cobalah strategi berikut:

- Jauhkan kunci baris dari transaksi jangka panjang.
- Batasi panjang kueri dengan menerapkan autocommit bila memungkinkan.

Lock:tuple

Peristiwa `Lock:tuple` terjadi saat proses backend menunggu perolehan kunci pada tuple.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

Konteks

Peristiwa `Lock:tuple` menunjukkan bahwa backend sedang menunggu untuk memperoleh kunci pada tuple, sementara backend lain memegang kunci yang bertentangan pada tuple yang sama. Tabel berikut menggambarkan skenario saat sesi membuat peristiwa `Lock:tuple`.

Wakt	Sesi 1	Sesi 2	Sesi 3
t1	Memulai transaksi.		
t2	Memperbarui baris 1.		
t3		Memperbarui baris 1. Sesi tersebut mendapatk an kunci eksklusif pada tuple, lalu menunggu sesi 1 untuk melepaskan kunci dengan metode commit atau rollback.	
t4			Memperbarui baris 1. Sesi tersebut menunggu sesi 2 untuk melepaskan kunci eksklusif pada tuple.

Atau Anda dapat menyimulasikan peristiwa tunggu ini dengan menggunakan alat benchmarking `pgbench`. Konfigurasikan jumlah sesi bersamaan yang tinggi untuk memperbarui baris yang sama pada tabel dengan file SQL khusus.

Untuk mempelajari selengkapnya tentang mode kunci yang bertentangan, lihat Penguncian Eksplisit pada dokumentasi PostgreSQL. Untuk mempelajari selengkapnya tentang `pgbench`, lihat pgbench pada dokumentasi PostgreSQL.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa ini ditampilkan lebih dari biasanya, mungkin menunjukkan masalah performa, penyebab umumnya terdiri dari berikut:

- Sesi bersamaan dalam jumlah yang tinggi mencoba mendapatkan kunci yang bertentangan untuk tuple yang sama dengan menjalankan pernyataan `UPDATE` atau `DELETE`.
- Sesi bersamaan yang tinggi menjalankan pernyataan `SELECT` menggunakan mode kunci `FOR UPDATE` atau `FOR NO KEY UPDATE`.
- Berbagai faktor mendorong aplikasi atau pool koneksi untuk membuka lebih banyak sesi agar menjalankan operasi yang sama. Saat sesi baru mencoba memodifikasi baris yang sama, beban DB dapat melonjak, dan `Lock:tuple` dapat ditampilkan.

Untuk informasi selengkapnya, lihat Penguncian Tingkat Baris pada dokumentasi PostgreSQL.

Tindakan

Kami merekomendasikan berbagai tindakan tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Menyelidiki logika aplikasi
- Menemukan sesi pemblokir
- Mengurangi konkurensi saat tinggi
- Memecahkan masalah

Menyelidiki logika aplikasi

Cari tahu apakah sesi pemblokir telah berada pada status `idle in transaction` dalam waktu yang lama. Jika demikian, coba akhiri sesi pemblokir sebagai solusi jangka pendek. Anda dapat menggunakan fungsi `pg_terminate_backend`. Untuk informasi selengkapnya tentang fungsi ini, lihat Fungsi Pemberi Sinyal Server pada dokumentasi PostgreSQL.

Untuk solusi jangka panjang, lakukan hal berikut:

- Sesuaikan logika aplikasi.
- Gunakan parameter `idle_in_transaction_session_timeout`. Parameter ini mengakhiri sesi apa pun dengan transaksi terbuka yang telah idle lebih lama dari jumlah waktu yang ditentukan. Untuk informasi selengkapnya, lihat Default Koneksi Klien pada dokumentasi PostgreSQL.
- Gunakan autocommit sebanyak mungkin. Untuk informasi selengkapnya, lihat SET AUTOCOMMIT pada dokumentasi PostgreSQL.

Menemukan sesi pemblokir

Saat peristiwa tunggu `Lock:tuple` terjadi, identifikasi pemblokir dan sesi yang diblokir dengan mencari tahu kunci yang bergantung satu sama lain. Untuk informasi selengkapnya, lihat Informasi dependensi kunci pada wiki PostgreSQL. Untuk menganalisis peristiwa `Lock:tuple` yang lampau, gunakan fungsi Aurora `aurora_stat_backend_waits`.

Contoh berikut memberikan kueri semua sesi, memfilter `tuple`, dan mengurutkan berdasarkan `wait_time`.

```
--AURORA_STAT_BACKEND_WAITS
      SELECT a.pid,
             a.usename,
             a.app_name,
             a.current_query,
             a.current_wait_type,
             a.current_wait_event,
             a.current_state,
             wt.type_name AS wait_type,
             we.event_name AS wait_event,
             a.waits,
             a.wait_time
        FROM (SELECT pid,
                     usename,
                     left(application_name,16) AS app_name,
                     coalesce(wait_event_type,'CPU') AS current_wait_type,
                     coalesce(wait_event,'CPU') AS current_wait_event,
                     state AS current_state,
                     left(query,80) as current_query,
                     (aurora_stat_backend_waits(pid)).*
                FROM pg_stat_activity
               WHERE pid <> pg_backend_pid()
                 AND usename<>'rdsadmin') a
NATURAL JOIN aurora_stat_wait_type() wt
```

```
NATURAL JOIN aurora_stat_wait_event() we
WHERE we.event_name = 'tuple'
    ORDER BY a.wait_time;

  pid  | usename | app_name |                 current_query                  |
 current_wait_type | current_wait_event | current_state | wait_type | wait_event |
 waits | wait_time
-------+---------+----------+------------------------------------------------
+-------------------+--------------------+--------------+-----------+------------
+-------+-----------
 32136 | sys     | psql     | /*session3*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000018
 11999 | sys     | psql     | /*session4*/ update tab set col=1 where col=1; | Lock
           | tuple              | active        | Lock      | tuple      |     1 |
 1000024
```

Mengurangi konkurensi saat tinggi

Peristiwa `Lock:tuple` mungkin terjadi terus-menerus, terutama pada waktu beban kerja yang sibuk. Dalam situasi ini, coba kurangi konkurensi tinggi untuk baris yang sangat sibuk. Sering kali, hanya beberapa baris mengontrol antrean atau logika Boolean, yang membuat baris ini sangat sibuk.

Anda dapat mengurangi konkurensi dengan menggunakan pendekatan yang berbeda berdasarkan ketentuan bisnis, logika aplikasi, dan jenis beban kerja. Misalnya, Anda dapat melakukan hal berikut:

- Mendesain ulang tabel dan logika data untuk mengurangi konkurensi tinggi.
- Mengubah logika aplikasi untuk mengurangi konkurensi tinggi di tingkat baris.
- Memanfaatkan dan mendesain ulang kueri dengan kunci tingkat baris.
- Menggunakan klausa `NOWAIT` dengan operasi coba lagi.
- Mempertimbangkan penggunaan kontrol konkurensi logika penguncian hibrid yang optimis.
- Mempertimbangkan perubahan tingkat isolasi basis data.

Memecahkan masalah

`Lock:tuple` dapat terjadi dengan masalah seperti kekurangan CPU atau penggunaan maksimum bandwidth Amazon EBS. Untuk mengurangi masalah, pertimbangkan pendekatan berikut:

- Menaikkan skala jenis kelas instans.

- Mengoptimalkan kueri intensif sumber daya.
- Mengubah logika aplikasi.
- Mengarsipkan data yang jarang diakses.

LWLock:buffer_content (BufferContent)

Peristiwa `LWLock:buffer_content` terjadi saat suatu sesi menunggu untuk membaca atau menulis halaman data di memori, sementara sesi lain mengunci halaman tersebut untuk penulisan. Di Aurora PostgreSQL 13 dan versi yang lebih tinggi, peristiwa tunggu ini disebut `BufferContent`.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

Konteks

Untuk membaca atau memanipulasi data, PostgreSQL mengaksesnya melalui buffer memori bersama. Untuk membaca dari buffer, suatu proses mendapatkan LWLock (lightweight lock) pada konten buffer dalam mode bersama. Untuk menulis ke buffer, aplikasi ini mendapatkan kunci tersebut dalam mode eksklusif. Kunci bersama memungkinkan proses lain untuk secara bersamaan memperoleh kunci bersama pada konten tersebut. Kunci eksklusif mencegah proses lain menerima jenis kunci apa pun.

Peristiwa `LWLock:buffer_content` (`BufferContent`) menunjukkan bahwa beberapa proses mencoba mendapatkan kunci pada konten buffer tertentu.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa `LWLock:buffer_content` (`BufferContent`) muncul lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, berikut adalah penyebab umumnya:

Peningkatan pembaruan bersamaan ke data yang sama

Mungkin terdapat peningkatan jumlah sesi bersamaan dengan kueri yang memperbarui konten buffer yang sama. Pertentangan ini dapat lebih terasa pada tabel dengan banyak indeks.

Data beban kerja tidak ada dalam memori

Saat data yang diproses oleh beban kerja aktif tidak ada dalam memori, peristiwa tunggu ini dapat meningkat. Efek ini terjadi karena proses yang menahan kunci dapat melakukannya lebih lama saat menjalankan operasi I/O disk.

Penggunaan batasan kunci asing yang berlebihan

Batasan kunci asing dapat meningkatkan jumlah waktu suatu proses menahan kunci konten buffer. Efek ini terjadi karena operasi baca memerlukan kunci konten buffer bersama pada kunci yang direferensikan saat kunci tersebut diperbarui.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda. Anda dapat mengidentifikasi peristiwa `LWLock:buffer_content` (`BufferContent`) dengan menggunakan Wawasan Performa Amazon RDS atau dengan membuat kueri tampilan `pg_stat_activity`.

Topik

- Meningkatkan efisiensi dalam memori
- Mengurangi penggunaan batasan kunci asing
- Menghapus indeks yang tidak digunakan

Meningkatkan efisiensi dalam memori

Untuk meningkatkan kemungkinan data beban kerja aktif berada dalam memori, buat partisi tabel atau naikkan skala kelas instans Anda. Untuk informasi tentang kelas instans DB, lihat Kelas instans DB Aurora.

Mengurangi penggunaan batasan kunci asing

Selidiki beban kerja yang mengalami jumlah peristiwa tunggu `LWLock:buffer_content` (`BufferContent`) yang tinggi untuk penggunaan batasan kunci asing. Hapus batasan kunci asing yang tidak perlu.

Menghapus indeks yang tidak digunakan

Untuk beban kerja yang mengalami jumlah peristiwa tunggu `LWLock:buffer_content` (`BufferContent`) yang tinggi, identifikasi indeks yang tidak digunakan, lalu hapus.

LWLock:buffer_mapping

Peristiwa ini terjadi saat sesi menunggu untuk mengaitkan blok data dengan buffer di pool buffer bersama.

> **Note**
>
> Peristiwa ini ditampilkan sebagai `LWLock:buffer_mapping` di Aurora PostgreSQL versi 12 dan yang lebih rendah, serta `LWLock:BufferMapping` di versi 13 dan yang lebih tinggi.

Topik

- Versi mesin yang didukung
- Konteks
- Penyebab
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini relevan untuk Aurora PostgreSQL versi 9.6 dan yang lebih tinggi.

Konteks

Pool buffer bersama adalah area memori Aurora PostgreSQL yang memegang semua halaman yang sedang atau telah digunakan oleh proses. Saat memerlukan halaman, suatu proses membaca halaman ke dalam pool buffer bersama. Parameter `shared_buffers` menetapkan ukuran buffer bersama dan menyimpan area memori untuk menyimpan tabel dan halaman indeks. Jika Anda mengubah parameter ini, pastikan untuk memulai ulang basis data. Untuk informasi selengkapnya, lihat Buffer bersama.

Peristiwa tunggu `LWLock:buffer_mapping` terjadi dalam skenario berikut:

- Suatu proses mencari tabel buffer untuk suatu halaman dan memperoleh kunci pemetaan buffer bersama.

- Suatu proses memuat suatu halaman ke dalam pool buffer dan memperoleh kunci pemetaan buffer eksklusif.
- Suatu proses menghapus suatu halaman dari pool dan memperoleh kunci pemetaan buffer eksklusif.

Penyebab

Saat peristiwa ini ditampilkan lebih dari biasanya, yang mungkin menunjukkan adanya masalah performa, basis data masuk dan keluar dari pool buffer bersama. Penyebab umumnya meliputi yang berikut:

- Kueri yang besar
- Bloat indeks dan tabel
- Pemindaian tabel lengkap
- Ukuran pool bersama yang lebih kecil dari set kerja

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Memantau metrik terkait buffer
- Menilai strategi pengindeksan
- Mengurangi jumlah buffer yang harus dialokasikan dengan cepat

Memantau metrik terkait buffer

Saat peristiwa tunggu `LWLock:buffer_mapping` melonjak, selidiki rasio hit buffer. Anda dapat menggunakan metrik ini untuk mendapatkan pemahaman yang lebih baik tentang apa yang terjadi pada cache buffer. Periksa metrik berikut:

`BufferCacheHitRatio`

Metrik Amazon CloudWatch ini mengukur persentase permintaan yang dilayani oleh cache buffer instans DB pada klaster DB Anda. Anda mungkin melihat penurunan metrik ini sebelum peristiwa tunggu `LWLock:buffer_mapping`.

`blks_hit`

Metrik penghitung Wawasan Performa ini menunjukkan jumlah blok yang diambil dari pool buffer bersama. Setelah peristiwa tunggu `LWLock:buffer_mapping` ditampilkan, Anda mungkin melihat lonjakan pada `blks_hit`.

`blks_read`

Metrik penghitung Wawasan Performa ini menunjukkan jumlah blok yang memerlukan I/O untuk dibaca ke dalam pool buffer bersama. Anda mungkin melihat lonjakan pada `blks_read` sebelum peristiwa tunggu `LWLock:buffer_mapping`.

Menilai strategi pengindeksan

Untuk mengonfirmasi bahwa strategi pengindeksan tidak menurunkan performa, periksa hal berikut:

Bloat indeks

Pastikan bloat indeks dan tabel tidak menyebabkan halaman yang tidak perlu turut dibaca ke buffer bersama. Jika tabel Anda berisi baris yang tidak digunakan, coba arsipkan data dan hapus baris dari tabel. Anda selanjutnya dapat membangun kembali indeks untuk tabel yang diubah ukurannya.

Indeks untuk kueri yang sering digunakan

Untuk menentukan apakah Anda memiliki indeks optimal, pantau metrik mesin DB di Wawasan Performa. Metrik `tup_returned` menunjukkan jumlah baris yang dibaca. Metrik `tup_fetched` menunjukkan jumlah baris yang dikembalikan ke klien. Jika `tup_returned` secara signifikan lebih besar dari `tup_fetched`, data mungkin tidak diindeks dengan benar. Selain itu, statistik tabel Anda mungkin tidak terkini.

Mengurangi jumlah buffer yang harus dialokasikan dengan cepat

Untuk mengurangi peristiwa tunggu `LWLock:buffer_mapping`, coba kurangi jumlah buffer yang harus dialokasikan dengan cepat. Salah satu strateginya adalah dengan melakukan operasi batch yang lebih kecil. Anda mungkin dapat mencapai batch yang lebih kecil dengan partisi tabel.

LWLock:BufferIO (IPC:BufferIO)

Peristiwa `LWLock:BufferIO` terjadi saat Aurora PostgreSQL atau RDS for PostgreSQL sedang menunggu proses lain untuk menyelesaikan operasi input/output (I/O) saat secara bersamaan

mencoba mengakses halaman. Tujuannya adalah agar halaman yang sama dibaca ke dalam buffer bersama.

Topik

- Versi mesin yang relevan
- Konteks
- Penyebab
- Tindakan

Versi mesin yang relevan

Informasi peristiwa tunggu ini relevan untuk semua versi Aurora PostgreSQL. Untuk Aurora PostgreSQL 12 dan versi sebelumnya, peristiwa tunggu ini disebut lwlock:buffer_io, sedangkan pada Aurora PostgreSQL versi 13 disebut lwlock:bufferio. Dari Aurora PostgreSQL versi 14, peristiwa tunggu BufferIO dipindahkan dari jenis peristiwa tunggu `LWLock` ke `IPC` (IPC:BufferIO).

Konteks

Setiap buffer bersama memiliki kunci I/O yang terkait dengan peristiwa tunggu `LWLock:BufferIO`, setiap kali suatu blok (atau halaman) harus diambil di luar pool buffer bersama.

Kunci ini digunakan untuk menangani beberapa sesi yang semuanya memerlukan akses ke blok yang sama. Blok ini harus dibaca dari luar pool buffer bersama, yang ditentukan oleh parameter `shared_buffers`.

Segera setelah halaman dibaca di dalam pool buffer bersama, kunci `LWLock:BufferIO` akan dilepaskan.

> **Note**
>
> Peristiwa tunggu `LWLock:BufferIO` mendahului peristiwa tunggu IO:DataFileRead. Peristiwa tunggu `IO:DataFileRead` terjadi saat data sedang dibaca dari penyimpanan.

Untuk informasi selengkapnya tentang kunci ringan, lihat Ikhtisar Penguncian.

Penyebab

Penyebab umum peristiwa `LWLock:BufferIO` muncul dalam peristiwa tunggu teratas mencakup yang berikut:

- Beberapa backend atau koneksi mencoba mengakses halaman yang sama yang juga menunggu operasi I/O
- Rasio antara ukuran pool buffer bersama (ditentukan oleh parameter `shared_buffers`) dan jumlah buffer yang dibutuhkan oleh beban kerja saat ini
- Ukuran pool buffer bersama tidak seimbang dengan jumlah halaman yang dikosongkan oleh operasi lain
- Indeks besar atau bloat indeks yang memerlukan mesin untuk membaca lebih banyak halaman dari yang diperlukan ke dalam pool buffer bersama
- Kurangnya indeks yang memaksa mesin DB untuk membaca lebih banyak halaman dari tabel daripada yang diperlukan
- Lonjakan mendadak untuk koneksi basis data yang mencoba melakukan operasi pada halaman yang sama

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda:

- Amati metrik Amazon CloudWatch untuk korelasi antara penurunan tajam pada peristiwa tunggu `BufferCacheHitRatio` dan `LWLock:BufferIO`. Efek ini dapat berarti bahwa Anda memiliki pengaturan buffer bersama kecil. Anda mungkin perlu meningkatkan atau menaikkan skala kelas instans DB Anda. Anda dapat membagi beban kerja Anda menjadi lebih banyak simpul pembaca.
- Sesuaikan `max_wal_size` dan `checkpoint_timeout` berdasarkan waktu puncak beban kerja jika Anda melihat `LWLock:BufferIO` bertepatan dengan penurunan metrik `BufferCacheHitRatio`. Lalu, identifikasi kueri yang mungkin menyebabkannya.
- Verifikasi apakah Anda memiliki indeks yang tidak digunakan, lalu hapus.
- Gunakan tabel yang dipartisi (yang juga memiliki indeks yang dipartisi). Dengan melakukan hal ini, Anda dapat menjaga penataan ulang indeks tetap rendah dan mengurangi dampaknya.
- Hindari pengindeksan kolom yang tidak perlu.
- Cegah lonjakan koneksi basis data mendadak dengan menggunakan pool koneksi.
- Batasi jumlah maksimum koneksi ke basis data sebagai tindakan terbaik.

LWLock:lock_manager

Peristiwa ini terjadi saat mesin Aurora PostgreSQL mempertahankan area memori kunci bersama untuk mengalokasikan, memeriksa, dan mendealokasikan kunci saat kunci jalur cepat tidak memungkinkan.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini relevan untuk Aurora PostgreSQL versi 9.6 dan yang lebih tinggi.

Konteks

Saat Anda mengeluarkan pernyataan SQL, Aurora PostgreSQL mencatat kunci untuk melindungi struktur, data, dan integritas basis data Anda selama operasi bersamaan. Mesin dapat mencapai tujuan ini menggunakan kunci jalur cepat atau kunci jalur yang tidak cepat. Kunci jalur yang tidak cepat lebih mahal dan membuat lebih banyak overhead daripada kunci jalur cepat.

Penguncian jalur cepat

Untuk mengurangi overhead kunci yang sering diambil dan dilepaskan, tetapi jarang terjadi konflik, proses backend dapat menggunakan penguncian jalur cepat. Basis data menggunakan mekanisme ini untuk kunci yang memenuhi kriteria berikut:

- Menggunakan metode kunci DEFAULT.
- Mewakili kunci pada relasi basis data, bukan relasi bersama.
- Merupakan kunci lemah yang tidak mungkin bertentangan.
- Mesin dapat dengan cepat memverifikasi bahwa tidak ada kunci yang saling bertentangan.

Mesin tidak dapat menggunakan penguncian jalur cepat jika salah satu kondisi berikut ini benar:

- Kunci tidak memenuhi kriteria sebelumnya.

- Tidak ada lagi slot yang tersedia untuk proses backend.

Untuk informasi selengkapnya tentang penguncian jalur cepat, lihat jalur cepat di README manajer kunci PostgreSQL dan pg-locks pada dokumentasi PostgreSQL.

Contoh masalah penskalaan untuk manajer kunci

Pada contoh ini, tabel `purchases` menyimpan data selama lima tahun, yang dipartisi berdasarkan hari. Setiap partisi memiliki dua indeks. Urutan peristiwa berikut terjadi:

1. Anda membuat kueri data bernilai beberapa hari, yang memerlukan basis data untuk membaca banyak partisi.
2. Basis data membuat entri kunci untuk setiap partisi. Jika indeks partisi adalah bagian dari jalur akses pengoptimal, basis data akan membuat entri kunci untuk mereka juga.
3. Saat jumlah entri kunci yang diminta untuk proses backend yang sama lebih tinggi dari 16, yang merupakan nilai `FP_LOCK_SLOTS_PER_BACKEND`, manajer kunci menggunakan metode kunci jalur yang tidak cepat.

Aplikasi modern mungkin memiliki ratusan sesi. Jika sesi bersamaan membuat kueri induk tanpa pemangkasan partisi yang tepat, basis data mungkin membuat ratusan atau bahkan ribuan kunci jalur yang tidak cepat. Biasanya, saat konkurensi ini lebih tinggi dari jumlah vCPU, peristiwa tunggu `LWLock:lock_manager` akan ditampilkan.

> **Note**
>
> Peristiwa tunggu `LWLock:lock_manager` tidak terkait dengan jumlah partisi atau indeks dalam skema basis data. Sebaliknya, peristiwa tunggu terkait dengan jumlah kunci jalur yang tidak cepat yang harus dikontrol oleh basis data.

Kemungkinan penyebab peningkatan peristiwa tunggu

Saat peristiwa tunggu `LWLock:lock_manager` terjadi lebih dari biasanya, mungkin menunjukkan masalah performa, kemungkinan penyebab lonjakan mendadak adalah berikut:

- Sesi aktif bersamaan menjalankan kueri yang tidak menggunakan kunci jalur cepat. Sesi ini juga melebihi vCPU maksimum.

- Sesi aktif bersamaan dalam jumlah besar mengakses tabel yang banyak dipartisi. Setiap partisi memiliki beberapa indeks.
- Basis data mengalami storm koneksi. Secara default, beberapa aplikasi dan perangkat lunak pool koneksi membuat lebih banyak koneksi saat basis data lambat. Hal ini memperburuk masalah. Sesuaikan perangkat lunak pool koneksi Anda sehingga storm koneksi tidak terjadi.
- Jumlah sesi yang besar membuat kueri tabel induk tanpa memangkas partisi.
- Data definition language (DDL), data manipulation language (DML), atau perintah pemeliharaan secara eksklusif mengunci relasi yang sibuk atau tuple yang sering diakses atau dimodifikasi.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda.

Topik

- Menggunakan pemangkasan partisi
- Menghapus indeks yang tidak diperlukan
- Menyetel kueri untuk penguncian jalur cepat
- Menyetel untuk peristiwa tunggu lainnya
- Mengurangi masalah perangkat keras
- Menggunakan pooler koneksi
- Meningkatkan versi Aurora PostgreSQL Anda

Menggunakan pemangkasan partisi

Pemangkasan partisi adalah strategi pengoptimalan kueri yang mengecualikan partisi yang tidak dibutuhkan dari pemindaian tabel, sehingga meningkatkan performa. Pemangkasan partisi diaktifkan secara default. Jika tidak aktif, aktifkan dengan cara berikut:

```
SET enable_partition_pruning = on;
```

Kueri dapat memanfaatkan pemangkasan partisi saat klausa `WHERE` mereka berisi kolom yang digunakan untuk partisi. Untuk informasi selengkapnya, lihat Pemangkasan Partisi pada dokumentasi PostgreSQL.

Menghapus indeks yang tidak diperlukan

Basis data Anda mungkin berisi indeks yang tidak digunakan atau jarang digunakan. Jika ya, pertimbangkanlah untuk menghapusnya. Lakukan salah satu dari langkah berikut:

- Mempelajari cara menemukan indeks yang tidak perlu dengan membaca Indeks yang Tidak Digunakan di wiki PostgreSQL.
- Menjalankan PG Collector. Skrip SQL ini mengumpulkan informasi basis data dan menyajikannya dalam laporan HTML terkonsolidasi. Memeriksa bagian “Indeks yang tidak digunakan”. Untuk informasi selengkapnya, lihat pg-collector dalam repositori GitHub Lab AWS.

Menyetel kueri untuk penguncian jalur cepat

Untuk mengetahui apakah kueri Anda menggunakan penguncian jalur cepat, buat kueri pada kolom `fastpath` dalam tabel `pg_locks`. Jika kueri Anda tidak menggunakan penguncian jalur cepat, cobalah kurangi jumlah relasi per kueri menjadi kurang dari 16.

Menyetel untuk peristiwa tunggu lainnya

Jika `LWLock:lock_manager` adalah yang pertama atau kedua dalam daftar peristiwa tunggu teratas, periksa apakah peristiwa tunggu berikut juga ditampilkan dalam daftar:

- `Lock:Relation`
- `Lock:transactionid`
- `Lock:tuple`

Jika peristiwa sebelumnya ditampilkan pada posisi tinggi dalam daftar, coba setel peristiwa tunggu ini terlebih dahulu. Peristiwa ini dapat menjadi pendorong untuk `LWLock:lock_manager`.

Mengurangi masalah perangkat keras

Anda mungkin memiliki masalah perangkat keras, seperti kekurangan CPU atau penggunaan maksimum bandwidth Amazon EBS. Jika demikian, coba kurangi masalah perangkat keras. Pertimbangkan tindakan berikut:

- Menaikkan skala kelas instans.
- Mengoptimalkan kueri yang mengonsumsi CPU dan memori dalam jumlah besar.
- Mengubah logika aplikasi.

- Mengarsipkan data.

Untuk informasi selengkapnya tentang CPU, memori, dan bandwidth jaringan EBS, lihat Jenis Instans Amazon RDS.

Menggunakan pooler koneksi

Jika jumlah total koneksi aktif Anda melebihi vCPU maksimum, lebih banyak proses OS memerlukan CPU daripada yang dapat didukung oleh jenis instans Anda. Jika demikian, pertimbangkanlah untuk menggunakan atau menyetel pool koneksi. Untuk informasi selengkapnya tentang vCPU untuk jenis instans Anda, lihat Jenis Instans Amazon RDS.

Untuk informasi selengkapnya tentang pengumpulan koneksi, lihat sumber daya berikut:

- Menggunakan Proksi Amazon RDS untuk Aurora
- pgbouncer
- Pool Koneksi dan Sumber Data pada Dokumentasi PostgreSQL

Meningkatkan versi Aurora PostgreSQL Anda

Jika versi Aurora PostgreSQL Anda saat ini lebih rendah dari 12, tingkatkan ke versi 12 atau yang lebih tinggi. PostgreSQL versi 12 dan 13 memiliki mekanisme partisi yang ditingkatkan. Untuk informasi selengkapnya tentang versi 12, lihat Catatan Rilis PostgreSQL 12.0. Untuk informasi selengkapnya tentang meningkatkan Aurora PostgreSQL, lihat Pembaruan Amazon Aurora PostgreSQL.

LWLock: MultiXact

Peristiwa tunggu `LWLock:MultiXactMemberBuffer`, `LWLock:MultiXactOffsetBuffer`, `LWLock:MultiXactMemberSLRU`, dan `LWLock:MultiXactOffsetSLRU` menunjukkan bahwa sesi sedang menunggu pengambilan daftar transaksi yang memodifikasi baris yang sama pada tabel tertentu.

- `LWLock:MultiXactMemberBuffer` – Suatu proses sedang menunggu I/O pada buffer simple least-recently used (SLRU) untuk anggota multixact.
- `LWLock:MultiXactMemberSLRU` – Suatu proses sedang menunggu akses cache simple least-recently used (SLRU) untuk anggota multixact.

- `LWLock:MultiXactOffsetBuffer` – Suatu proses sedang menunggu I/O pada buffer simple least-recently used (SLRU) untuk offset multixact.
- `LWLock:MultiXactOffsetSLRU` – Suatu proses sedang menunggu akses cache simple least-recently used (SLRU) untuk offset multixact.

Topik

- Versi mesin yang didukung
- Konteks
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

Konteks

Multixact adalah struktur data yang menyimpan daftar ID transaksi (XID) yang memodifikasi baris tabel yang sama. Saat satu transaksi merujuk suatu baris dalam tabel, ID transaksi disimpan pada baris header tabel. Saat beberapa transaksi merujuk baris yang sama pada tabel, daftar ID transaksi disimpan dalam struktur data multixact. Peristiwa tunggu multixact menunjukkan bahwa sesi mengambil dari struktur data daftar transaksi yang merujuk ke baris tertentu dalam tabel.

Kemungkinan penyebab peningkatan peristiwa tunggu

Tiga penyebab umum penggunaan multixact adalah sebagai berikut:

- Subtransaksi dari titik simpan eksplisit – Secara eksplisit membuat titik simpan pada transaksi Anda yang memunculkan transaksi baru untuk baris yang sama. Misalnya, menggunakan `SELECT FOR UPDATE`, lalu `SAVEPOINT`, dan `UPDATE`.

  Beberapa driver, object-relational mappers (ORM), dan lapisan abstraksi memiliki opsi konfigurasi untuk secara otomatis membungkus semua operasi dengan titik simpan. Hal ini dapat membuat banyak peristiwa tunggu multixact di beberapa beban kerja. Opsi `autosave` Driver JDBC PostgreSQL adalah contohnya. Untuk informasi selengkapnya, lihat pgJDBC pada dokumentasi JDBC PostgreSQL. Contoh lainnya adalah driver ODBC PostgreSQL dan opsi `protocol`.

Untuk informasi selengkapnya, lihat Opsi Konfigurasi psqlODBC pada dokumentasi driver ODBC PostgreSQL.

- Subtransaksi dari klausa PL/pgSQL EXCEPTION – Setiap klausa `EXCEPTION` yang Anda tulis dalam fungsi atau prosedur PL/pgSQL membuat `SAVEPOINT` secara internal.
- Kunci asing – Beberapa transaksi memperoleh kunci bersama pada catatan induk.

Saat baris tertentu disertakan dalam beberapa operasi transaksi, pemrosesan baris memerlukan pengambilan ID transaksi dari daftar `multixact`. Jika pencarian tidak mendapatkan multixact dari cache memori, struktur data harus dibaca dari lapisan penyimpanan Aurora. I/O dari penyimpanan ini berarti kueri SQL dapat memerlukan waktu lebih lama. Kesalahan cache memori dapat mulai terjadi dengan penggunaan yang berat karena jumlah transaksi ganda yang besar. Semua faktor ini berkontribusi pada peningkatan peristiwa tunggu ini.

Tindakan

Kami merekomendasikan berbagai tindakan, tergantung pada penyebab peristiwa tunggu Anda. Beberapa tindakan ini dapat membantu untuk segera mengurangi peristiwa tunggu. Namun, yang lainnya mungkin memerlukan penyelidikan dan koreksi untuk menskalakan beban kerja Anda.

Topik

- Lakukan pembekuan vakum di atas tabel dengan peristiwa tunggu ini
- Tingkatkan frekuensi autovacuum pada tabel dengan peristiwa tunggu ini
- Meningkatkan parameter memori
- Mengurangi transaksi jangka panjang
- Tindakan jangka panjang

Lakukan pembekuan vakum di atas tabel dengan peristiwa tunggu ini

Jika peristiwa tunggu ini melonjak mendadak dan mempengaruhi lingkungan produksi, Anda dapat menggunakan salah satu metode sementara berikut untuk mengurangi penghitungannya.

- Gunakan VACUUM FREEZE pada tabel atau partisi tabel yang terpengaruh untuk segera menyelesaikan masalah. Untuk informasi selengkapnya, lihat VACUUM.
- Gunakan klausa VACUUM (FREEZE, INDEX_CLEANUP FALSE) untuk melakukan vakum cepat dengan melewati indeks. Untuk informasi selengkapnya, lihat Memvakum tabel secepat mungkin.

Tingkatkan frekuensi autovacuum pada tabel dengan peristiwa tunggu ini

Setelah memindai semua tabel di semua basis data, VACUUM akan menghapus multixact, dan nilai multixact tertua akan dimajukan. Untuk informasi selengkapnya, lihat Multixact dan Wraparound. Untuk menjaga LWLock: MultiXact tunggu acara seminimal mungkin, Anda harus menjalankan VACUUM sesering yang diperlukan. Untuk melakukannya, pastikan bahwa VACUUM di klaster DB Aurora PostgreSQL Anda dikonfigurasi secara optimal.

Jika menggunakan VACUUM FREEZE pada tabel atau partisi tabel yang terpengaruh menyelesaikan masalah peristiwa tunggu, sebaiknya gunakan penjadwal, seperti `pg_cron`, untuk melakukan VACUUM daripada menyesuaikan autovacuum pada tingkat instans.

Agar autovacuum terjadi lebih sering, Anda dapat mengurangi nilai parameter penyimpanan `autovacuum_multixact_freeze_max_age` pada tabel yang terpengaruh. Untuk informasi selengkapnya, lihat autovacuum_multixact_freeze_max_age.

Meningkatkan parameter memori

Anda dapat mengatur parameter berikut di tingkat klaster, sehingga semua instans pada klaster Anda tetap konsisten. Hal ini membantu mengurangi peristiwa tunggu pada beban kerja Anda. Kami merekomendasikan Anda untuk tidak mengatur nilai-nilai ini terlalu tinggi sehingga kehabisan memori.

- `multixact_offsets_cache_size` ke 128
- `multixact_members_cache_size` ke 256

Anda harus boot ulang instans agar perubahan parameter diterapkan. Dengan parameter ini, Anda dapat menggunakan lebih banyak RAM instans untuk menyimpan struktur multixact di memori sebelum tumpah ke disk.

Mengurangi transaksi jangka panjang

Transaksi jangka panjang menyebabkan vakum mempertahankan informasinya hingga transaksi tersebut dilakukan atau hingga transaksi hanya dibaca ditutup. Kami merekomendasikan Anda untuk secara proaktif memantau dan mengelola transaksi jangka panjang. Untuk informasi selengkapnya, lihat Basis data telah lama idle dalam koneksi transaksi. Cobalah untuk memodifikasi aplikasi Anda untuk menghindari atau meminimalkan penggunaan transaksi jangka panjang Anda.

Tindakan jangka panjang

Periksa beban kerja Anda untuk menemukan penyebab limpahan multixact. Anda harus memperbaiki masalah tersebut untuk menskalakan beban kerja Anda dan mengurangi peristiwa tunggu.

- Anda harus menganalisis DDL (data definition language) yang digunakan untuk membuat tabel Anda. Pastikan bahwa struktur dan indeks tabel dirancang dengan baik.
- Saat tabel yang terpengaruh memiliki kunci asing, tentukan apakah diperlukan atau jika ada cara lain untuk memberlakukan integritas referensial.
- Saat sebuah tabel memiliki indeks besar yang tidak digunakan, hal tersebut dapat menyebabkan autovacuum tidak sesuai dengan beban kerja Anda dan mungkin memblokirnya agar tidak berjalan. Untuk menghindari hal ini, periksa indeks yang tidak digunakan dan hapus sepenuhnya. Untuk informasi selengkapnya, lihat Mengelola autovacuum dengan indeks besar.
- Kurangi penggunaan titik simpan dalam transaksi Anda.

Timeout:PgSleep

Peristiwa `Timeout:PgSleep` terjadi saat proses server telah memanggil fungsi `pg_sleep` dan menunggu batas waktu tidur berakhir.

Topik

- Versi mesin yang didukung
- Kemungkinan penyebab peningkatan peristiwa tunggu
- Tindakan

Versi mesin yang didukung

Informasi peristiwa tunggu ini didukung untuk semua versi Aurora PostgreSQL.

Kemungkinan penyebab peningkatan peristiwa tunggu

Peristiwa tunggu ini terjadi saat aplikasi, fungsi yang tersimpan, atau pengguna mengeluarkan pernyataan SQL yang memanggil salah satu fungsi berikut:

- `pg_sleep`
- `pg_sleep_for`

- `pg_sleep_until`

Fungsi sebelumnya menunda eksekusi hingga waktu yang ditentukan dalam detik telah berlalu. Misalnya, `SELECT pg_sleep(1)` dijeda selama 1 detik. Untuk informasi selengkapnya, lihat Menunda Eksekusi pada dokumentasi PostgreSQL.

Tindakan

Identifikasi pernyataan yang menjalankan fungsi `pg_sleep`. Tentukan apakah penggunaan fungsi tersebut sesuai.

Menyesuaikan Aurora PostgreSQL dengan wawasan proaktif Amazon DevOps Guru

Wawasan proaktif DevOps Guru mendeteksi kondisi pada RDS Anda untuk klaster DB Aurora PostgreSQL yang dapat menyebabkan masalah, dan memberi tahu Anda tentangnya sebelum terjadi. DevOps Guru dapat melakukan hal berikut:

- Mencegah banyak masalah umum pada basis data dengan memeriksa silang konfigurasi basis data terhadap pengaturan umum yang direkomendasikan.
- Memberi tahu Anda tentang masalah kritis dalam armada yang, jika dibiarkan tanpa diperiksa, dapat menyebabkan masalah yang lebih besar di kemudian hari.
- Memberi tahu Anda tentang masalah yang baru ditemukan.

Setiap wawasan proaktif berisi analisis penyebab masalah dan rekomendasi untuk tindakan korektif.

Topik

- Basis data telah lama berjalan idle dalam koneksi transaksi

Basis data telah lama berjalan idle dalam koneksi transaksi

Koneksi ke basis berada dalam status `idle in transaction` selama lebih dari 1.800 detik.

Topik

- Versi mesin yang didukung
- Konteks

- Kemungkinan penyebab masalah ini
- Tindakan
- Metrik terkait

Versi mesin yang didukung

Informasi wawasan ini didukung untuk semua versi Aurora PostgreSQL.

Konteks

Transaksi dalam status `idle in transaction` dapat menahan kunci yang memblokir kueri lain. Ini juga dapat mencegah `VACUUM` (termasuk autovacuum) membersihkan baris mati, yang menyebabkan penggelembungan indeks atau tabel atau ID transaksi tumpang tindih.

Kemungkinan penyebab masalah ini

Transaksi yang dimulai dalam sesi interaktif dengan BEGIN atau START TRANSACTION belum berakhir dengan menggunakan perintah COMMIT, ROLLBACK, atau END. Hal ini menyebabkan status transaksi berubah ke `idle in transaction`.

Tindakan

Anda dapat menemukan transaksi idle dengan mengkueri `pg_stat_activity`.

Di klien SQL Anda, jalankan kueri berikut untuk mencantumkan semua koneksi dalam status `idle in transaction` dan mengurutkannya berdasarkan durasi:

```
SELECT now() - state_change as idle_in_transaction_duration, now() - xact_start as
 xact_duration,*
FROM  pg_stat_activity
WHERE state  = 'idle in transaction'
AND   xact_start is not null
ORDER BY 1 DESC;
```

Kami merekomendasikan tindakan yang berbeda bergantung pada penyebab wawasan Anda.

Topik

- Transaksi akhir

- Mengakhiri koneksi
- Konfigurasikan parameter idle_in_transaction_session_timeout
- Memeriksa status AUTOCOMMIT
- Memeriksa logika transaksi dalam kode aplikasi Anda

Transaksi akhir

Ketika Anda memulai transaksi dalam sesi interaktif dengan BEGIN atau START TRANSACTION, status akan berubah ke `idle in transaction`. Status ini tidak akan berubah hingga Anda mengakhiri transaksi dengan mengeluarkan perintah COMMIT, ROLLBACK, END atau memutuskan koneksi sepenuhnya untuk meluncurkan kembali transaksi.

Mengakhiri koneksi

Akhiri koneksi dengan transaksi idle menggunakan kueri berikut:

```
SELECT pg_terminate_backend(pid);
```

pid adalah ID proses koneksi.

Konfigurasikan parameter idle_in_transaction_session_timeout

Konfigurasikan parameter `idle_in_transaction_session_timeout` di grup parameter baru. Dengan mengonfigurasi parameter ini, intervensi manual tidak diperlukan untuk mengakhiri status idle yang lama dalam transaksi. Untuk informasi lebih lanjut tentang parameter ini, lihat dokumentasi PostgreSQL.

Pesan berikut akan dilaporkan dalam file log PostgreSQL setelah koneksi dihentikan jika transaksi berada dalam status idle_in_transaction yang lebih lama dari waktu yang ditentukan.

```
FATAL: terminating connection due to idle in transaction timeout
```

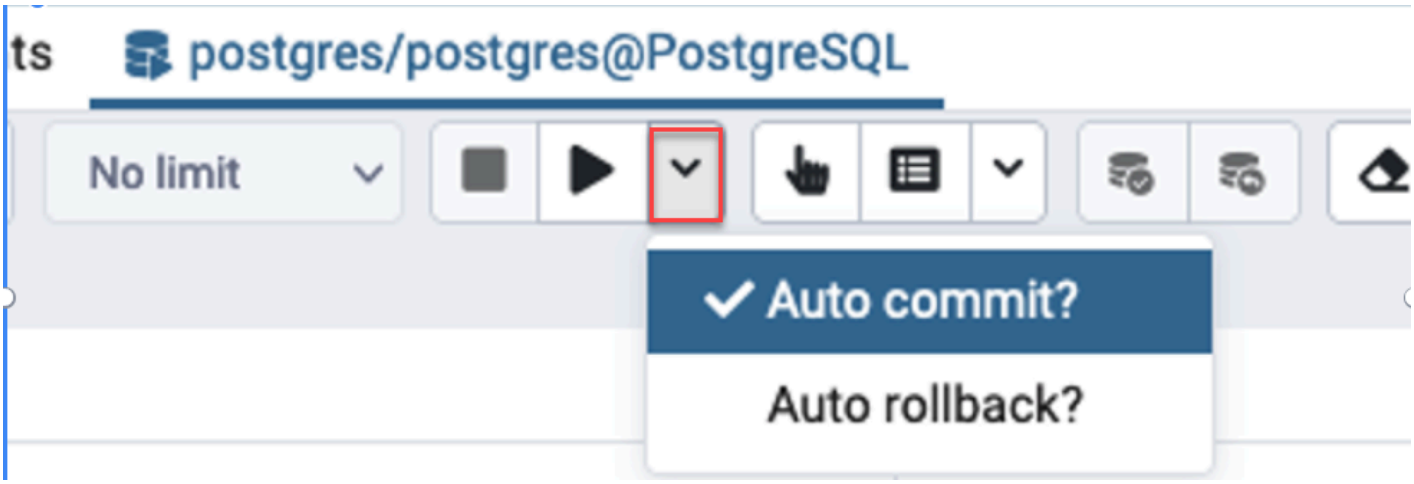
Memeriksa status AUTOCOMMIT

AUTOCOMMIT diaktifkan secara default. Namun, jika dinonaktifkan secara tidak sengaja di klien, pastikan Anda mengaktifkannya kembali.

- Di klien psql Anda, jalankan perintah berikut:

```
postgres=> \set AUTOCOMMIT on
```

- Di pgadmin, aktifkan dengan memilih opsi AUTOCOMMIT dari tanda panah bawah.



Memeriksa logika transaksi dalam kode aplikasi Anda

Selidiki logika aplikasi Anda untuk kemungkinan masalah. Pertimbangkan tindakan berikut:

- Periksa apakah JDBC auto commit diatur ke true dalam aplikasi Anda. Pertimbangkan juga untuk menggunakan perintah `COMMIT` eksplisit dalam kode Anda.
- Periksa logika penanganan kesalahan untuk mengetahui apakah transaksi ditutup setelah kesalahan terjadi.
- Periksa apakah aplikasi Anda membutuhkan waktu lama untuk memproses baris yang ditampilkan oleh kueri saat transaksi terbuka. Jika demikian, pertimbangkan untuk mengodekan aplikasi guna menutup transaksi sebelum memproses baris.
- Periksa apakah transaksi berisi banyak operasi yang berjalan lama. Jika demikian, bagi satu transaksi menjadi beberapa transaksi.

Metrik terkait

Metrik PI berikut terkait dengan wawasan ini:

- idle_in_transaction_count - Jumlah sesi dalam status `idle in transaction`.
- idle_in_transaction_max_time - Durasi transaksi yang berjalan paling lama dalam status `idle in transaction`.

Praktik terbaik dengan Amazon Aurora PostgreSQL

Di bagian berikut ini, Anda dapat menemukan beberapa praktik terbaik untuk mengelola klaster DB Amazon Aurora PostgreSQL. Pastikan juga untuk meninjau tugas pemeliharaan dasar. Untuk informasi selengkapnya, lihat Mengelola Amazon Aurora PostgreSQL.

Topik

- Menghindari performa lambat, pengaktifan ulang otomatis, dan failover untuk instans DB Aurora PostgreSQL
- Mendiagnosis bloat tabel dan indeks
- Manajemen memori yang ditingkatkan dalam Aurora PostgreSQL
- Failover cepat dengan Amazon Aurora PostgreSQL
- Pemulihan cepat setelah failover dengan manajemen cache klaster untuk Aurora PostgreSQL
- Mengelola churn koneksi Aurora PostgreSQL dengan pooling
- Menyetel parameter memori untuk Aurora PostgreSQL
- Menggunakan CloudWatch metrik Amazon untuk menganalisis penggunaan sumber daya untuk Aurora PostgreSQL
- Menggunakan replikasi logis untuk melakukan upgrade versi mayor untuk Aurora PostgreSQL
- Pemecahan masalah penyimpanan

Menghindari performa lambat, pengaktifan ulang otomatis, dan failover untuk instans DB Aurora PostgreSQL

Jika Anda menjalankan beban kerja berat atau beban kerja yang melonjak melampaui sumber daya yang dialokasikan untuk instans DB Anda, Anda dapat menghabiskan sumber daya yang Anda gunakan untuk menjalankan aplikasi dan basis data Aurora. Untuk mendapatkan metrik tentang instans basis data seperti pemanfaatan CPU, penggunaan memori, dan jumlah koneksi basis data yang digunakan, Anda dapat merujuk ke metrik yang disediakan oleh Amazon CloudWatch, Wawasan Performa, dan Pemantauan yang Ditingkatkan. Untuk informasi selengkapnya tentang pemantauan instans DB, lihat Memantau metrik di klaster Amazon Aurora.

Jika beban kerja Anda menghabiskan sumber daya yang Anda gunakan, instans DB Anda mungkin akan menjadi lambat, diaktifkan ulang, atau bahkan melakukan failover ke instans DB lain. Untuk menghindari hal ini, pantau pemanfaatan sumber daya Anda, periksa beban kerja yang berjalan pada instans DB Anda, dan buat pengoptimalan jika diperlukan. Jika pengoptimalan tidak meningkatkan

metrik instans dan mengurangi kehabisan sumber daya, pertimbangkan untuk menaikkan skala instans DB Anda sebelum mencapai batasnya. Untuk informasi selengkapnya tentang kelas instans DB yang tersedia dan spesifikasinya, lihat Kelas instans DB Aurora.

Mendiagnosis bloat tabel dan indeks

Anda dapat menggunakan Multiversion Concurrency Control (MVCC) PostgreSQL untuk membantu menjaga integritas data. MVCC PostgreSQL beroperasi dengan menyimpan salinan internal baris yang diperbarui atau dihapus (juga disebut tuple) sampai transaksi dikomit atau di-rollback. Salinan internal yang disimpan ini tidak terlihat oleh pengguna. Namun, bloat tabel dapat terjadi ketika salinan yang tidak terlihat tersebut tidak dibersihkan secara teratur oleh utilitas VACUUM atau AUTOVACUUM. Bloat tabel yang tidak terpantau dapat menimbulkan peningkatan biaya penyimpanan dan memperlambat kecepatan pemrosesan Anda.

Dalam banyak kasus, pengaturan default untuk VACUUM atau AUTOVACUUM di Aurora cukup untuk menangani bloat tabel yang tidak diinginkan. Namun, Anda sebaiknya memeriksa bloat jika aplikasi Anda mengalami kondisi berikut:

- Memproses sejumlah besar transaksi dalam waktu yang relatif singkat di antara proses VACUUM.
- Beperforma buruk dan kehabisan penyimpanan.

Untuk memulai, kumpulkan informasi paling akurat tentang jumlah ruang yang digunakan oleh tuple nonaktif dan jumlah yang dapat Anda pulihkan dengan membersihkan bloat tabel dan indeks. Untuk melakukannya, gunakan ekstensi `pgstattuple` untuk mengumpulkan statistik tentang klaster Aurora Anda. Untuk informasi selengkapnya, lihat pgstattuple. Hak akses untuk menggunakan ekstensi `pgstattuple` dibatasi pada peran `pg_stat_scan_tables` dan superuser basis data.

Untuk membuat ekstensi `pgstattuple` di Aurora, hubungkan sesi klien ke klaster, misalnya, psql atau pgAdmin, lalu gunakan perintah berikut:

```
CREATE EXTENSION pgstattuple;
```

Buat ekstensi di setiap basis data yang ingin Anda buat profilnya. Setelah membuat ekstensi, gunakan antarmuka baris perintah (CLI) untuk mengukur jumlah ruang yang tidak dapat digunakan yang dapat Anda klaim kembali. Sebelum mengumpulkan statistik, ubah grup parameter klaster dengan menetapkan AUTOVACUUM ke 0. Pengaturan 0 akan mencegah Aurora membersihkan tuple nonaktif secara otomatis yang ditinggalkan oleh aplikasi Anda, yang dapat memengaruhi keakuratan hasil. Masukkan perintah berikut untuk membuat tabel sederhana:

```
postgres=> CREATE TABLE lab AS SELECT generate_series (0,100000);
SELECT 100001
```

Dalam contoh berikut, kita menjalankan kueri dengan AUTOVACUUM diaktifkan untuk klaster DB. `dead_tuple_count` adalah 0, yang menunjukkan bahwa AUTOVACUUM telah menghapus data atau tuple usang dari basis data PostgreSQL.

Untuk menggunakan `pgstattuple` dalam mengumpulkan informasi tentang tabel, tentukan nama tabel atau pengidentifikasi objek (OID) dalam kueri:

```
postgres=> SELECT * FROM pgstattuple('lab');
```

```
table_len  | tuple_count | tuple_len | tuple_percent | dead_tuple_count |
 dead_tuple_len | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056    | 100001      | 2800028   | 77.16         | 0                | 0
  | 0                  | 16616      | 0.46
(1 row)
```

Dalam kueri berikut, kita menonaktifkan AUTOVACUUM dan memasukkan perintah yang menghapus 25.000 baris dari tabel. Akibatnya, `dead_tuple_count` meningkat menjadi 25000.

```
postgres=> DELETE FROM lab WHERE generate_series < 25000;

DELETE 25000
```

```
SELECT * FROM pgstattuple('lab');
```

```
table_len | tuple_count | tuple_len | tuple_percent | dead_tuple_count | dead_tuple_len
 | dead_tuple_percent | free_space | free_percent
-----------+-------------+-----------+---------------+------------------
+----------------+--------------------+------------+--------------
3629056 | 75001 | 2100028 | 57.87 | 25000 | 700000 | 19.29 | 16616 | 0.46
(1 row)
```

Untuk mengklaim kembali tuple nonaktif, mulailah proses VACUUM.

Mengamati bloat tanpa menghentikan aplikasi Anda

Pengaturan pada klaster Aurora dioptimalkan untuk memberikan praktik terbaik untuk sebagian besar beban kerja. Namun, Anda sebaiknya mengoptimalkan klaster agar lebih sesuai dengan aplikasi dan pola penggunaan Anda. Dalam hal ini, Anda dapat menggunakan ekstensi `pgstattuple` tanpa menghentikan aplikasi yang sibuk. Untuk melakukannya, lakukan langkah-langkah berikut ini:

1. Kloning instans Aurora Anda.
2. Ubah file parameter untuk menonaktifkan AUTOVACUUM di klon.
3. Lakukan kueri `pgstattuple` saat menguji klon dengan beban kerja sampel atau dengan pgbench, yang merupakan program untuk menjalankan pengujian tolok ukur di PostgreSQL. Untuk informasi selengkapnya, lihat pgbench.

Setelah menjalankan aplikasi Anda dan melihat hasilnya, gunakan pg_repack atau VACUUM FULL pada salinan yang dipulihkan dan bandingkan perbedaannya. Jika Anda melihat penurunan yang signifikan pada dead_tuple_count, dead_tuple_len, atau dead_tuple_percent, sesuaikan jadwal vacuum pada klaster produksi Anda untuk meminimalkan bloat.

Menghindari bloat di tabel sementara

Jika aplikasi Anda membuat tabel sementara, pastikan bahwa aplikasi Anda menghapus tabel sementara ketika sudah tidak lagi diperlukan. Proses autovacuum tidak menemukan tabel sementara. Jika tidak terpantau, tabel sementara dapat dengan cepat membuat bloat basis data. Selain itu, bloat dapat meluas ke tabel sistem, yang merupakan tabel internal yang melacak objek dan atribut PostgreSQL, seperti pg_attribute dan pg_depend.

Ketika tabel sementara tidak lagi diperlukan, Anda dapat menggunakan pernyataan TRUNCATE untuk mengosongkan tabel dan membebaskan ruang. Kemudian, lakukan vacuum manual pada

tabel pg_attribute dan pg_depend. Dengan melakukan vacuum pada tabel ini, akan dipastikan bahwa pembuatan dan pemotongan/penghapusan tabel sementara secara terus-menerus tidak akan menambahkan tuple dan berkontribusi pada bloat sistem.

Anda dapat menghindari masalah ini saat membuat tabel sementara dengan menyertakan sintaksis berikut yang menghapus baris baru saat konten dikomit:

```
CREATE TEMP TABLE IF NOT EXISTS table_name(table_description) ON COMMIT DELETE ROWS;
```

Klausa `ON COMMIT DELETE ROWS` memotong tabel sementara ketika transaksi dikomit.

Menghindari bloat dalam indeks

Saat Anda mengubah bidang yang diindeks dalam tabel, pembaruan indeks akan menghasilkan satu atau beberapa tuple nonaktif dalam indeks tersebut. Secara default, proses autovacuum membersihkan bloat dalam indeks, tetapi pembersihan tersebut menggunakan sejumlah besar waktu dan sumber daya. Untuk menentukan preferensi pembersihan indeks saat Anda membuat tabel, sertakan klausa vacuum_index_cleanup. Secara default, pada waktu pembuatan tabel, klausa ini diatur ke AUTO, yang berarti bahwa server akan memutuskan apakah indeks Anda memerlukan pembersihan saat melakukan vaccum pada tabel. Anda dapat mengatur klausa ini ke ON untuk mengaktifkan pembersihan indeks untuk tabel tertentu, atau OFF untuk menonaktifkan pembersihan indeks untuk tabel tersebut. Ingat, menonaktifkan pembersihan indeks mungkin menghemat waktu, tetapi berpotensi menyebabkan indeks mengalami bloat.

Anda dapat mengontrol pembersihan indeks secara manual saat Anda melakukan VACUUM pada tabel di baris perintah. Untuk melakukan vacuum pada tabel dan menghapus tuple nonaktif dari indeks, sertakan klausa INDEX_CLEANUP dengan nilai ON dan nama tabel:

```
acctg=> VACUUM (INDEX_CLEANUP ON) receivables;

INFO: aggressively vacuuming "public.receivables"
VACUUM
```

Untuk melakukan vacuum pada tabel tanpa membersihkan indeks, tentukan nilai OFF:

```
acctg=> VACUUM (INDEX_CLEANUP OFF) receivables;

INFO: aggressively vacuuming "public.receivables"
VACUUM
```

Manajemen memori yang ditingkatkan dalam Aurora PostgreSQL

Beban kerja pelanggan yang menghabiskan memori bebas yang tersedia di instans DB menyebabkan pengaktifan ulang basis data oleh sistem operasi sehingga menyebabkan tidak tersedianya basis data. Aurora PostgreSQL telah memperkenalkan peningkatan kemampuan manajemen memori yang secara proaktif mencegah masalah stabilitas dan pengaktifan ulang basis data yang disebabkan oleh memori bebas yang tidak mencukupi. Peningkatan ini tersedia secara default dalam versi berikut:

- 15.3 dan versi 15 yang lebih tinggi
- 14.8 dan versi 14 yang lebih tinggi
- Versi 13.11 dan versi 13 yang lebih tinggi
- Versi 12.15 dan versi 12 yang lebih tinggi
- 11.20 dan versi 11 yang lebih tinggi

Untuk meningkatkan manajemen memori, layanan tersebut melakukan hal berikut:

- Membatalkan transaksi basis data yang meminta lebih banyak memori ketika sistem mendekati tekanan memori kritis.
- Sistem dikatakan berada di bawah tekanan memori kritis ketika menghabiskan semua memori fisik dan akan menghabiskan swap. Dalam keadaan ini, setiap transaksi yang meminta memori akan dibatalkan agar dapat segera mengurangi tekanan memori dalam instans DB.
- Peluncur PostgreSQL penting dan pekerja latar belakang seperti pekerja autovacuum selalu dilindungi.

Mengkonfigurasi parameter manajemen memori

Untuk mengaktifkan manajemen memori

Fitur ini diaktifkan secara default. Pesan kesalahan ditampilkan ketika transaksi dibatalkan karena memori tidak mencukupi seperti yang ditunjukkan pada contoh berikut:

```
ERROR: out of memory Detail: Failed on request of size 16777216.
```

Untuk mematikan manajemen memori

Untuk mematikan fitur ini, sambungkan ke cluster Aurora PostgreSQL DB dengan psql dan gunakan pernyataan SET untuk nilai parameter seperti yang disebutkan di bawah ini.

Untuk Aurora PostgreSQL versi 11.21, 12.16, 13.12, 14.9, 15.4, dan versi yang lebih lama:

```
postgres=>SET rds.memory_allocation_guard = true;
```

Nilai default `rds.memory_allocation_guard` parameter diatur ke `false` dalam kelompok Parameter.

Untuk Aurora PostgreSQL 12.17, 13.13, 14.10, 15.5, dan versi yang lebih tinggi:

```
postgres=>rds.enable_memory_management = false;
```

Nilai default `rds.enable_memory_management` parameter diatur ke `true` dalam kelompok Parameter.

Menyetel nilai parameter ini dalam grup parameter cluster DB mencegah kueri dibatalkan. Untuk informasi selengkapnya tentang grup parameter cluster DB, lihatBekerja dengan grup parameter.

Nilai parameter dinamis ini juga dapat diatur pada tingkat sesi untuk memasukkan atau mengecualikan sesi dalam manajemen memori yang ditingkatkan.

> **Note**
>
> Kami tidak menyarankan untuk mematikan fitur ini karena dapat menyebabkan out-of-memory kesalahan yang dapat menyebabkan restart basis data yang diinduksi beban kerja karena kelelahan memori di sistem.

Failover cepat dengan Amazon Aurora PostgreSQL

Di bagian berikut ini, Anda dapat mempelajari cara memastikan failover terjadi secepat mungkin. Untuk memulihkan dengan cepat setelah failover, Anda dapat menggunakan manajemen cache klaster untuk klaster DB Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Pemulihan cepat setelah failover dengan manajemen cache klaster untuk Aurora PostgreSQL.

Beberapa langkah yang dapat Anda ambil untuk membuat failover beperforma cepat antara lain sebagai berikut:

- Atur keepalive Protokol Kontrol Transmisi (TCP) dengan rentang waktu singkat, untuk menghentikan kueri yang berjalan lebih lama sebelum batas waktu baca berakhir jika ada kegagalan.
- Atur batas waktu untuk caching Java Sistem Nama Domain (DNS) secara agresif. Tindakan ini akan membantu memastikan titik akhir hanya baca Aurora dapat disikluskan dengan benar melalui simpul hanya baca pada percobaan koneksi selanjutnya.
- Atur variabel batas waktu yang digunakan dalam string koneksi JDBC serendah mungkin. Gunakan objek koneksi terpisah untuk kueri berjalan singkat dan berjalan lama.
- Gunakan titik akhir baca dan tulis Aurora yang diberikan untuk terhubung ke klaster.
- Gunakan operasi API RDS untuk menguji respons aplikasi saat terjadi kegagalan sisi server. Selain itu, gunakan alat penghapusan paket untuk menguji respons aplikasi terhadap kegagalan sisi klien.
- Gunakan AWS JDBC Driver for PostgreSQL untuk sepenuhnya memanfaatkan kemampuan failover Aurora PostgreSQL. Untuk informasi selengkapnya tentang AWS JDBC Driver for PostgreSQL dan petunjuk lengkap untuk menggunakannya, lihat Repositori GitHub AWS JDBC Driver for PostgreSQL.

Hal ini dibahas secara lebih terperinci sebagai berikut.

Topik

- Mengatur parameter keepalive TCP
- Mengonfigurasi aplikasi Anda untuk failover cepat
- Menguji failover
- Contoh failover cepat di Java

Mengatur parameter keepalive TCP

Saat Anda mengatur koneksi TCP, serangkaian pengatur waktu dikaitkan dengan koneksi. Saat pengatur waktu keepalive mencapai nol, paket probe keepalive dikirim ke titik akhir koneksi. Jika probe menerima balasan, Anda dapat mengasumsikan bahwa koneksi masih aktif dan berjalan.

Dengan mengaktifkan parameter keepalive TCP dan mengaturnya secara agresif, akan dipastikan bahwa jika klien Anda tidak dapat terhubung ke basis data, koneksi aktif apa pun akan ditutup dengan cepat. Aplikasi kemudian dapat terhubung ke titik akhir baru.

Pastikan untuk mengatur parameter keepalive TCP berikut:

- `tcp_keepalive_time` mengontrol waktu, dalam hitungan detik, setelah paket keepalive dikirim ketika tidak ada data yang dikirim oleh soket. ACK tidak dianggap sebagai data. Kami menyarankan pengaturan berikut:

  ```
  tcp_keepalive_time = 1
  ```

- `tcp_keepalive_intvl` mengontrol waktu, dalam hitungan detik, di antara waktu pengiriman paket keepalive berikutnya setelah paket awal dikirim. Atur waktu ini dengan menggunakan parameter `tcp_keepalive_time`. Kami menyarankan pengaturan berikut:

  ```
  tcp_keepalive_intvl = 1
  ```

- `tcp_keepalive_probes` adalah jumlah probe keepalive yang tidak diakui yang terjadi sebelum aplikasi diberi tahu. Kami menyarankan pengaturan berikut:

  ```
  tcp_keepalive_probes = 5
  ```

Pengaturan ini harus memberi tahu aplikasi dalam waktu lima detik ketika basis data berhenti merespons. Jika paket keepalive sering dihapus di dalam jaringan aplikasi, Anda dapat mengatur nilai `tcp_keepalive_probes` yang lebih tinggi. Tindakan ini memungkinkan lebih banyak buffer di jaringan yang kurang andal, meskipun akan menambah waktu yang dibutuhkan untuk mendeteksi kegagalan sebenarnya.

Untuk mengatur parameter keepalive TCP di Linux

1. Uji konfigurasi parameter TCP keepalive Anda.

   Untuk melakukannya, kami merekomendasikan penggunaan baris perintah dengan perintah berikut. Konfigurasi yang disarankan ini ditujukan untuk seluruh sistem. Dengan kata lain, konfigurasi ini juga memengaruhi semua aplikasi lain yang membuat soket dengan opsi `SO_KEEPALIVE` aktif.

   ```
   sudo sysctl net.ipv4.tcp_keepalive_time=1
   sudo sysctl net.ipv4.tcp_keepalive_intvl=1
   sudo sysctl net.ipv4.tcp_keepalive_probes=5
   ```

2. Setelah Anda menemukan konfigurasi yang cocok untuk aplikasi Anda, pertahankan pengaturan ini dengan menambahkan baris berikut ke `/etc/sysctl.conf`, termasuk setiap perubahan yang Anda buat:

   ```
   tcp_keepalive_time = 1
   ```

```
tcp_keepalive_intvl = 1
tcp_keepalive_probes = 5
```

Mengonfigurasi aplikasi Anda untuk failover cepat

Di bagian berikut ini, Anda dapat menemukan diskusi tentang beberapa perubahan konfigurasi untuk Aurora PostgreSQL yang dapat Anda buat untuk failover cepat. Untuk mempelajari selengkapnya tentang penyiapan dan konfigurasi driver JDBC PostgreSQL, lihat dokumentasi Driver JDBC PostgreSQL.

Topik

- Mengurangi batas waktu cache DNS
- Mengatur string koneksi Aurora PostgreSQL untuk failover cepat
- Opsi lain untuk mendapatkan string host

Mengurangi batas waktu cache DNS

Saat aplikasi Anda mencoba membuat koneksi setelah failover, penulis Aurora PostgreSQL baru akan menjadi pembaca sebelumnya. Anda dapat menemukannya dengan menggunakan titik akhir hanya baca Aurora sebelum pembaruan DNS disebarkan sepenuhnya. Mengatur time to live (TTL) DNS java ke nilai rendah, seperti kurang dari 30 detik, akan membantu siklus di antara simpul pembaca pada percobaan koneksi selanjutnya.

```
// Sets internal TTL to match the Aurora RO Endpoint TTL
java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
// If the lookup fails, default to something like small to retry
java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
```

Mengatur string koneksi Aurora PostgreSQL untuk failover cepat

Untuk menggunakan failover cepat Aurora PostgreSQL, pastikan bahwa string koneksi aplikasi Anda memiliki daftar host, bukan hanya satu host. Berikut ini adalah contoh string koneksi yang dapat Anda gunakan untuk terhubung ke klaster Aurora PostgreSQL. Dalam contoh ini, host dicetak tebal.

```
jdbc:postgresql://myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

```
/postgres?user=<primaryuser>&password=<primarypw>&loginTimeout=2
&connectTimeout=2&cancelSignalTimeout=2&socketTimeout=60
&tcpKeepAlive=true&targetServerType=primary
```

Untuk ketersediaan terbaik dan untuk menghindari ketergantungan pada API RDS, kami sarankan Anda mempertahankan file yang digunakan untuk terhubung. File ini berisi string host yang dibaca aplikasi sejak Anda membuat koneksi ke basis data. String host ini memiliki semua titik akhir Aurora yang tersedia untuk klaster. Untuk informasi selengkapnya tentang titik akhir Aurora, lihat Manajemen koneksi Amazon Aurora.

Misalnya, Anda dapat menyimpan titik akhir Anda dalam file lokal seperti yang ditunjukkan berikut ini.

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

Aplikasi Anda akan membaca dari file ini untuk mengisi bagian host dalam string koneksi JDBC. Mengganti nama klaster DB akan menyebabkan perubahan pada titik akhir ini. Pastikan bahwa aplikasi Anda menangani peristiwa ini jika terjadi.

Opsi lainnya adalah menggunakan daftar simpul instans DB sebagai berikut.

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node3.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node4.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

Kelebihan dari pendekatan ini adalah driver koneksi JDBC PostgreSQL akan melewati semua simpul pada daftar ini untuk menemukan koneksi yang valid. Sebaliknya, ketika Anda menggunakan titik akhir Aurora, hanya dua simpul yang akan dicoba pada setiap percobaan koneksi. Namun, ada kekurangan pada penggunaan simpul instans DB. Jika Anda menambahkan atau menghapus simpul dari klaster dan daftar instans titik akhir menjadi kedaluwarsa, driver koneksi mungkin tidak akan menemukan host yang tepat untuk terhubung.

Untuk membantu memastikan bahwa aplikasi Anda tidak menunggu terlalu lama untuk terhubung ke satu host, atur parameter berikut secara agresif:

- `targetServerType` – Mengontrol apakah driver terhubung ke simpul tulis atau simpul baca. Untuk memastikan bahwa aplikasi Anda hanya terhubung kembali ke simpul tulis, atur nilai `targetServerType` ke `primary`.

Nilai untuk parameter `targetServerType` mencakup `primary`, `secondary`, `any`, dan `preferSecondary`. Nilai `preferSecondary` akan mencoba membuat koneksi ke pembaca terlebih dahulu. Koneksi dibuat ke penulis jika tidak ada koneksi pembaca yang dapat dibuat.

- `loginTimeout` – Mengontrol waktu tunggu aplikasi untuk login ke basis data setelah koneksi soket dibuat.
- `connectTimeout` – Mengontrol waktu tunggu soket untuk membuat koneksi ke basis data.

Anda dapat mengubah parameter aplikasi lain untuk mempercepat proses koneksi, tergantung pada seberapa agresif perilaku aplikasi yang Anda inginkan.

- `cancelSignalTimeout` – Di beberapa aplikasi, Anda mungkin ingin mengirim sinyal pembatalan “percobaan terbaik” pada kueri yang telah habis batas waktunya. Jika sinyal pembatalan ini berada di jalur failover Anda, pertimbangkan untuk mengaturnya secara agresif agar sinyal ini tidak dikirim ke host yang nonaktif.
- `socketTimeout` – Parameter ini mengontrol waktu tunggu soket untuk operasi baca. Parameter ini dapat digunakan sebagai "batas waktu kueri" global untuk memastikan tidak ada kueri yang menunggu lebih lama dari nilai ini. Praktik yang baik adalah memiliki dua handler koneksi. Satu handler koneksi menjalankan kueri yang bermasa aktif singkat dan menetapkan nilai ini lebih rendah. Handler koneksi lain, untuk kueri yang berjalan lama, mengatur nilai ini jauh lebih tinggi. Dengan pendekatan ini, Anda dapat mengandalkan parameter keepalive TCP untuk menghentikan kueri yang berjalan lama jika server mengalami gangguan.
- `tcpKeepAlive` – Aktifkan parameter ini untuk memastikan parameter keepalive TCP yang Anda tetapkan dipatuhi.
- `loadBalanceHosts` – Saat ditetapkan ke `true`, parameter ini akan membuat aplikasi terhubung ke host acak yang dipilih dari daftar host kandidat.

Opsi lain untuk mendapatkan string host

Anda bisa mendapatkan string host dari beberapa sumber, termasuk fungsi `aurora_replica_status` dan dengan menggunakan API Amazon RDS.

Dalam banyak kasus, Anda perlu menentukan penulis klaster atau untuk menemukan simpul pembaca lainnya di klaster. Untuk melakukannya, aplikasi Anda dapat terhubung ke instans DB apa pun di klaster DB dan mengkueri fungsi `aurora_replica_status`. Anda dapat menggunakan fungsi ini untuk mengurangi waktu yang dibutuhkan untuk menemukan host untuk terhubung. Namun,

dalam skenario kegagalan jaringan tertentu, fungsi `aurora_replica_status` dapat menampilkan informasi yang usang atau tidak lengkap.

Cara yang tepat untuk memastikan bahwa aplikasi Anda dapat menemukan simpul untuk terhubung adalah dengan mencoba terhubung ke titik akhir penulis klaster lalu titik akhir pembaca klaster. Anda melakukannya sampai Anda dapat membuat koneksi yang dapat dibaca. Titik akhir ini tidak berubah kecuali jika Anda mengganti nama klaster DB Anda. Dengan demikian, Anda umumnya dapat membiarkannya sebagai anggota statis aplikasi Anda atau menyimpannya dalam file sumber daya yang dibaca oleh aplikasi Anda.

Setelah membuat koneksi menggunakan salah satu titik akhir ini, Anda dapat memperoleh informasi tentang klaster lainnya. Untuk melakukannya, panggil fungsi `aurora_replica_status`. Misalnya, perintah berikut mengambil informasi dengan `aurora_replica_status`.

```
postgres=> SELECT server_id, session_id, highest_lsn_rcvd, cur_replay_latency_in_usec,
 now(), last_update_timestamp
FROM aurora_replica_status();

server_id | session_id | highest_lsn_rcvd | cur_replay_latency_in_usec | now |
 last_update_timestamp
-----------+--------------------------------------+-----------------
+---------------------------+-------------------------------+------------------------
mynode-1 | 3e3c5044-02e2-11e7-b70d-95172646d6ca | 594221001 | 201421 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-2 | 1efdd188-02e4-11e7-becd-f12d7c88a28a | 594221001 | 201350 | 2017-03-07
 19:50:24.695322+00 | 2017-03-07 19:50:23+00
mynode-3 | MASTER_SESSION_ID | | | 2017-03-07 19:50:24.695322+00 | 2017-03-07
 19:50:23+00
(3 rows)
```

Misalnya, bagian host dalam string koneksi Anda mungkin dimulai dengan titik akhir klaster penulis dan pembaca, seperti yang ditunjukkan berikut.

```
myauroracluster.cluster-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-beta.rds.amazonaws.com:5432
```

Dalam skenario ini, aplikasi Anda mencoba membuat koneksi ke jenis simpul apa pun, primer atau sekunder. Saat aplikasi Anda terhubung, praktik yang baik adalah memeriksa terlebih dahulu status baca/tulis simpul tersebut. Untuk melakukannya, jalankan kueri untuk hasil perintah `SHOW transaction_read_only`.

Jika nilai yang dihasilkan dari kueri adalah `OFF`, berarti Anda berhasil terhubung ke simpul primer. Namun, misalkan nilai yang dihasilkan adalah `ON` dan aplikasi Anda memerlukan koneksi baca/tulis. Dalam hal ini, Anda dapat memanggil fungsi `aurora_replica_status` untuk menentukan `server_id` yang memiliki `session_id='MASTER_SESSION_ID'`. Fungsi ini akan memberikan nama simpul primer. Anda dapat menggunakannya dengan `endpointPostfix` yang dijelaskan berikut.

Pastikan Anda mengetahui saat Anda terhubung ke replika yang memiliki data yang kedaluwarsa (stale). Saat ini terjadi, fungsi `aurora_replica_status` mungkin menampilkan informasi yang sudah tidak berlaku. Anda dapat menetapkan ambang batas untuk staleness di tingkat aplikasi. Untuk memeriksanya, Anda dapat melihat perbedaan antara waktu server dan nilai `last_update_timestamp`. Secara umum, aplikasi Anda harus menghindari peralihan di antara dua host karena informasi yang bertentangan yang dihasilkan oleh fungsi `aurora_replica_status`. Aplikasi Anda harus mencoba semua host yang diketahui terlebih dahulu, bukan mengikuti data yang dihasilkan oleh `aurora_replica_status`.

Menampilkan daftar instans menggunakan operasi API DescribeDBClusters beserta contohnya di Java

Anda dapat menemukan daftar instans secara programatis menggunakan AWS SDK for Java, khususnya operasi API DescribeDBClusters.

Berikut ini adalah contoh sederhana tentang cara melakukannya di Java 8.

```
AmazonRDS client = AmazonRDSClientBuilder.defaultClient();
DescribeDBClustersRequest request = new DescribeDBClustersRequest()
   .withDBClusterIdentifier(clusterName);
DescribeDBClustersResult result =
rdsClient.describeDBClusters(request);

DBCluster singleClusterResult = result.getDBClusters().get(0);

String pgJDBCEndpointStr =
singleClusterResult.getDBClusterMembers().stream()
   .sorted(Comparator.comparing(DBClusterMember::getIsClusterWriter)
   .reversed()) // This puts the writer at the front of the list
   .map(m -> m.getDBInstanceIdentifier() + endpointPostfix + ":" +
 singleClusterResult.getPort()))
   .collect(Collectors.joining(","));
```

Di sini, `pgJDBCEndpointStr` berisi daftar titik akhir berformat, seperti yang ditunjukkan berikut ini.

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

Variabel `endpointPostfix` dapat berupa konstanta yang ditetapkan aplikasi Anda. Atau, aplikasi Anda bisa mendapatkannya dengan mengkueri operasi API `DescribeDBInstances` untuk satu instans di klaster Anda. Nilai ini tetap konstan dalam Wilayah AWS dan untuk pelanggan individu. Jadi, variabel tersebut menyimpan panggilan API hanya untuk mempertahankan konstanta ini dalam file sumber daya yang dibaca aplikasi Anda. Dalam contoh sebelumnya, variabel tersebut diatur sebagai berikut.

```
.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com
```

Untuk tujuan ketersediaan, praktik yang baik adalah menggunakan titik akhir Aurora pada klaster DB Anda secara default jika API ini tidak merespons atau memerlukan waktu terlalu lama untuk merespons. Titik akhir dijamin akan diperbarui selama waktu yang diperlukan untuk memperbarui catatan DNS. Pembaruan catatan DNS dengan titik akhir biasanya membutuhkan waktu kurang dari 30 detik. Anda dapat menyimpan titik akhir dalam file sumber daya yang digunakan aplikasi Anda.

Menguji failover

Dalam semua kasus, Anda harus memiliki klaster DB yang berisi dua atau beberapa instans DB.

Dari sisi server, operasi API tertentu dapat menyebabkan pemadaman yang dapat digunakan untuk menguji respons aplikasi Anda:

- FailoverDBCluster – Operasi ini mencoba mempromosikan instans DB baru di klaster DB Anda menjadi penulis.

  Contoh kode berikut menunjukkan cara Anda dapat menggunakan `failoverDBCluster` untuk menyebabkan pemadaman. Untuk detail selengkapnya tentang menyiapkan klien Amazon RDS, lihat Menggunakan AWS SDK for Java.

  ```
  public void causeFailover() {

      final AmazonRDS rdsClient = AmazonRDSClientBuilder.defaultClient();

      FailoverDBClusterRequest request = new FailoverDBClusterRequest();
      request.setDBClusterIdentifier("cluster-identifier");

      rdsClient.failoverDBCluster(request);
  ```

```
}
```

- RebootDBInstance – Failover tidak dijamin dengan operasi API ini. Namun, operasi ini menonaktifkan basis data pada penulis. Anda dapat menggunakannya untuk menguji respons aplikasi Anda terhadap koneksi yang terputus. Parameter `ForceFailover` tidak berlaku untuk mesin Aurora. Sebagai gantinya, gunakan operasi API `FailoverDBCluster`.
- ModifyDBCluster – Mengubah parameter `Port` akan menyebabkan pemadaman saat simpul dalam klaster mulai mendengarkan di port baru. Secara umum, aplikasi Anda dapat merespons kegagalan ini terlebih dahulu dengan memastikan bahwa hanya aplikasi Anda yang mengontrol perubahan port. Selan itu, pastikan bahwa aplikasi Anda dapat dengan tepat memperbarui titik akhir yang diandalkannya. Anda dapat melakukannya dengan meminta seseorang memperbarui port secara manual saat melakukan perubahan di tingkat API. Atau, Anda dapat melakukannya dengan menggunakan API RDS di aplikasi Anda untuk menentukan apakah port telah berubah.
- ModifyDBInstance – Mengubah parameter `DBInstanceClass` akan menyebabkan pemadaman.
- DeleteDBInstance – Menghapus instans DB primer (penulis) akan menyebabkan instans DB baru dipromosikan menjadi penulis di klaster DB Anda.

Dari sisi aplikasi atau klien, jika Anda menggunakan Linux, Anda dapat menguji respons aplikasi terhadap penghapusan paket yang tiba-tiba. Anda dapat melakukannya berdasarkan apakah port atau host telah berubah, atau apakah paket keepalive TCP dikirim atau diterima dengan menggunakan perintah iptables.

Contoh failover cepat di Java

Contoh kode berikut menunjukkan cara aplikasi dapat mengatur pengelola driver Aurora PostgreSQL.

Aplikasi memanggil fungsi `getConnection` saat memerlukan koneksi. Panggilan ke `getConnection` dapat gagal menemukan host yang valid. Contohnya adalah ketika tidak ada penulis yang ditemukan, tetapi parameter `targetServerType` diatur ke `primary`. Dalam hal ini, aplikasi yang memanggil harus mencoba memanggil lagi fungsi tersebut.

Untuk menghindari mendorong perilaku coba lagi ke aplikasi, Anda dapat menggabungkan panggilan coba lagi ini ke pooler koneksi. Dengan sebagian besar pooler koneksi, Anda dapat menentukan string koneksi JDBC. Jadi, aplikasi Anda dapat memanggil `getJdbcConnectionString` dan meneruskannya ke pooler koneksi. Tindakan ini memungkinkan Anda menggunakan failover lebih cepat dengan Aurora PostgreSQL.

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

import org.joda.time.Duration;

public class FastFailoverDriverManager {
   private static Duration LOGIN_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CONNECT_TIMEOUT = Duration.standardSeconds(2);
   private static Duration CANCEL_SIGNAL_TIMEOUT = Duration.standardSeconds(1);
   private static Duration DEFAULT_SOCKET_TIMEOUT = Duration.standardSeconds(5);

   public FastFailoverDriverManager() {
       try {
            Class.forName("org.postgresql.Driver");
       } catch (ClassNotFoundException e) {
            e.printStackTrace();
       }

       /*
         * RO endpoint has a TTL of 1s, we should honor that here. Setting this
 aggressively makes sure that when
         * the PG JDBC driver creates a new connection, it will resolve a new different
 RO endpoint on subsequent attempts
         * (assuming there is > 1 read node in your cluster)
         */
        java.security.Security.setProperty("networkaddress.cache.ttl" , "1");
       // If the lookup fails, default to something like small to retry
       java.security.Security.setProperty("networkaddress.cache.negative.ttl" , "3");
   }

   public Connection getConnection(String targetServerType) throws SQLException {
       return getConnection(targetServerType, DEFAULT_SOCKET_TIMEOUT);
   }

   public Connection getConnection(String targetServerType, Duration queryTimeout)
 throws SQLException {
        Connection conn =
 DriverManager.getConnection(getJdbcConnectionString(targetServerType, queryTimeout));
```

```
      /*
        * A good practice is to set socket and statement timeout to be the same thing
since both
        * the client AND server will stop the query at the same time, leaving no
running queries
        * on the backend
        */
       Statement st = conn.createStatement();
       st.execute("set statement_timeout to " + queryTimeout.getMillis());
       st.close();

      return conn;
  }

  private static String urlFormat = "jdbc:postgresql://%s"
          + "/postgres"
          + "?user=%s"
          + "&password=%s"
          + "&loginTimeout=%d"
          + "&connectTimeout=%d"
          + "&cancelSignalTimeout=%d"
          + "&socketTimeout=%d"
          + "&targetServerType=%s"
          + "&tcpKeepAlive=true"
          + "&ssl=true"
          + "&loadBalanceHosts=true";
  public String getJdbcConnectionString(String targetServerType, Duration
queryTimeout) {
      return String.format(urlFormat,
                getFormattedEndpointList(getLocalEndpointList()),
                CredentialManager.getUsername(),
                CredentialManager.getPassword(),
                LOGIN_TIMEOUT.getStandardSeconds(),
                CONNECT_TIMEOUT.getStandardSeconds(),
                CANCEL_SIGNAL_TIMEOUT.getStandardSeconds(),
                queryTimeout.getStandardSeconds(),
                targetServerType
      );
  }

  private List<String> getLocalEndpointList() {
      /*
        * As mentioned in the best practices doc, a good idea is to read a local
resource file and parse the cluster endpoints.
```

```
         * For illustration purposes, the endpoint list is hardcoded here
         */
        List<String> newEndpointList = new ArrayList<>();
        newEndpointList.add("myauroracluster.cluster-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");
        newEndpointList.add("myauroracluster.cluster-ro-c9bfei4hjlrd.us-east-1-
beta.rds.amazonaws.com:5432");

       return newEndpointList;
   }

   private static String getFormattedEndpointList(List<String> endpoints) {
       return IntStream.range(0, endpoints.size())
               .mapToObj(i -> endpoints.get(i).toString())
               .collect(Collectors.joining(","));
   }
}
```

Pemulihan cepat setelah failover dengan manajemen cache klaster untuk Aurora PostgreSQL

Untuk pemulihan cepat instans DB penulis di klaster Aurora PostgreSQL Anda jika terjadi failover, gunakan manajemen cache klaster untuk Amazon Aurora PostgreSQL. Manajemen cache klaster memastikan agar performa aplikasi dipertahankan jika terjadi failover.

Dalam situasi failover yang umum, Anda mungkin mendapati penurunan performa yang besar tetapi sementara setelah failover. Penurunan performa ini terjadi karena ketika instans DB failover dimulai, cache buffer kosong. Cache kosong juga dikenal sebagai cold cache. Cold cache menurunkan performa karena instans DB harus membaca dari disk yang lebih lambat, bukan memanfaatkan nilai yang tersimpan dalam cache buffer.

Dengan manajemen cache klaster, Anda menetapkan instans DB pembaca tertentu sebagai target failover. Manajemen cache klaster memastikan bahwa data dalam cache pembaca yang ditetapkan masih tersinkronisasi dengan data dalam cache instans DB penulis. Cache pembaca yang ditetapkan dengan nilai yang sudah diisi dikenal sebagai warm cache. Jika terjadi failover, pembaca yang ditetapkan akan langsung menggunakan nilai yang ada di warm cache-nya saat dipromosikan ke instans DB penulis baru. Pendekatan ini memberikan performa pemulihan yang lebih baik pada aplikasi Anda.

Manajemen cache klaster mengharuskan instans pembaca yang ditetapkan memiliki jenis kelas dan ukuran instans yang sama (contohnya, `db.r5.2xlarge` atau `db.r5.xlarge`) seperti instans

penulis. Ingatlah hal ini ketika Anda membuat klaster DB Aurora PostgreSQL, sehingga klaster Anda dapat dipulihkan selama failover. Untuk daftar jenis kelas dan ukuran instans, lihat Spesifikasi perangkat keras kelas instans DB untuk Aurora.

> **Note**
>
> Manajemen cache klaster tidak didukung untuk klaster DB Aurora PostgreSQL yang merupakan bagian dari basis data global Aurora. Sebaiknya beban kerja tidak berjalan pada pembaca tingkat-0 yang ditetapkan.

Daftar Isi

- Mengonfigurasi manajemen cache klaster
  - Mengaktifkan manajemen cache klaster
  - Mengatur prioritas tingkat promosi untuk instans DB penulis
  - Mengatur prioritas tingkat promosi untuk instans DB pembaca
- Memantau cache buffer
- Memecahkan masalah konfigurasi CCM

Mengonfigurasi manajemen cache klaster

Untuk mengonfigurasi manajemen cache klaster, lakukan proses berikut ini secara berurutan.

Topik

- Mengaktifkan manajemen cache klaster
- Mengatur prioritas tingkat promosi untuk instans DB penulis
- Mengatur prioritas tingkat promosi untuk instans DB pembaca

> **Note**
>
> Berikan waktu setidaknya 1 menit setelah menyelesaikan langkah ini agar manajemen cache klaster dapat beroperasi sepenuhnya.

Mengaktifkan manajemen cache klaster

Untuk mengaktifkan manajemen cache klaster, lakukan langkah-langkah yang dijelaskan sebagai berikut.

Konsol

Untuk mengaktifkan manajemen cache klaster

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter untuk klaster DB Aurora PostgreSQL Anda.

   Klaster DB harus menggunakan grup parameter selain default karena Anda tidak dapat mengubah nilai di grup parameter default.
4. Untuk Tindakan grup parameter, pilih Edit.
5. Tetapkan nilai parameter klaster `apg_ccm_enabled` ke 1.
6. Pilih Simpan perubahan.

AWS CLI

Guna mengaktifkan manajemen cache klaster untuk klaster DB Aurora PostgreSQL, gunakan perintah AWS CLI `modify-db-cluster-parameter-group` dengan parameter yang diperlukan berikut ini:

- `--db-cluster-parameter-group-name`
- `--parameters`

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my-db-cluster-parameter-group \
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my-db-cluster-parameter-group ^
    --parameters "ParameterName=apg_ccm_enabled,ParameterValue=1,ApplyMethod=immediate"
```

Mengatur prioritas tingkat promosi untuk instans DB penulis

Untuk manajemen cache klaster, pastikan bahwa prioritas promosi adalah tingkat-0 untuk instans DB penulis pada klaster DB Aurora PostgreSQL. Prioritas tingkat promosi adalah nilai yang menentukan urutan promosi pembaca Aurora ke instans DB penulis setelah kegagalan. Nilai yang valid adalah 0–15, dengan 0 adalah prioritas pertama dan 15 adalah prioritas terakhir. Untuk informasi selengkapnya tentang tingkat promosi, lihat Toleransi kesalahan untuk klaster DB Aurora.

Konsol

Untuk mengatur prioritas promosi instans DB penulis ke tingkat-0

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih instans DB Penulis pada klaster DB Aurora PostgreSQL.
4. Pilih Ubah. Halaman Modifikasi Instans DB akan muncul.
5. Di panel Konfigurasi tambahan, pilih tingkat-0 untuk Prioritas failover.
6. Pilih Lanjutkan dan periksa ringkasan modifikasi.
7. Untuk segera menerapkan perubahan setelah Anda menyimpannya, pilih Terapkan segera.
8. Pilih Modifikasi Instans DB untuk menyimpan perubahan Anda.

AWS CLI

Untuk menetapkan prioritas tingkat promosi ke 0 untuk instans DB penulis menggunakanAWS CLI, panggil modify-db-instanceperintah dengan parameter wajib berikut:

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier writer-db-instance \
    --promotion-tier 0 \
    --apply-immediately
```

Untuk Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier writer-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

Mengatur prioritas tingkat promosi untuk instans DB pembaca

Anda harus menetapkan hanya satu instans DB pembaca untuk manajemen cache cluster. Untuk melakukannya, pilih pembaca dari klaster Aurora PostgreSQL yang memiliki kelas dan ukuran instans yang sama dengan instans DB penulis. Misalnya, jika penulis menggunakan `db.r5.xlarge`, pilih pembaca yang menggunakan jenis kelas dan ukuran instans yang sama ini. Kemudian, atur prioritas tingkat promosinya ke 0.

Prioritas tingkat promosi adalah nilai yang menentukan urutan promosi pembaca Aurora ke instans DB penulis setelah kegagalan. Nilai yang valid adalah 0–15, dengan 0 adalah prioritas pertama dan 15 adalah prioritas terakhir.

Konsol

Untuk menetapkan prioritas promosi instans DB pembaca ke tingkat-0

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih instans DB Pembaca pada klaster DB Aurora PostgreSQL yang memiliki kelas instans yang sama dengan instans DB penulis.
4. Pilih Ubah. Halaman Modifikasi Instans DB akan muncul.

5. Di panel Konfigurasi tambahan, pilih tingkat-0 untuk Prioritas failover.
6. Pilih Lanjutkan dan periksa ringkasan modifikasi.
7. Untuk segera menerapkan perubahan setelah Anda menyimpannya, pilih Terapkan segera.
8. Pilih Modifikasi Instans DB untuk menyimpan perubahan Anda.

AWS CLI

Untuk menetapkan prioritas tingkat promosi ke 0 untuk instans DB pembaca menggunakanAWS CLI, panggil modify-db-instanceperintah dengan parameter wajib berikut:

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier reader-db-instance \
    --promotion-tier 0 \
    --apply-immediately
```

Untuk Windows:

```
aws rds modify-db-instance ^
    --db-instance-identifier reader-db-instance ^
    ---promotion-tier 0  ^
    --apply-immediately
```

Memantau cache buffer

Setelah mengatur manajemen cache klaster, Anda dapat memantau status sinkronisasi antara cache buffer instans DB penulis dan warm cache buffer pembaca yang ditetapkan. Untuk memeriksa konten cache buffer di instans DB penulis dan instans DB pembaca yang ditetapkan, gunakan

modul pg_buffercache PostgreSQL. Untuk informasi selengkapnya, lihat Dokumentasi pg_buffercache PostgreSQL.

Menggunakan Fungsi **aurora_ccm_status**

Manajemen cache klaster juga menyediakan fungsi aurora_ccm_status. Gunakan fungsi aurora_ccm_status pada instans DB penulis untuk mendapatkan informasi berikut tentang progres cache warming pada pembaca yang ditetapkan:

- buffers_sent_last_minute – Jumlah buffer yang telah dikirim ke pembaca yang ditetapkan dalam satu menit terakhir.
- buffers_found_last_minute – Jumlah buffer yang sering diakses yang diidentifikasi selama beberapa menit terakhir.
- buffers_sent_last_scan – Jumlah buffer yang telah dikirim ke pembaca yang ditetapkan selama pemindaian lengkap cache buffer.
- buffers_found_last_scan – Jumlah buffer yang telah diidentifikasi sering diakses dan perlu dikirim selama pemindaian lengkap terakhir cache buffer. Buffer yang sudah di-cache pada pembaca yang ditetapkan tidak akan dikirimkan.
- buffers_sent_current_scan – Jumlah buffer yang telah dikirim sejauh ini selama pemindaian saat ini.
- buffers_found_current_scan – Jumlah buffer yang telah diidentifikasi sering diakses dalam pemindaian saat ini.
- current_scan_progress – Jumlah buffer yang telah diakses sejauh ini selama pemindaian saat ini.

Contoh berikut menunjukkan cara menggunakan fungsi aurora_ccm_status untuk mengonversi sebagian output-nya menjadi warm rate dan warm percentage.

```
SELECT buffers_sent_last_minute*8/60 AS warm_rate_kbps,
   100*(1.0-buffers_sent_last_scan::float/buffers_found_last_scan) AS warm_percent
   FROM aurora_ccm_status();
```

Memecahkan masalah konfigurasi CCM

Ketika Anda mengaktifkan parameter apg_ccm_enabled cluster, manajemen cache cluster secara otomatis diaktifkan pada tingkat instans pada instans DB penulis dan satu instance DB pembaca

pada cluster Aurora PostgreSQL DB. Instance writer dan reader harus menggunakan tipe dan ukuran kelas instance yang sama. Prioritas tingkat promosi mereka diatur ke`0`. Instans pembaca lain di cluster DB harus memiliki tingkat promosi bukan nol dan manajemen cache cluster dinonaktifkan untuk instance tersebut.

Alasan berikut dapat menyebabkan masalah dalam konfigurasi dan menonaktifkan manajemen cache cluster:

- Ketika tidak ada instans DB pembaca tunggal yang disetel ke tingkat promosi 0.
- Ketika instance DB penulis tidak disetel ke tingkat promosi 0.
- Ketika lebih dari satu instans DB pembaca diatur ke tingkat promosi 0.
- Ketika penulis dan satu pembaca, instans DB dengan tingkat promosi 0 tidak memiliki ukuran instans yang sama.

Mengelola churn koneksi Aurora PostgreSQL dengan pooling

Ketika aplikasi klien terhubung dan terputus begitu sering sehingga waktu respons klaster DB Aurora PostgreSQL melambat, klaster dikatakan mengalami churn koneksi. Setiap koneksi baru ke titik akhir klaster DB Aurora PostgreSQL menghabiskan sumber daya, sehingga akan mengurangi sumber daya yang dapat digunakan untuk memproses beban kerja yang sebenarnya. Churn koneksi adalah masalah yang kami sarankan agar Anda kelola dengan mengikuti beberapa praktik terbaik yang dibahas berikut.

Sebagai permulaan, Anda dapat mempersingkat waktu respons pada klaster DB Aurora PostgreSQL yang memiliki tingkat churn koneksi yang tinggi. Untuk melakukannya, Anda dapat menggunakan pooler koneksi, seperti Proksi RDS. Pooler koneksi menyediakan cache koneksi siap pakai untuk klien. Hampir semua versi Aurora PostgreSQL mendukung Proksi RDS. Untuk informasi selengkapnya, lihat Proksi Amazon RDS dengan Aurora PostgreSQL.

Jika versi spesifik Aurora PostgreSQL Anda tidak mendukung Proksi RDS, Anda dapat menggunakan pooler koneksi lain yang kompatibel dengan PostgreSQL, seperti PgBouncer. Untuk mempelajari selengkapnya, lihat situs web PgBouncer.

Untuk melihat apakah klaster DB Aurora PostgreSQL Anda dapat memperoleh manfaat dari pooling koneksi, Anda dapat memeriksa file `postgresql.log` apakah ada koneksi dan pemutusan koneksi. Anda juga dapat menggunakan Wawasan Performa untuk mengetahui jumlah koneksi churn yang dialami klaster DB Aurora PostgreSQL Anda. Di bagian berikut ini, Anda dapat menemukan informasi tentang kedua topik tersebut.

Logging koneksi dan pemutusan koneksi

Parameter `log_connections` dan `log_disconnections` PostgreSQL dapat menangkap koneksi dan pemutusan koneksi ke instans penulis klaster DB Aurora PostgreSQL. Secara default, parameter ini dinonaktifkan. Untuk mengaktifkan parameter ini, gunakan grup parameter kustom dan aktifkan dengan mengubah nilainya menjadi 1. Untuk informasi selengkapnya tentang grup parameter kustom, lihat Bekerja dengan grup parameter klaster DB. Untuk memeriksa pengaturan, hubungkan ke titik akhir klaster DB Anda untuk Aurora PostgreSQL dengan menggunakan psql dan kueri sebagai berikut.

```
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_connections';
 setting
---------
on
(1 row)
labdb=> SELECT setting FROM pg_settings
  WHERE name = 'log_disconnections';
setting
---------
on
(1 row)
```

Dengan kedua parameter ini diaktifkan, log akan menangkap semua koneksi dan pemutusan koneksi baru. Anda melihat pengguna dan basis data untuk setiap koneksi baru yang diotorisasi. Pada waktu pemutusan koneksi, durasi sesi juga dicatat dalam log, seperti yang ditunjukkan pada contoh berikut.

```
2022-03-07 21:44:53.978 UTC [16641] LOG: connection authorized: user=labtek
 database=labdb application_name=psql
2022-03-07 21:44:55.718 UTC [16641] LOG: disconnection: session time: 0:00:01.740
 user=labtek database=labdb host=[local]
```

Untuk memeriksa churn koneksi aplikasi Anda, aktifkan parameter ini jika belum aktif. Kemudian kumpulkan data di log PostgreSQL untuk analisis dengan menjalankan aplikasi Anda dengan beban kerja dan periode waktu yang realistis. Anda dapat melihat file log di konsol RDS. Pilih instans penulis klaster DB Aurora PostgreSQL Anda, lalu pilih tab Log & peristiwa. Untuk informasi selengkapnya, lihat Melihat dan mencantumkan file log basis data.

Atau, Anda dapat mengunduh file log dari konsol dan menggunakan urutan perintah berikut. Urutan ini menemukan jumlah total koneksi yang diotorisasi dan diputus per menit.

```
grep "connection authorized\|disconnection: session time:"
 postgresql.log.2022-03-21-16|\
awk {'print $1,$2}' |\
sort |\
uniq -c |\
sort -n -k1
```

Dalam contoh output ini, Anda dapat melihat lonjakan koneksi yang diotorisasi diikuti dengan pemutusan koneksi mulai dari 16:12:10.

```
.....
,.......
.........
5 2022-03-21 16:11:55 connection authorized:
9 2022-03-21 16:11:55 disconnection: session
5 2022-03-21 16:11:56 connection authorized:
5 2022-03-21 16:11:57 connection authorized:
5 2022-03-21 16:11:57 disconnection: session
32 2022-03-21 16:12:10 connection authorized:
30 2022-03-21 16:12:10 disconnection: session
31 2022-03-21 16:12:11 connection authorized:
27 2022-03-21 16:12:11 disconnection: session
27 2022-03-21 16:12:12 connection authorized:
27 2022-03-21 16:12:12 disconnection: session
41 2022-03-21 16:12:13 connection authorized:
47 2022-03-21 16:12:13 disconnection: session
46 2022-03-21 16:12:14 connection authorized:
41 2022-03-21 16:12:14 disconnection: session
24 2022-03-21 16:12:15 connection authorized:
29 2022-03-21 16:12:15 disconnection: session
28 2022-03-21 16:12:16 connection authorized:
24 2022-03-21 16:12:16 disconnection: session
40 2022-03-21 16:12:17 connection authorized:
42 2022-03-21 16:12:17 disconnection: session
40 2022-03-21 16:12:18 connection authorized:
40 2022-03-21 16:12:18 disconnection: session
.....
,.......
.........
1 2022-03-21 16:14:10 connection authorized:
1 2022-03-21 16:14:10 disconnection: session
1 2022-03-21 16:15:00 connection authorized:
```

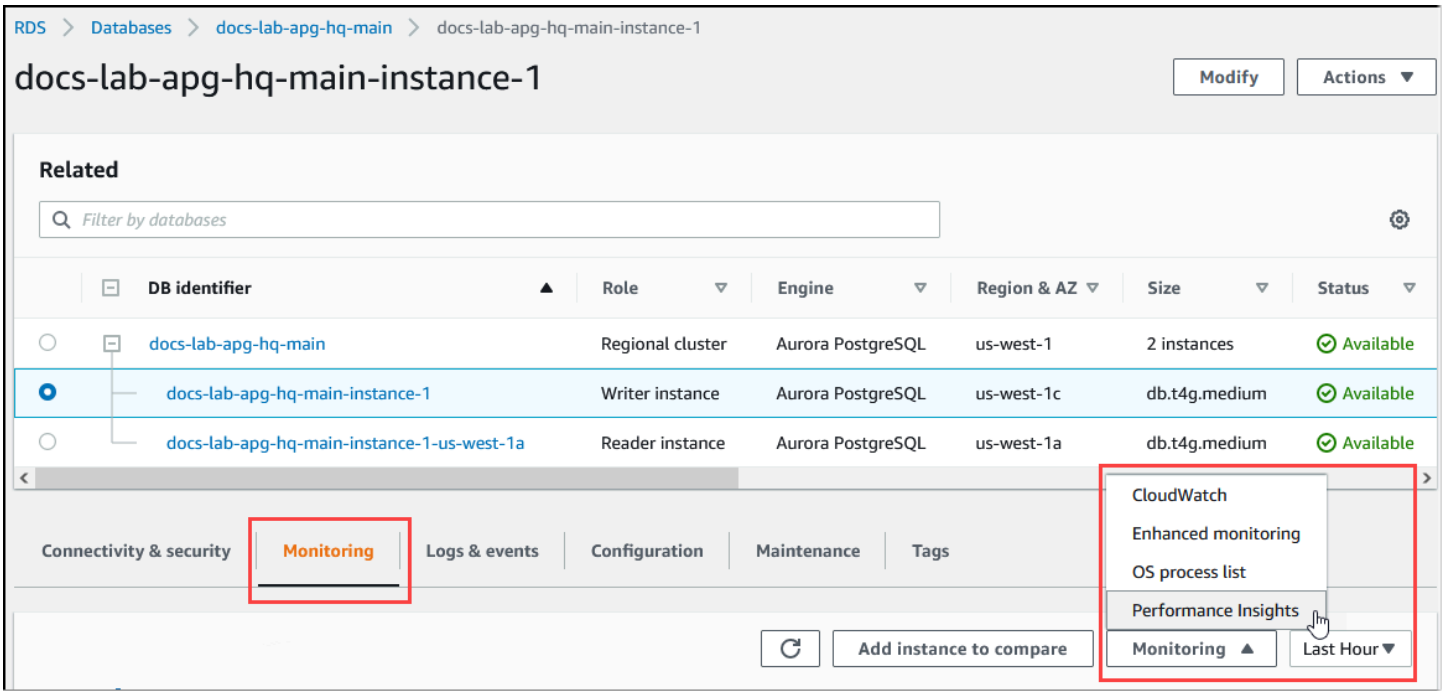
```
1 2022-03-21 16:16:00 connection authorized:
```

Dengan informasi ini, Anda dapat memutuskan apakah beban kerja Anda dapat memperoleh manfaat dari pooler koneksi. Untuk analisis yang lebih mendetail, Anda dapat menggunakan Wawasan Performa.

Mendeteksi churn koneksi dengan Wawasan Performa

Anda dapat menggunakan Wawasan Performa untuk menilai jumlah koneksi pada klaster DB Aurora PostgreSQL-Compatible Edition. Saat Anda membuat klaster DB Aurora PostgreSQL, pengaturan untuk Wawasan Performa diaktifkan secara default. Jika Anda menghapus pilihan ini saat membuat klaster DB, ubah klaster Anda agar mengaktifkan fitur tersebut. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora.

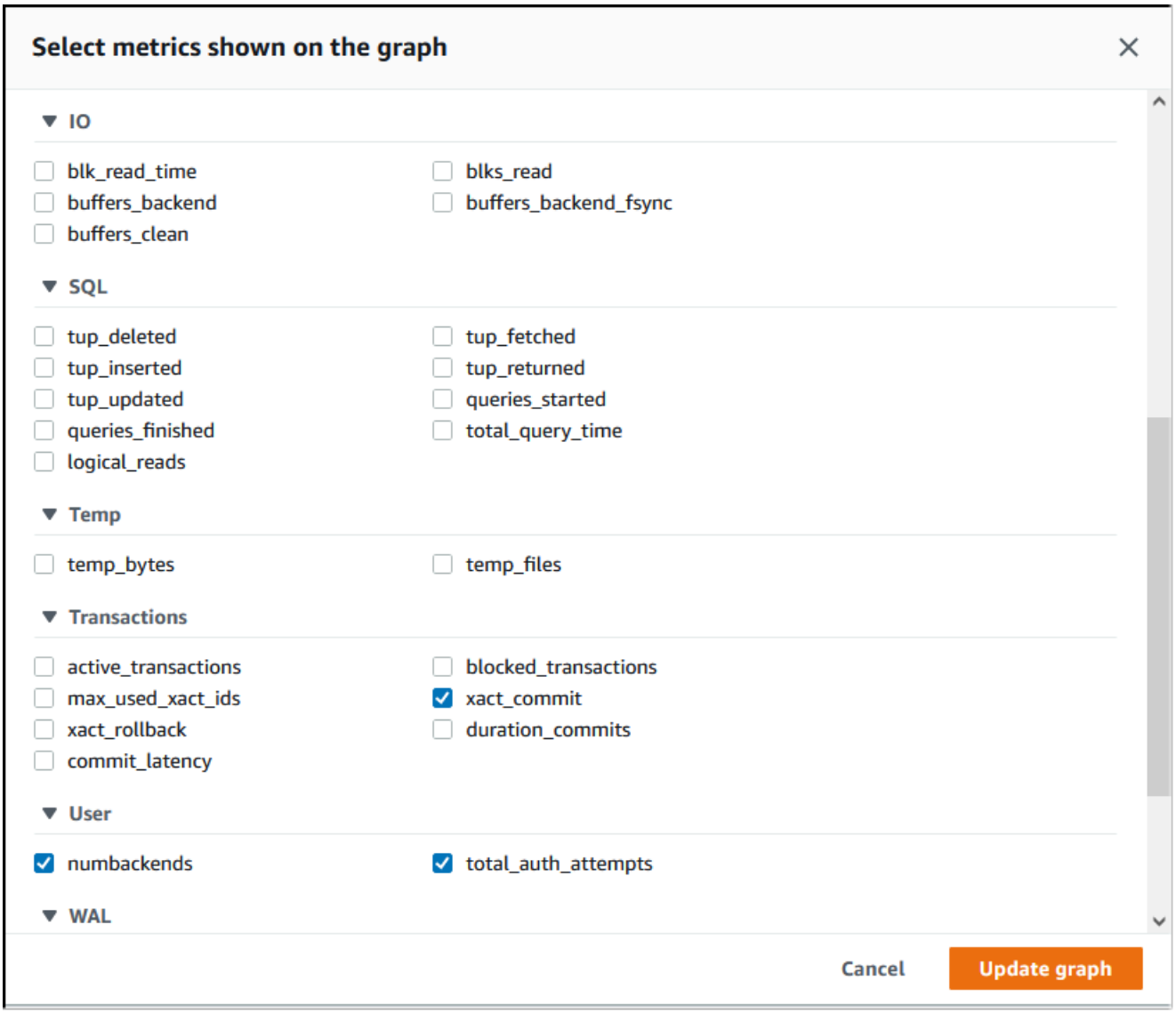
Dengan Wawasan Performa yang berjalan di klaster DB Aurora PostgreSQL, Anda dapat memilih metrik yang ingin dipantau. Anda dapat mengakses Wawasan Performa dari panel navigasi di konsol. Anda juga dapat mengakses Wawasan Performa dari tab Pemantauan untuk instans penulis klaster DB Aurora PostgreSQL Anda, seperti yang ditunjukkan pada gambar berikut.



Dari konsol Wawasan Performa, pilih Kelola metrik. Untuk menganalisis aktivitas koneksi dan pemutusan koneksi klaster DB Aurora PostgreSQL Anda, pilih metrik berikut. Ini semua adalah metrik dari PostgreSQL.

- `xact_commit` – Jumlah transaksi yang dikomit.

- `total_auth_attempts` – Jumlah percobaan koneksi pengguna yang diautentikasi per menit.
- `numbackends` – Jumlah backend yang saat ini terhubung ke basis data.

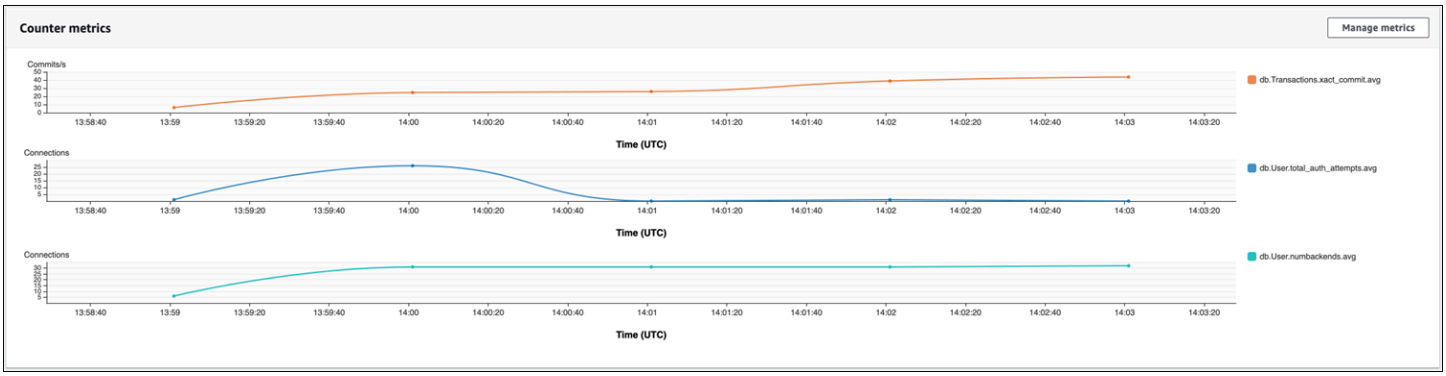


Untuk menyimpan pengaturan dan menampilkan aktivitas koneksi, pilih Perbarui grafik.

Pada gambar berikut, Anda dapat melihat dampak dari pgbench yang dijalankan dengan 100 pengguna. Garis yang menunjukkan koneksi berada pada kemiringan ke atas yang konsisten. Untuk mempelajari selengkapnya tentang pgbench dan cara menggunakannya, lihat pgbench dalam dokumentasi PostgreSQL.

Gambar ini menunjukkan bahwa menjalankan beban kerja dengan sedikitnya 100 pengguna tanpa pooler koneksi dapat menyebabkan peningkatan yang signifikan dalam jumlah `total_auth_attempts` selama durasi pemrosesan beban kerja.

Dengan pooling koneksi Proksi RDS, percobaan koneksi meningkat pada awal beban kerja. Setelah mengatur pool koneksi, rata-ratanya menurun. Sumber daya yang digunakan oleh transaksi dan penggunaan backend tetap konsisten selama pemrosesan beban kerja.



Untuk informasi selengkapnya tentang cara menggunakan Wawasan Performa dengan klaster DB Aurora PostgreSQL, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora. Untuk menganalisis metrik, lihat Menganalisis metrik dengan dasbor Wawasan Performa.

Menunjukkan manfaat dari pooling koneksi

Seperti disebutkan sebelumnya, jika Anda menentukan bahwa klaster DB Aurora PostgreSQL Anda memiliki masalah churn koneksi, Anda dapat menggunakan Proksi RDS untuk meningkatkan performa. Di bagian berikut ini, Anda dapat menemukan contoh yang menunjukkan perbedaan dalam memproses beban kerja saat koneksi di-pool dan saat tidak. Contoh ini menggunakan pgbench untuk memodelkan beban kerja transaksi.

Seperti psql, pgbench adalah aplikasi klien PostgreSQL yang dapat Anda instal dan jalankan dari mesin klien lokal Anda. Anda juga dapat menginstal dan menjalankannya dari instans Amazon

EC2 yang Anda gunakan untuk mengelola klaster DB Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat pgbench dalam dokumentasi PostgreSQL.

Untuk mempelajari contoh ini, pertama-tama Anda perlu membuat lingkungan pgbench dalam basis data Anda. Perintah berikut adalah templat dasar untuk menginisialisasi tabel pgbench dalam basis data yang ditentukan. Contoh ini menggunakan akun pengguna utama default, `postgres`, untuk login. Ubah akunnya sesuai kebutuhan untuk klaster DB Aurora PostgreSQL Anda. Anda membuat lingkungan pgbench dalam basis data pada instans penulis klaster Anda.

> **Note**
>
> Proses inisialisasi pgbench menghapus dan membuat ulang tabel bernama `pgbench_accounts`, `pgbench_branches`, `pgbench_history`, dan `pgbench_tellers`. Pastikan basis data yang Anda pilih untuk *dbname* ketika Anda menginisialisasi pgbench tidak menggunakan nama-nama ini.

```
pgbench -U postgres -h db-cluster-instance-1.111122223333.aws-region.rds.amazonaws.com
 -p 5432 -d -i -s 50 dbname
```

Untuk pgbench, tentukan parameter berikut.

-d

Menghasilkan laporan debugging saat pgbench berjalan.

-h

Menentukan titik akhir instans penulis klaster DB Aurora PostgreSQL.

-i

Menginisialisasi lingkungan pgbench dalam basis data untuk pengujian tolok ukur.

-p

Mengidentifikasi port yang digunakan untuk koneksi basis data. Port default untuk Aurora PostgreSQL biasanya adalah 5432 atau 5433.

-s

Menentukan faktor penskalaan yang akan digunakan untuk mengisi tabel dengan baris. Faktor penskalaan default adalah 1, yang menghasilkan 1 baris dalam tabel `pgbench_branches`, 10 baris dalam tabel `pgbench_tellers`, dan 100.000 baris dalam tabel `pgbench_accounts`.

-U

Menentukan akun pengguna untuk instans penulis klaster DB Aurora PostgreSQL.

Setelah lingkungan pgbench diatur, Anda kemudian dapat menjalankan pengujian tolok ukur dengan dan tanpa pooling koneksi. Pengujian default terdiri dari serangkaian lima perintah SELECT, UPDATE, dan INSERT per transaksi yang berjalan berulang kali selama waktu yang ditentukan. Anda dapat menentukan faktor penskalaan, jumlah klien, dan detail lainnya untuk memodelkan kasus penggunaan Anda sendiri.

Sebagai contoh, perintah yang ditampilkan berikutnya akan menjalankan tolok ukur selama 60 detik (opsi -T, untuk waktu) dengan 20 koneksi konkuren (opsi -c). Opsi -C membuat pengujian menggunakan koneksi baru setiap kali dijalankan, bukan sekali per sesi klien. Pengaturan ini memberi Anda indikasi overhead koneksi.

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 -C labdb
Password:**********
pgbench (14.3, server 13.3)
  starting vacuum...end.
  transaction type: <builtin: TPC-B (sort of)>
    scaling factor: 50
    query mode: simple
    number of clients: 20
    number of threads: 1
    duration: 60 s
    number of transactions actually processed: 495
    latency average = 2430.798 ms
    average connection time = 120.330 ms
    tps = 8.227750 (including reconnection times)
```

Dengan menjalankan pgbench pada instans penulis klaster DB Aurora PostgreSQL tanpa menggunakan kembali koneksi, akan terlihat bahwa hanya sekitar 8 transaksi yang diproses setiap detik. Artinya, ada total 495 transaksi selama pengujian 1 menit.

Jika Anda menggunakan kembali koneksi, respons dari klaster DB Aurora PostgreSQL untuk jumlah pengguna tersebut hampir 20 kali lebih cepat. Dengan penggunaan kembali koneksi, total 9.042 transaksi diproses dibandingkan dengan 495 dalam jumlah waktu yang sama dan untuk jumlah koneksi pengguna yang sama. Perbedaannya adalah bahwa dalam contoh berikut ini, setiap koneksi digunakan kembali.

```
pgbench -h docs-lab-apg-133-test-instance-1.c3zr2auzukpa.us-west-1.rds.amazonaws.com -U
 postgres -p 5432 -T 60 -c 20 labdb
Password:*********
pgbench (14.3, server 13.3)
      starting vacuum...end.
      transaction type: <builtin: TPC-B (sort of)>
      scaling factor: 50
      query mode: simple
      number of clients: 20
      number of threads: 1
      duration: 60 s
      number of transactions actually processed: 9042
      latency average = 127.880 ms
      initial connection time = 2311.188 ms
      tps = 156.396765 (without initial connection time)
```

Contoh ini menunjukkan kepada Anda bahwa pooling koneksi dapat secara signifikan mempersingkat waktu respons. Untuk informasi tentang cara menyiapkan Proksi RDS untuk klaster DB Aurora PostgreSQL Anda, lihat Menggunakan Proksi Amazon RDS untuk Aurora.

Menyetel parameter memori untuk Aurora PostgreSQL

Di Amazon Aurora PostgreSQL, Anda dapat menggunakan beberapa parameter yang mengontrol jumlah memori yang digunakan untuk berbagai tugas pemrosesan. Jika sebuah tugas membutuhkan lebih banyak memori daripada jumlah yang ditetapkan untuk parameter tertentu, Aurora PostgreSQL akan menggunakan sumber daya lain untuk pemrosesan, seperti dengan menulis ke disk. Hal ini dapat menyebabkan klaster DB Aurora PostgreSQL Anda menjadi lambat atau berpotensi berhenti, dengan kesalahan kehabisan memori.

Pengaturan default untuk setiap parameter memori biasanya dapat menangani tugas pemrosesan yang dimaksudkan. Namun, Anda juga dapat menyetel parameter terkait memori klaster DB Aurora PostgreSQL Anda. Anda melakukan penyetelan ini untuk memastikan bahwa memori yang cukup dialokasikan untuk memproses beban kerja spesifik Anda.

Di bagian berikut ini, Anda dapat menemukan informasi tentang parameter yang mengontrol manajemen memori. Anda juga dapat mempelajari cara menilai pemanfaatan memori.

Memeriksa dan mengatur nilai parameter

Parameter yang dapat Anda atur untuk mengelola memori dan menilai penggunaan memori klaster DB Aurora PostgreSQL Anda meliputi:

- `work_mem` – Menentukan jumlah memori yang digunakan klaster DB Aurora PostgreSQL untuk operasi pengurutan internal dan tabel hash sebelum menulis ke file disk sementara.
- `log_temp_files` – Mencatat log pembuatan file sementara, nama file, dan ukuran. Ketika parameter ini diaktifkan, entri log disimpan untuk setiap file sementara yang dibuat. Aktifkan parameter ini untuk melihat seberapa sering klaster DB Aurora PostgreSQL Anda perlu menulis ke disk. Nonaktifkan lagi setelah Anda mengumpulkan informasi tentang pembuatan file sementara klaster DB Aurora PostgreSQL Anda untuk menghindari logging yang berlebihan.
- `logical_decoding_work_mem` – Menentukan jumlah memori (dalam megabyte) yang akan digunakan untuk pendekodean logis. Pendekodean logis adalah proses yang digunakan untuk membuat replika. Proses ini dilakukan dengan mengonversi data dari file write-ahead log (WAL) menjadi output streaming logis yang dibutuhkan oleh target.

  Nilai parameter ini membuat buffer tunggal dari ukuran yang ditentukan untuk setiap koneksi replikasi. Secara default, nilainya adalah 65536 KB. Setelah buffer ini diisi, kelebihannya akan ditulis ke disk sebagai file. Untuk meminimalkan aktivitas disk, Anda dapat mengatur nilai parameter ini ke nilai yang jauh lebih tinggi daripada `work_mem`.

Ini semua adalah parameter dinamis, sehingga Anda dapat mengubahnya untuk sesi saat ini. Untuk melakukannya, hubungkan ke klaster DB Aurora PostgreSQL dengan psql dan menggunakan pernyataan `SET`, seperti yang ditunjukkan berikut.

```
SET parameter_name TO parameter_value;
```

Pengaturan sesi hanya berlaku selama sesi berlangsung. Ketika sesi berakhir, parameter akan kembali ke pengaturannya dalam grup parameter klaster DB. Sebelum mengubah parameter apa pun, periksa terlebih dahulu nilai saat ini dengan mengkueri tabel `pg_settings` sebagai berikut.

```
SELECT unit, setting, max_val
   FROM pg_settings WHERE name='parameter_name';
```

Misalnya, untuk menemukan nilai parameter `work_mem`, hubungkan ke instans penulis klaster DB Aurora PostgreSQL dan jalankan kueri berikut.

```
SELECT unit, setting, max_val, pg_size_pretty(max_val::numeric)
  FROM pg_settings WHERE name='work_mem';
unit  | setting  | max_val   | pg_size_pretty
------+----------+-----------+---------------
 kB   | 1024     | 2147483647| 2048 MB
(1 row)
```

Untuk mengubah pengaturan parameter agar dipersistensi, grup parameter klaster DB kustom harus digunakan. Setelah menjalankan klaster DB Aurora PostgreSQL Anda dengan nilai yang berbeda-beda untuk parameter ini menggunakan pernyataan `SET`, Anda dapat membuat grup parameter kustom dan menerapkannya ke klaster DB Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Bekerja dengan grup parameter.

Memahami parameter memori kerja

Parameter memori kerja (`work_mem`) menentukan jumlah maksimum memori yang dapat digunakan Aurora PostgreSQL untuk memproses kueri kompleks. Kueri kompleks mencakup kueri yang memerlukan operasi pengurutan atau pengelompokan, dengan kata lain, kueri yang menggunakan klausa berikut:

- ORDER BY
- DISTINCT
- GROUP BY
- JOIN (MERGE dan HASH)

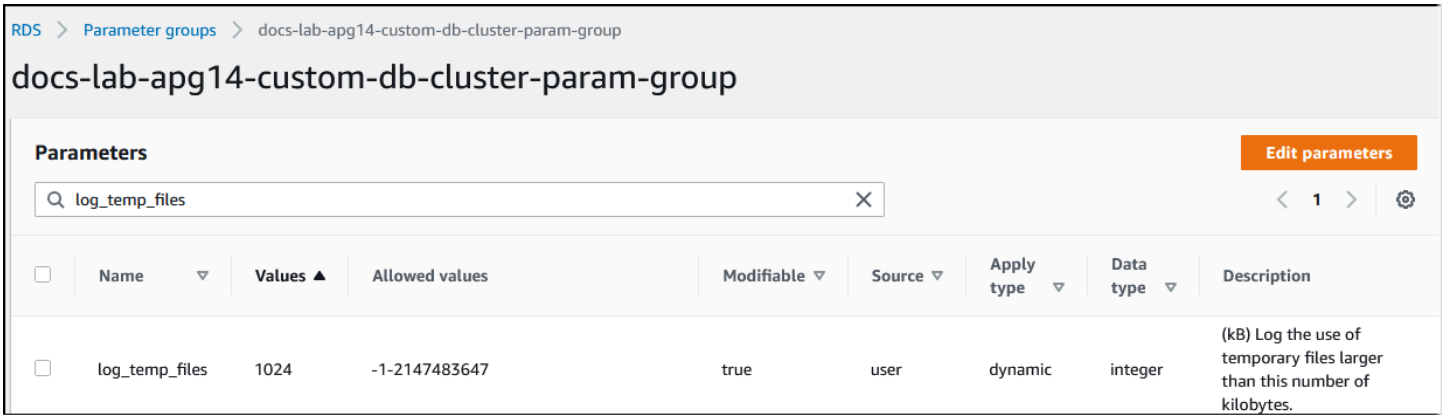
Perencana kueri secara tidak langsung memengaruhi bagaimana klaster DB Aurora PostgreSQL Anda menggunakan memori kerja. Perencana kueri menghasilkan rencana eksekusi untuk memproses pernyataan SQL. Rencana tertentu dapat memecah kueri kompleks menjadi beberapa unit kerja yang dapat dijalankan secara paralel. Jika memungkinkan, Aurora PostgreSQL akan menggunakan jumlah memori yang ditentukan dalam parameter `work_mem` untuk setiap sesi sebelum menulis ke disk untuk setiap proses paralel.

Beberapa pengguna basis data yang menjalankan banyak operasi secara bersamaan dan menghasilkan banyak unit kerja secara paralel dapat menghabiskan memori kerja yang dialokasikan untuk klaster DB Aurora PostgreSQL Anda. Hal ini dapat menyebabkan pembuatan file sementara

dan I/O disk yang berlebihan, atau lebih buruk lagi, dapat menyebabkan kesalahan kehabisan memori.

Mengidentifikasi penggunaan file sementara

Setiap kali memori yang diperlukan untuk memproses kueri melebihi nilai yang ditentukan dalam parameter `work_mem`, data kerja akan dialihkan ke disk dalam file sementara. Anda dapat melihat seberapa sering ini terjadi dengan mengaktifkan parameter `log_temp_files`. Secara default, parameter ini nonaktif (diatur ke -1). Untuk menangkap semua informasi file sementara, atur parameter ini ke 0. Atur `log_temp_files` ke bilangan bulat positif lainnya untuk menangkap informasi file sementara untuk file yang sama dengan atau lebih besar dari jumlah data tersebut (dalam kilobyte). Pada gambar berikut, Anda dapat melihat contoh dari AWS Management Console.



Setelah mengonfigurasi logging file sementara, Anda dapat menguji dengan beban kerja Anda sendiri untuk melihat apakah pengaturan memori kerja Anda cukup. Anda juga dapat menyimulasikan beban kerja dengan menggunakan pgbench, sebuah aplikasi tolok ukur sederhana dari komunitas PostgreSQL.

Contoh berikut menginisialisasi (`-i`) `pgbench` dengan membuat tabel dan baris yang diperlukan untuk menjalankan pengujian. Dalam contoh ini, faktor penskalaan (`-s` 50) membuat 50 baris dalam tabel `pgbench_branches`, 500 baris dalam `pgbench_tellers`, dan 5.000.000 baris dalam tabel `pgbench_accounts` di basis data `labdb`.

```
pgbench -U postgres -h your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -i -s 50 labdb
Password:
dropping old tables...
NOTICE:  table "pgbench_accounts" does not exist, skipping
NOTICE:  table "pgbench_branches" does not exist, skipping
NOTICE:  table "pgbench_history" does not exist, skipping
```

```
NOTICE:  table "pgbench_tellers" does not exist, skipping
creating tables...
generating data (client-side)...
5000000 of 5000000 tuples (100%) done (elapsed 15.46 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 61.13 s (drop tables 0.08 s, create tables 0.39 s, client-side generate 54.85
 s, vacuum 2.30 s, primary keys 3.51 s)
```

Setelah menginisialisasi lingkungan, Anda dapat menjalankan tolok ukur selama waktu (`-T`) dan jumlah klien (`-c`) tertentu. Contoh ini juga menggunakan opsi `-d` untuk menghasilkan informasi debugging seiring transaksi diproses oleh klaster DB Aurora PostgreSQL.

```
pgbench -h -U postgres your-cluster-instance-1.111122223333.aws-regionrds.amazonaws.com
 -p 5432 -d -T 60 -c 10 labdb
Password:*******
pgbench (14.3)
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 50
query mode: simple
number of clients: 10
number of threads: 1
duration: 60 s
number of transactions actually processed: 1408
latency average = 398.467 ms
initial connection time = 4280.846 ms
tps = 25.096201 (without initial connection time)
```

Untuk informasi selengkapnya tentang pgbench, lihat pgbench dalam dokumentasi PostgreSQL.

Anda dapat menggunakan perintah metacommand psql (\d) untuk membuat daftar relasi seperti tabel, tampilan, dan indeks yang dibuat oleh pgbench.

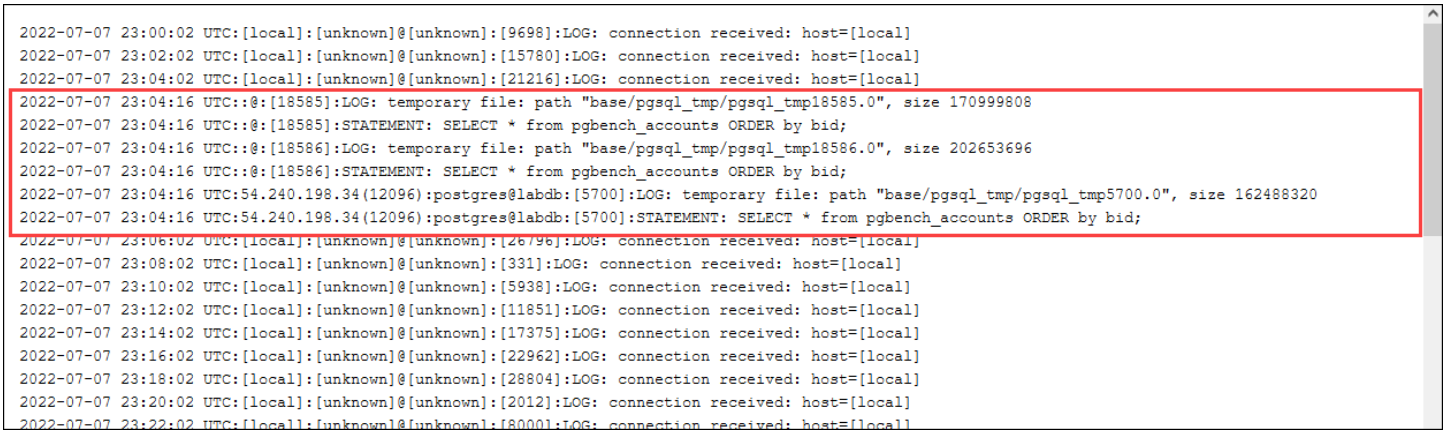
```
labdb=>  \d pgbench_accounts
 Table "public.pgbench_accounts"
  Column  |     Type      | Collation | Nullable | Default
----------+---------------+-----------+----------+---------
 aid      | integer       |           | not null |
 bid      | integer       |           |          |
 abalance | integer       |           |          |
 filler   | character(84) |           |          |
```

```
Indexes:
    "pgbench_accounts_pkey" PRIMARY KEY, btree (aid)
```

Seperti yang ditunjukkan pada output, tabel `pgbench_accounts` diindeks pada kolom `aid`. Untuk memastikan bahwa kueri berikutnya menggunakan memori kerja, jalankan kueri terhadap kolom yang tidak diindeks, seperti yang ditunjukkan pada contoh berikut.

```
postgres=> SELECT * FROM pgbench_accounts ORDER BY bid;
```

Periksa log apakah ada file sementara. Untuk melakukannya, buka AWS Management Console, pilih instans klaster DB Aurora PostgreSQL, lalu pilih tab Log & Peristiwa. Lihat log di konsol atau unduh untuk analisis lebih lanjut. Seperti yang ditunjukkan pada gambar berikut, ukuran file sementara yang diperlukan untuk memproses kueri menunjukkan bahwa Anda harus mempertimbangkan untuk meningkatkan jumlah yang ditentukan untuk parameter `work_mem`.



Anda dapat mengonfigurasi parameter ini secara berbeda untuk individu dan grup, berdasarkan kebutuhan operasional Anda. Misalnya, Anda dapat mengatur parameter `work_mem` ke 8 GB untuk peran bernama`dev_team`.

```
postgres=> ALTER ROLE dev_team SET work_mem='8GB';
```

Dengan pengaturan untuk `work_mem` ini, peran apa pun yang merupakan anggota peran `dev_team` akan diberi alokasi hingga 8 GB memori kerja.

Menggunakan indeks untuk waktu respons yang lebih cepat

Jika kueri Anda terlalu lama untuk memberikan hasil, Anda dapat memverifikasi bahwa indeks Anda digunakan seperti yang diharapkan. Pertama, aktifkan `\timing`, metacommand psql, sebagai berikut.

```
postgres=>  \timing on
```

Setelah mengaktifkan pewaktuan, gunakan pernyataan SELECT sederhana.

```
postgres=> SELECT COUNT(*) FROM
  (SELECT * FROM pgbench_accounts
  ORDER BY bid)
  AS accounts;
count
-------
5000000
(1 row)
Time: 3119.049 ms (00:03.119)
```

Seperti yang ditunjukkan pada output, kueri ini membutuhkan waktu lebih dari 3 detik untuk diselesaikan. Untuk mempersingkat waktu respons, buat indeks di `pgbench_accounts` sebagai berikut.

```
postgres=> CREATE INDEX ON pgbench_accounts(bid);
CREATE INDEX
```

Jalankan kembali kueri, dan perhatikan waktu respons yang lebih cepat. Dalam contoh ini, kueri selesai sekitar 5 kali lebih cepat, dalam waktu sekitar setengah detik.

```
postgres=>  SELECT COUNT(*) FROM (SELECT * FROM pgbench_accounts ORDER BY bid) AS
 accounts;
 count
-------
 5000000
(1 row)
Time: 567.095 ms
```

Menyesuaikan memori kerja untuk pendekodean logis

Replikasi logis telah tersedia di semua versi Aurora PostgreSQL sejak diperkenalkan di PostgreSQL versi 10. Saat Anda mengonfigurasi replikasi logis, Anda juga dapat mengatur parameter `logical_decoding_work_mem` untuk menentukan jumlah memori yang dapat digunakan proses pendekodean logis untuk proses pendekodean dan streaming.

Selama pendekodean logis, catatan write-ahead log (WAL) dikonversi menjadi pernyataan SQL yang kemudian dikirim ke target lain untuk replikasi logis atau tugas lain. Ketika transaksi ditulis ke WAL lalu dikonversi, seluruh transaksi harus sesuai dengan nilai yang ditentukan untuk `logical_decoding_work_mem`. Secara default, parameter ini diatur ke 65536 KB. Setiap kelebihan akan ditulis ke disk. Artinya data tersebut harus dibaca ulang dari disk sebelum dapat dikirim ke tujuannya, sehingga memperlambat prosesnya secara keseluruhan.

Anda dapat menilai jumlah kelebihan transaksi dalam beban kerja Anda saat ini pada titik waktu tertentu dengan menggunakan fungsi `aurora_stat_file` seperti yang ditunjukkan pada contoh berikut.

```
SELECT split_part (filename, '/', 2)
   AS slot_name, count(1) AS num_spill_files,
   sum(used_bytes) AS slot_total_bytes,
   pg_size_pretty(sum(used_bytes)) AS slot_total_size
   FROM aurora_stat_file()
   WHERE filename like '%spill%'
   GROUP BY 1;
  slot_name | num_spill_files | slot_total_bytes | slot_total_size
------------+-----------------+------------------+-----------------
 slot_name  |       590       |      411600000   | 393 MB
(1 row)
```

Kueri ini menghasilkan jumlah dan ukuran spill file pada klaster DB Aurora PostgreSQL Anda saat kueri diinvokasi. Beban kerja yang berjalan lebih lama mungkin belum memiliki spill file pada disk. Untuk membuat profil beban kerja yang berjalan lama, sebaiknya Anda membuat tabel untuk menangkap informasi spill file saat beban kerja berjalan. Anda dapat membuat tabel sebagai berikut.

```
CREATE TABLE spill_file_tracking AS
    SELECT now() AS spill_time,*
    FROM aurora_stat_file()
    WHERE filename LIKE '%spill%';
```

Untuk melihat bagaimana spill file digunakan selama replikasi logis, siapkan penerbit dan pelanggan lalu mulai replikasi sederhana. Untuk informasi selengkapnya, lihat Menyiapkan replikasi logis untuk klaster DB Aurora PostgreSQL Anda. Dengan replikasi yang sedang berlangsung, Anda dapat membuat pekerjaan yang menangkap hasil yang ditetapkan dari fungsi spill file `aurora_stat_file()` sebagai berikut.

```
INSERT INTO spill_file_tracking
```

```
  SELECT now(),*
  FROM aurora_stat_file()
  WHERE filename LIKE '%spill%';
```

Gunakan perintah psql berikut untuk menjalankan pekerjaan sekali per detik.

```
\watch 0.5
```

Saat pekerjaan sedang berjalan, hubungkan ke instans penulis dari sesi psql lain. Gunakan rangkaian pernyataan berikut untuk menjalankan beban kerja yang melebihi konfigurasi memori dan menyebabkan Aurora PostgreSQL membuat spill file.

```
labdb=> CREATE TABLE my_table (a int PRIMARY KEY, b int);
CREATE TABLE
labdb=> INSERT INTO my_table SELECT x,x FROM generate_series(0,10000000) x;
INSERT 0 10000001
labdb=> UPDATE my_table SET b=b+1;
UPDATE 10000001
```

Pernyataan ini membutuhkan waktu beberapa menit untuk selesai. Setelah selesai, tekan tombol Ctrl dan tombol C bersama-sama untuk menghentikan fungsi pemantauan. Kemudian, gunakan perintah berikut untuk membuat tabel untuk menyimpan informasi tentang penggunaan spill file klaster DB Aurora PostgreSQL.

```
SELECT spill_time, split_part (filename, '/', 2)
    AS slot_name, count(1)
    AS spills, sum(used_bytes)
    AS slot_total_bytes, pg_size_pretty(sum(used_bytes))
    AS slot_total_size FROM spill_file_tracking
  GROUP BY 1,2 ORDER BY 1;
                  spill_time | slot_name             | spills | slot_total_bytes |
 slot_total_size
------------------------------+-----------------------+--------+------------------
+-----------------
2022-04-15 13:42:52.528272+00 | replication_slot_name | 1      | 142352280        | 136
 MB
2022-04-15 14:11:33.962216+00 | replication_slot_name | 4      | 467637996        | 446
 MB
2022-04-15 14:12:00.997636+00 | replication_slot_name | 4      | 569409176        | 543
 MB
```

```
2022-04-15 14:12:03.030245+00 | replication_slot_name | 4      | 569409176        | 543
 MB
2022-04-15 14:12:05.059761+00 | replication_slot_name | 5      | 618410996        | 590
 MB
2022-04-15 14:12:07.22905+00  | replication_slot_name | 5      | 640585316        | 611
 MB
(6 rows)
```

Output ini menunjukkan bahwa contoh yang dijalankan telah membuat lima spill file yang menggunakan memori 611 MB. Untuk menghindari penulisan ke disk, kami menyarankan agar mengatur parameter `logical_decoding_work_mem` ke ukuran memori tertinggi berikutnya, 1024.

Menggunakan CloudWatch metrik Amazon untuk menganalisis penggunaan sumber daya untuk Aurora PostgreSQL

Aurora secara otomatis mengirimkan data metrik ke CloudWatch dalam periode 1 menit. Anda dapat menganalisis penggunaan sumber daya untuk Aurora PostgreSQL menggunakan metrik. CloudWatch Anda dapat mengevaluasi throughput jaringan dan penggunaan jaringan dengan metrik tersebut.

Mengevaluasi throughput jaringan dengan CloudWatch

Ketika penggunaan sistem Anda mendekati batas sumber daya untuk jenis instans Anda, pemrosesan dapat melambat. Anda dapat menggunakan Wawasan CloudWatch Log untuk memantau penggunaan sumber daya penyimpanan Anda dan memastikan bahwa sumber daya yang cukup tersedia. Bila diperlukan, Anda dapat mengubah instans DB ke kelas instans yang lebih besar.

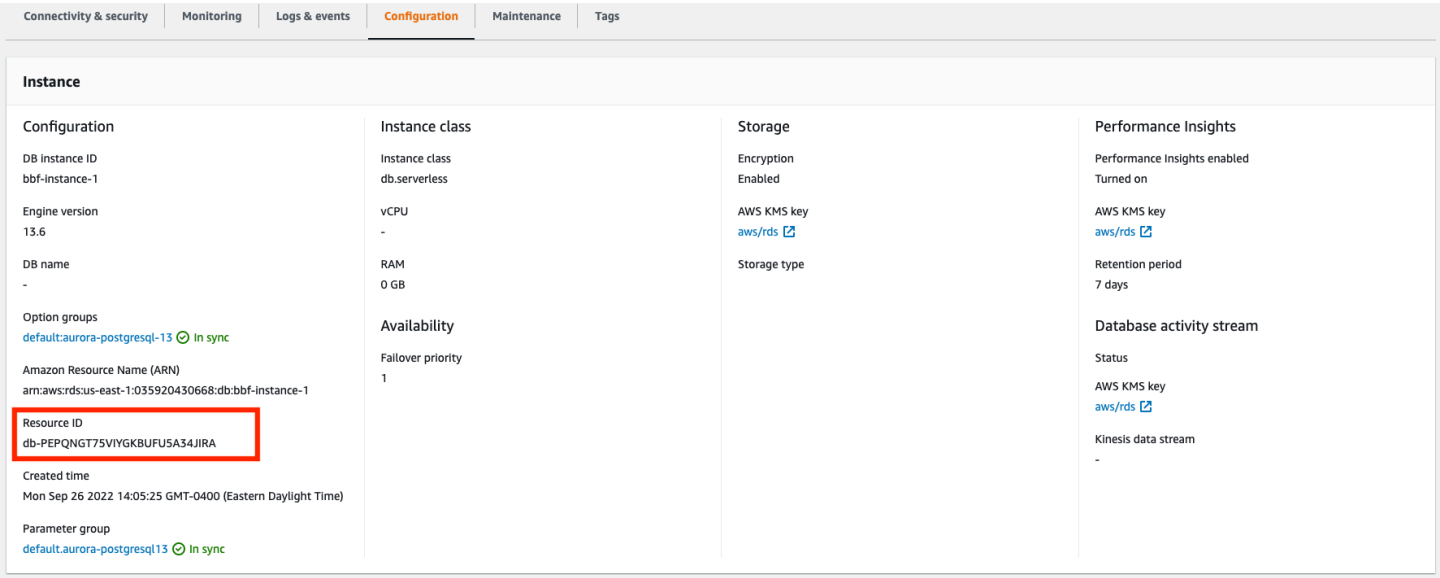
Pemrosesan penyimpanan Aurora mungkin lambat karena:

- Bandwidth jaringan tidak mencukupi antara instans DB dan klien.
- Bandwidth jaringan tidak mencukupi untuk subsistem penyimpanan.
- Beban kerja yang besar untuk jenis instans Anda.

Anda dapat meminta Wawasan CloudWatch Log untuk menghasilkan grafik penggunaan sumber daya penyimpanan Aurora untuk memantau sumber daya. Grafik tersebut menampilkan metrik dan pemanfaatan CPU untuk membantu Anda memutuskan apakah akan menaikkan skala menjadi instans yang lebih besar. Untuk informasi tentang sintaks kueri untuk Wawasan CloudWatch Log, lihat sintaks kueri Wawasan CloudWatch Log

Untuk menggunakannya CloudWatch, Anda perlu mengekspor file log Aurora PostgreSQL Anda ke file log. CloudWatch Anda juga dapat memodifikasi klaster yang ada untuk mengekspor log ke CloudWatch. Untuk informasi tentang mengekspor log ke CloudWatch, lihatMengaktifkan opsi untuk menerbitkan log ke Amazon CloudWatch.

Anda memerlukan ID Sumber Daya instans DB Anda untuk menanyakan Wawasan CloudWatch Log. ID Sumber Daya tersedia di tab Konfigurasi di konsol Anda:



Untuk mengueri file log Anda untuk metrik penyimpanan sumber daya:

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.

   Halaman beranda CloudWatch ikhtisar muncul.

2. Jika perlu, ubah Wilayah AWS. Di bilah navigasi, pilih Wilayah AWS tempat AWS sumber daya Anda berada. Untuk informasi selengkapnya, lihat Wilayah dan titik akhir.

3. Di panel navigasi, pilih Log, lalu Wawasan Log.

   Halaman Wawasan Log muncul.

4. Pilih file log dari daftar tarik-turun untuk menganalisis.

5. Masukkan kueri berikut di bidang, ganti `<resource ID>` dengan ID sumber daya klaster DB Anda:

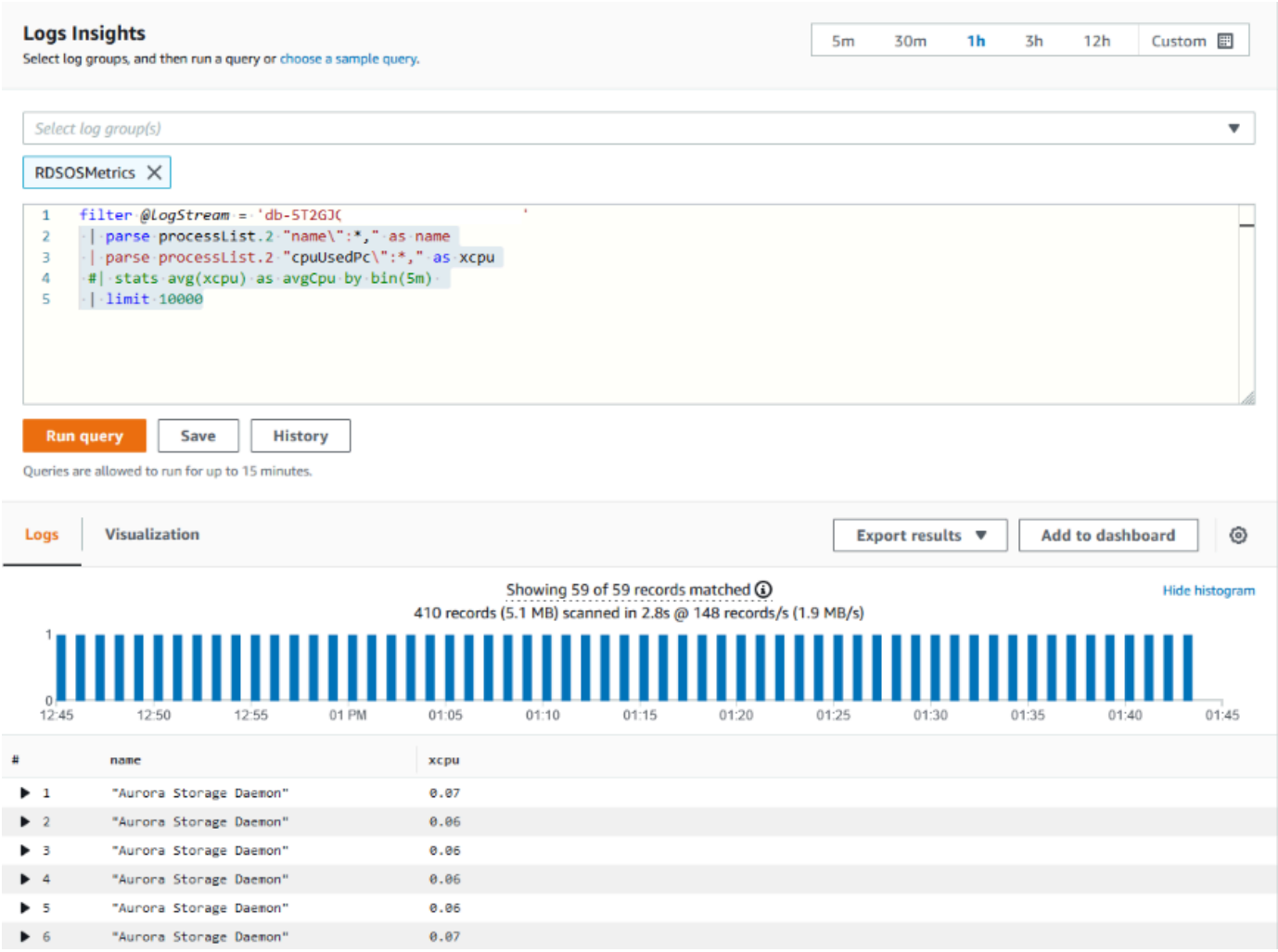
   ```
   filter @logStream = <resource ID> | parse @message "\"Aurora Storage
   Daemon\"*memoryUsedPc\":*,\"cpuUsedPc\":*," as a,memoryUsedPc,cpuUsedPc
   ```

```
| display memoryUsedPc,cpuUsedPc #| stats avg(xcpu) as avgCpu by
bin(5m) | limit 10000
```

6. Klik Jalankan kueri.

   Grafik pemanfaatan penyimpanan ditampilkan.

   Gambar berikut menyediakan halaman Wawasan Log dan tampilan grafik.



Mengevaluasi penggunaan instans DB dengan metrik CloudWatch

Anda dapat menggunakan CloudWatch metrik untuk melihat throughput instans dan mengetahui apakah kelas instans menyediakan sumber daya yang memadai untuk aplikasi Anda. Untuk informasi tentang batas kelas instans DB Anda, buka Spesifikasi perangkat keras kelas instans DB untuk Aurora dan cari spesifikasi untuk kelas instans DB Anda guna menemukan performa jaringan.

Jika penggunaan instans DB Anda mendekati batas kelas instans, maka performa mungkin mulai melambat. CloudWatch Metrik dapat mengonfirmasi situasi ini sehingga Anda dapat merencanakan untuk meningkatkan skala secara manual ke kelas instance yang lebih besar.

Gabungkan nilai CloudWatch metrik berikut untuk mengetahui apakah Anda mendekati batas kelas instance:

- NetworkThroughput— Jumlah throughput jaringan yang diterima dan ditransmisikan oleh klien untuk setiap instance di cluster Aurora DB. Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster.
- StorageNetworkThroughput— Jumlah throughput jaringan yang diterima dan dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB.

Tambahkan ke NetworkThroughputStorageNetworkThroughputuntuk menemukan throughput jaringan yang diterima dari dan dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster Aurora DB Anda. Batas kelas instans untuk instans Anda harus lebih besar dari jumlah kedua metrik gabungan ini.

Anda dapat menggunakan metrik berikut untuk meninjau detail tambahan lalu lintas jaringan dari aplikasi klien Anda saat mengirim dan menerima:

- NetworkReceiveThroughput— Jumlah throughput jaringan yang diterima dari klien oleh setiap instance di cluster DB PostgreSQL Aurora. Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster.
- NetworkTransmitThroughput— Jumlah throughput jaringan yang dikirim ke klien oleh setiap instance di cluster Aurora DB. Throughput ini tidak mencakup lalu lintas jaringan di antara instans dalam klaster DB dan volume klaster.
- StorageNetworkReceiveThroughput— Jumlah throughput jaringan yang diterima dari subsistem penyimpanan Aurora oleh setiap instance di cluster DB.
- StorageNetworkTransmitThroughput— Jumlah throughput jaringan yang dikirim ke subsistem penyimpanan Aurora oleh setiap instance di cluster DB.

Tambahkan semua metrik ini bersama-sama untuk mengevaluasi bagaimana penggunaan jaringan Anda dibandingkan dengan batas kelas instans. Batas kelas instans harus lebih besar dari jumlah metrik gabungan ini.

Batas jaringan dan pemanfaatan CPU untuk penyimpanan saling berkaitan. Ketika throughput jaringan meningkat, maka pemanfaatan CPU juga meningkat. Pemantauan penggunaan CPU dan jaringan memberikan informasi tentang bagaimana dan mengapa sumber daya habis.

Untuk membantu meminimalkan penggunaan jaringan, Anda dapat mempertimbangkan:

- Menggunakan kelas instans yang lebih besar.
- Menggunakan strategi partisi `pg_partman`.
- Membagi permintaan tulis dalam batch untuk mengurangi keseluruhan transaksi.
- Mengarahkan beban kerja hanya-baca ke instans hanya-baca.
- Menghapus indeks yang tidak digunakan.
- Memeriksa objek bloat dan VACUUM. Dalam kasus bloat parah, gunakan ekstensi `pg_repack` PostgreSQL. Untuk informasi selengkapnya tentang `pg_repack`, lihat Menata ulang tabel di basis data PostgreSQL dengan kunci minimal.

Menggunakan replikasi logis untuk melakukan upgrade versi mayor untuk Aurora PostgreSQL

Menggunakan replikasi logis dan kloning cepat Aurora, Anda dapat melakukan upgrade versi mayor yang menggunakan versi basis data Aurora PostgreSQL saat ini sambil secara bertahap memigrasikan data yang berubah ke basis data versi mayor yang baru. Proses upgrade dengan waktu henti rendah ini disebut sebagai blue/green upgrade. Versi basis data saat ini disebut sebagai lingkungan “blue” dan versi basis data baru disebut sebagai lingkungan “green”.

Kloning cepat Aurora sepenuhnya memuat data yang ada dengan mengambil snapshot basis data sumber. Kloning cepat menggunakan copy-on-write protokol yang dibangun di atas lapisan penyimpanan Aurora, yang memungkinkan Anda membuat tiruan database dalam waktu singkat. Metode ini sangat efektif saat meng-upgrade ke basis data besar.

Replikasi logis di PostgreSQL melacak dan mentransfer perubahan data Anda dari instans awal ke instans baru yang berjalan secara paralel hingga Anda pindah ke versi PostgreSQL yang lebih baru. Replikasi logis menggunakan model "publish and subscribe". Untuk informasi selengkapnya tentang replikasi logis Aurora PostgreSQL, lihat Replikasi dengan Amazon Aurora PostgreSQL.

> **Tip**
>
> Anda dapat meminimalkan waktu henti yang diperlukan untuk upgrade versi mayor dengan menggunakan fitur Deployment Blue/Green Amazon RDS terkelola. Untuk informasi selengkapnya, lihat Menggunakan Deployment Blue/Green Amazon RDS untuk pembaruan basis data.

Topik

- Persyaratan
- Batasan
- Mengatur dan memeriksa nilai parameter
- Meng-upgrade Aurora PostgreSQL ke versi mayor yang baru
- Melakukan tugas pasca-upgrade

Persyaratan

Anda harus memenuhi persyaratan berikut untuk melakukan proses upgrade dengan waktu henti rendah ini:

- Anda harus memiliki izin rds_superuser.
- Klaster DB Aurora PostgreSQL yang ingin Anda upgrade harus menjalankan versi yang didukung yang dapat melakukan upgrade versi mayor menggunakan replikasi logis. Pastikan untuk menerapkan pembaruan dan patch versi minor apa pun ke klaster DB Anda. Fungsi `aurora_volume_logical_start_lsn` yang digunakan dalam teknik ini didukung dalam versi Aurora PostgreSQL berikut:
  - 15.2 dan versi 15 yang lebih tinggi
  - Versi 14.3 dan versi 14 yang lebih tinggi
  - Versi 13.6 dan versi 13 yang lebih tinggi
  - Versi 12.10 dan versi 12 yang lebih tinggi
  - Versi 11.15 dan versi 11 yang lebih tinggi
  - 10.20 dan versi 10 yang lebih tinggi

  Untuk informasi selengkapnya tentang fungsi `aurora_volume_logical_start_lsn`, lihat aurora_volume_logical_start_lsn.

- Semua tabel Anda harus memiliki kunci primer atau menyertakan kolom identitas PostgreSQL.
- Konfigurasikan grup keamanan untuk VPC Anda agar mengizinkan akses masuk dan keluar antara dua klaster DB Aurora PostgreSQL, baik yang lama maupun yang baru. Anda dapat memberikan akses ke rentang perutean antar domain tanpa kelas (CIDR) tertentu atau ke grup keamanan lain di VPC Anda atau VPC peer. (VPC peer membutuhkan koneksi peering VPC.)

> **Note**
>
> Untuk informasi terperinci tentang izin yang diperlukan untuk mengonfigurasi dan mengelola skenario replikasi logis yang berjalan, lihat Dokumentasi inti PostgreSQL.

Batasan

Saat Anda melakukan upgrade dengan waktu henti rendah pada klaster DB Aurora PostgreSQL Anda untuk meng-upgrade-nya ke versi mayor baru, Anda menggunakan fitur replikasi logis PostgreSQL native. Fitur ini memiliki kemampuan dan batasan yang sama dengan replikasi logis PostgreSQL. Untuk informasi selengkapnya, lihat Replikasi logis PostgreSQL.

- Perintah bahasa definisi data (DDL) tidak direplikasi.
- Replikasi tidak mendukung perubahan skema dalam basis data aktif. Skema dibuat kembali dalam bentuk aslinya selama proses kloning. Jika Anda mengubah skema setelah kloning, tetapi sebelum menyelesaikan upgrade, perubahan tersebut tidak akan tercermin dalam instans yang di-upgrade.
- Objek besar tidak direplikasi, tetapi Anda dapat menyimpan data dalam tabel normal.
- Replikasi hanya didukung oleh tabel, termasuk tabel yang dipartisi. Replikasi ke jenis relasi lain, seperti tampilan, tampilan terwujud, atau tabel asing, tidak didukung.
- Data urutan tidak direplikasi dan memerlukan pembaruan manual pasca-failover.

> **Note**
>
> Upgrade ini tidak mendukung auto-scripting. Anda harus melakukan semua langkah secara manual.

Mengatur dan memeriksa nilai parameter

Sebelum meng-upgrade, konfigurasikan instans penulis klaster DB Aurora PostgreSQL Anda untuk bertindak sebagai server penerbitan. Instans harus menggunakan grup parameter klaster DB kustom dengan pengaturan berikut:

- `rds.logical_replication` – Atur parameter ini ke 1. Parameter`rds.logical_replication` memiliki tujuan yang sama dengan parameter `wal_level` server PostgreSQL mandiri dan parameter lain yang mengontrol manajemen file write-ahead log.
- `max_replication_slots` – Atur parameter ini ke jumlah total langganan yang berencana Anda buat. Jika Anda menggunakan AWS DMS, atur parameter ini ke jumlah AWS DMS tugas yang Anda rencanakan untuk digunakan untuk pengambilan data yang diubah dari cluster DB ini.
- `max_wal_senders` – Atur ke jumlah koneksi konkuren, plus beberapa tambahan, untuk mengakomodasi tugas manajemen dan sesi baru. Jika Anda menggunakan AWS DMS, jumlah max_wal_senders harus sama dengan jumlah sesi bersamaan ditambah jumlah AWS DMS tugas yang mungkin bekerja pada waktu tertentu.
- `max_logical_replication_workers` – Atur ke jumlah pekerja replikasi logis dan pekerja sinkronisasi tabel yang Anda inginkan. Umumnya aman untuk mengatur jumlah pekerja replikasi ke nilai yang sama yang digunakan untuk max_wal_senders. Pekerja diambil dari kumpulan proses latar belakang (max_worker_processes) yang dialokasikan untuk server.
- `max_worker_processes` – Atur ke jumlah proses latar belakang untuk server. Jumlah ini harus cukup besar untuk mengalokasikan pekerja untuk replikasi, proses autovacuum, dan proses pemeliharaan lainnya yang mungkin terjadi secara bersamaan.

Saat Anda meng-upgrade ke versi Aurora PostgreSQL yang lebih baru, Anda perlu menduplikasi parameter apa pun yang Anda modifikasi di grup parameter versi sebelumnya. Parameter ini diterapkan ke versi yang di-upgrade. Anda dapat mengkueri tabel `pg_settings` untuk mendapatkan daftar pengaturan parameter sehingga Anda dapat membuatnya kembali di klaster DB Aurora PostgreSQL baru Anda.

Misalnya, untuk mendapatkan pengaturan untuk parameter replikasi, jalankan kueri berikut:

```
SELECT name, setting FROM pg_settings WHERE name in
('rds.logical_replication', 'max_replication_slots',
'max_wal_senders', 'max_logical_replication_workers',
'max_worker_processes');
```

Meng-upgrade Aurora PostgreSQL ke versi mayor yang baru

Untuk mempersiapkan penerbit (blue)

1. Dalam contoh berikut, instans penulis sumber (blue) adalah klaster DB Aurora PostgreSQL yang menjalankan PostgreSQL versi 11.15. Ini adalah simpul penerbitan dalam skenario replikasi kita. Untuk demonstrasi ini, instans penulis sumber kita menghosting tabel sampel yang menyimpan serangkaian nilai:

   ```
   CREATE TABLE my_table (a int PRIMARY KEY);
   INSERT INTO my_table VALUES (generate_series(1,100));
   ```

2. Untuk membuat penerbitan pada instans sumber, hubungkan ke simpul penulis instans dengan psql (CLI untuk PostgreSQL) atau dengan klien pilihan Anda). Masukkan perintah berikut di setiap basis data:

   ```
   CREATE PUBLICATION publication_name FOR ALL TABLES;
   ```

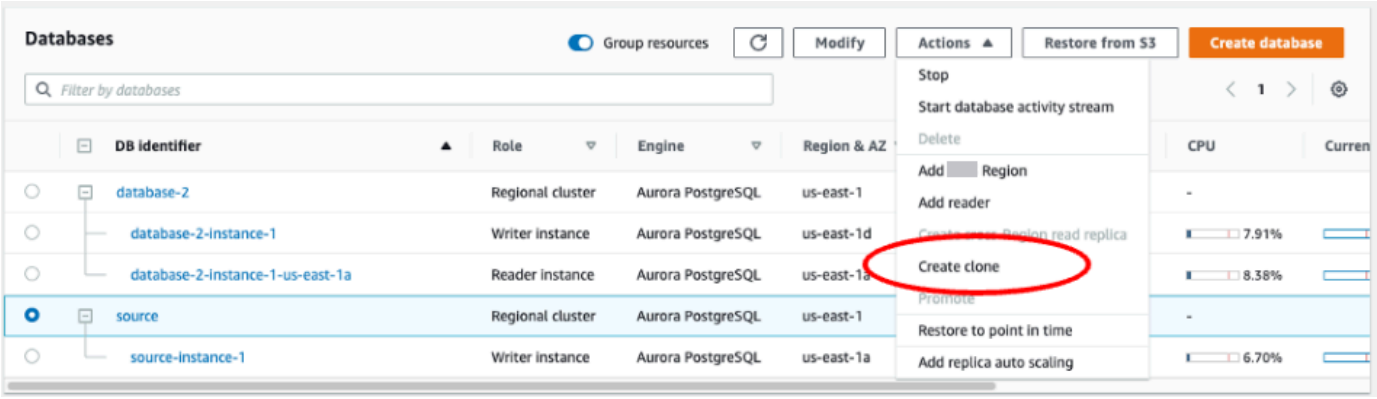
   Publication_name menentukan nama penerbitan.

3. Anda juga perlu membuat slot replikasi pada instans. Perintah berikut membuat slot replikasi dan memuat plug-in pendekodean logis `pgoutput`. Plug-in ini mengubah pembacaan konten dari write-ahead logging (WAL) ke protokol replikasi logis, dan memfilter data sesuai dengan spesifikasi penerbitan.

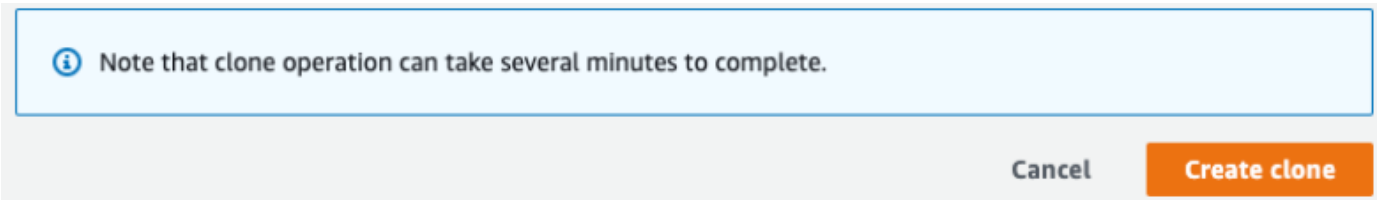
   ```
   SELECT pg_create_logical_replication_slot('replication_slot_name', 'pgoutput');
   ```

Untuk mengkloning penerbit

1. Gunakan Konsol Amazon RDS untuk membuat klon dari instans sumber. Sorot nama instans di Konsol Amazon RDS, lalu pilih Buat klona di menu Tindakan.



2. Berikan nama yang unik untuk instans. Sebagian besar pengaturan adalah default dari instans sumber. Ketika Anda telah membuat perubahan yang diperlukan untuk instans baru, pilih Buat klona.



3. Saat instans target dimulai, kolom Status simpul penulis akan menampilkan Membuat di kolom Status. Saat instans siap, statusnya berubah menjadi Tersedia.

Untuk mempersiapkan klon untuk upgrade

1. Klon adalah instans 'green' dalam model deployment. Ini adalah host dari simpul langganan replikasi. Ketika simpul tersedia, hubungkan dengan psql dan kueri simpul penulis baru untuk mendapatkan nomor urutan log (LSN). LSN mengidentifikasi awal catatan di stream WAL.

```
SELECT aurora_volume_logical_start_lsn();
```

2. Dalam respons dari kueri, Anda akan menemukan nomor LSN. Anda memerlukan nomor ini nanti dalam prosesnya, jadi catatlah.

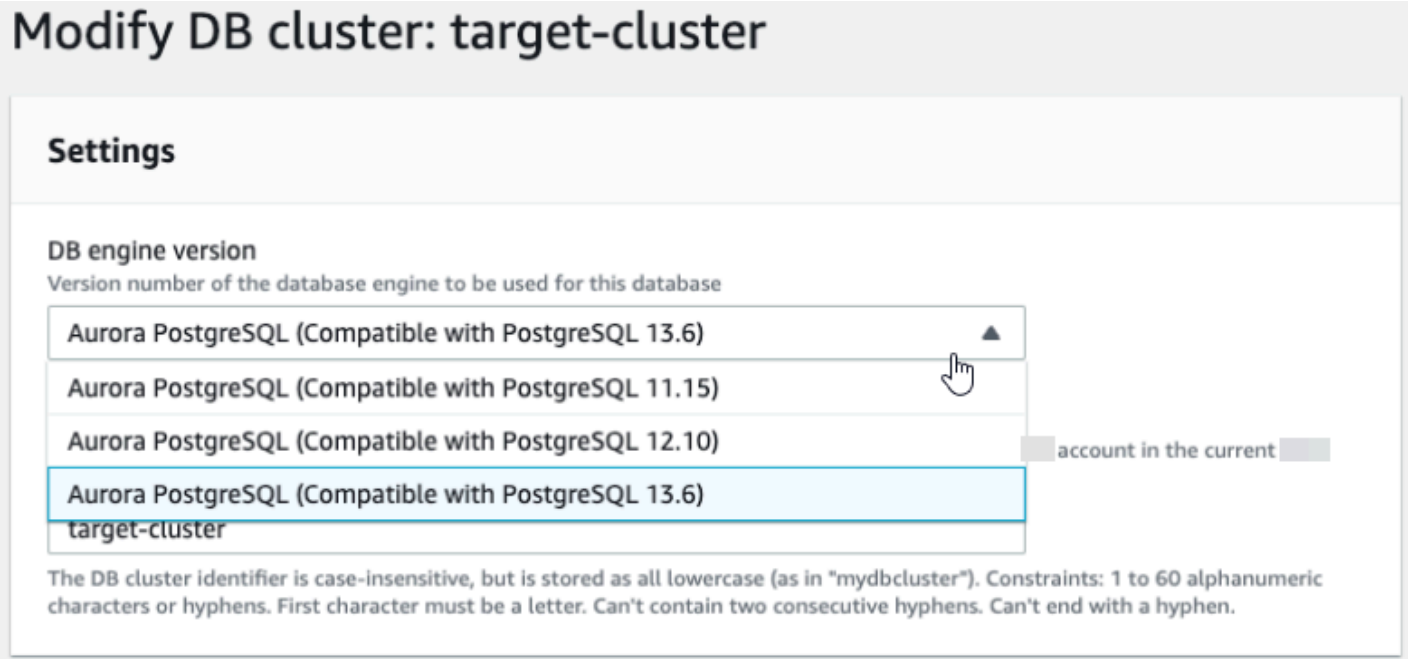
```
postgres=> SELECT aurora_volume_logical_start_lsn();
aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

3. Sebelum meng-upgrade klon, hapus slot replikasi klon.

```
SELECT pg_drop_replication_slot('replication_slot_name');
```

Untuk meng-upgrade klaster ke versi mayor yang baru

- Setelah mengkloning simpul penyedia, gunakan Konsol Amazon RDS untuk memulai upgrade versi mayor pada simpul langganan. Sorot nama instans di konsol RDS, dan pilih tombol Modifikasi. Pilih versi yang diperbarui dan grup parameter Anda yang diperbarui, lalu segera terapkan pengaturan untuk meng-upgrade instans target.



- Anda juga dapat menggunakan CLI untuk melakukan upgrade:

```
aws rds modify-db-cluster –db-cluster-identifier $TARGET_Aurora_ID –engine-version
 13.6 –allow-major-version-upgrade –apply-immediately
```

Untuk mempersiapkan pelanggan (hijau)

1. Saat klon tersedia setelah upgrade, hubungkan dengan psql dan tentukan langganan. Untuk melakukannya, Anda perlu menentukan opsi berikut dalam perintah `CREATE SUBSCRIPTION`:

   - `subscription_name` – Nama langganan.
   - `admin_user_name` – Nama pengguna administratif dengan izin rds_superuser.
   - `admin_user_password` – Kata sandi yang terkait dengan pengguna administratif.

- `source_instance_URL` – URL instans server penerbitan.
- `database` – Basis data yang akan terhubung dengan server langganan.
- `publication_name` – Nama server penerbitan.
- `replication_slot_name` – Nama slot replikasi.

```
CREATE SUBSCRIPTION subscription_name
CONNECTION 'postgres://admin_user_name:admin_user_password@source_instance_URL/
database' PUBLICATION publication_name
WITH (copy_data = false, create_slot = false, enabled = false, connect = true,
 slot_name = 'replication_slot_name');
```

2. Setelah membuat langganan, jalankan kueri terhadap tampilan pg_replication_origin untuk mengambil nilai roname, yang merupakan pengidentifikasi asal replikasi. Setiap instans memiliki satu `roname`:

   ```
   SELECT * FROM pg_replication_origin;
   ```

   Sebagai contoh:

   ```
   postgres=>
   SELECT * FROM pg_replication_origin;

   roident | roname
   ---------+----------
   1 | pg_24586
   ```

3. Berikan LSN yang Anda catat dari kueri sebelumnya dari simpul penerbitan dan `roname` yang dihasilkan dari simpul langganan [INSTANCE] dalam perintah. Perintah ini menggunakan fungsi `pg_replication_origin_advance` untuk menentukan titik awal dalam urutan log untuk replikasi.

   ```
   SELECT pg_replication_origin_advance('roname', 'log_sequence_number');
   ```

   `roname` adalah pengidentifikasi yang dihasilkan oleh tampilan pg_replication_origin.

   `log_sequence_number` adalah nilai yang dihasilkan oleh kueri sebelumnya terhadap fungsi `aurora_volume_logical_start_lsn`.

4. Kemudian, gunakan klausa `ALTER SUBSCRIPTION... ENABLE` untuk mengaktifkan replikasi logis.

```
ALTER SUBSCRIPTION subscription_name ENABLE;
```

5. Pada tahap ini, Anda dapat mengonfirmasi bahwa replikasi berfungsi. Tambahkan nilai ke instans penerbitan, lalu konfirmasikan bahwa nilainya direplikasi ke simpul langganan.

   Kemudian, gunakan perintah berikut untuk memantau lag replikasi pada simpul penerbitan:

```
SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
 pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
 'logical';
```

   Sebagai contoh:

```
postgres=> SELECT now() AS CURRENT_TIME, slot_name, active, active_pid,
 pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS diff_size, pg_wal_lsn_diff(pg_current_wal_lsn(),
 confirmed_flush_lsn) AS diff_bytes FROM pg_replication_slots WHERE slot_type =
 'logical';

current_time                   | slot_name             | active | active_pid |
 diff_size | diff_bytes
-------------------------------+-----------------------+--------+------------
+-----------+------------
2022-04-13 15:11:00.243401+00  | replication_slot_name | t      | 21854      | 136
 bytes | 136
(1 row)
```

   Anda dapat memantau lag replikasi menggunakan nilai `diff_size` dan `diff_bytes`. Saat nilai ini mencapai 0, replika telah menyamai instans DB sumber.

Melakukan tugas pasca-upgrade

Saat upgrade selesai, status instans ditampilkan sebagai Tersedia di kolom Status dalam dasbor konsol. Pada instans baru, kami menyarankan agar Anda melakukan hal berikut:

- Arahkan ulang aplikasi Anda untuk mengarah ke simpul penulis.
- Tambahkan simpul pembaca untuk mengelola caseload dan memberikan ketersediaan tinggi jika terjadi masalah dengan simpul penulis.
- Klaster DB Aurora PostgreSQL terkadang memerlukan pembaruan sistem operasi. Pembaruan ini mungkin menyertakan pustaka glibc versi yang lebih baru. Selama pembaruan tersebut, sebaiknya Anda mengikuti pedoman seperti yang dijelaskan dalam Kolasi yang didukung di Aurora PostgreSQL.
- Perbarui izin pengguna pada instans baru untuk memastikan akses.

Setelah menguji aplikasi dan data pada instans baru, kami sarankan agar Anda membuat cadangan akhir instans awal Anda sebelum menghapusnya. Untuk informasi selengkapnya tentang penggunaan replikasi logis pada host Aurora, lihat Menyiapkan replikasi logis untuk klaster DB Aurora PostgreSQL Anda.

Pemecahan masalah penyimpanan

Jika jumlah memori kerja yang diperlukan untuk operasi pengurutan atau pembuatan indeks melampaui jumlah yang dialokasikan oleh parameter `work_mem`, Aurora PostgreSQL akan menulis kelebihan data ke file disk sementara. Ketika menulis data, Aurora PostgreSQL menggunakan ruang penyimpanan yang sama seperti yang digunakan untuk menyimpan log kesalahan dan pesan, yaitu penyimpanan lokal. Setiap instans di klaster DB Aurora PostgreSQL Anda memiliki jumlah penyimpanan lokal yang tersedia. Jumlah penyimpanan didasarkan pada kelas instans DB-nya. Untuk meningkatkan jumlah penyimpanan lokal, Anda perlu mengubah instans agar menggunakan kelas instans DB yang lebih besar. Untuk spesifikasi kelas instans DB, lihat Spesifikasi perangkat keras kelas instans DB untuk Aurora.

Anda dapat memantau ruang penyimpanan lokal klaster DB Aurora PostgreSQL Anda dengan memantau metrik `FreeLocalStorage` Amazon CloudWatch. Metrik ini melaporkan jumlah penyimpanan yang tersedia untuk setiap instans DB di klaster DB Aurora untuk tabel dan log sementara. Untuk informasi selengkapnya, lihat Memantau metrik Amazon Aurora dengan Amazon CloudWatch.

Operasi pengurutan, pengindeksan, dan pengelompokan dimulai dalam memori kerja, tetapi sering harus dialihkan ke penyimpanan lokal. Jika klaster DB Aurora PostgreSQL Anda kehabisan penyimpanan lokal karena jenis operasi ini, Anda dapat menyelesaikan masalahnya dengan mengambil salah satu tindakan berikut.

- Tingkatkan jumlah memori kerja. Tindakan ini mengurangi kebutuhan untuk menggunakan penyimpanan lokal. Secara default, PostgreSQL mengalokasikan 4 MB untuk setiap operasi pengurutan, pengelompokan, dan pengindeksan. Untuk memeriksa nilai memori kerja saat ini untuk instans penulis klaster DB Aurora PostgreSQL Anda, hubungkan ke instans menggunakan `psql` dan jalankan perintah berikut.

```
postgres=> SHOW work_mem;
work_mem
----------
 4MB
(1 row)
```

  Anda dapat meningkatkan memori kerja di tingkat sesi sebelum operasi pengurutan, pengelompokan, dan lainnya, sebagai berikut.

```
SET work_mem TO '1 GB';
```

  Untuk informasi selengkapnya tentang memori kerja, lihat Resource Consumption dalam dokumentasi PostgreSQL.
- Ubah periode retensi log sehingga log disimpan untuk jangka waktu yang lebih pendek. Untuk mempelajari caranya, lihat File log basis data Aurora PostgreSQL.

Untuk klaster DB Aurora PostgreSQL yang lebih besar dari 40 TB, jangan gunakan kelas instans db.t2, db.t3, atau db.t4g. Kami menyarankan penggunaan kelas instans DB T hanya untuk server pengembangan dan pengujian, atau server non-produksi lainnya. Untuk informasi selengkapnya, lihat Jenis kelas instans DB.

Replikasi dengan Amazon Aurora PostgreSQL

Berikut ini, Anda dapat menemukan informasi tentang replikasi dengan Amazon Aurora PostgreSQL, termasuk cara memantaunya.

Topik

- Menggunakan Aurora Replica
- Meningkatkan ketersediaan baca Aurora Replica
- Memantau replikasi Aurora PostgreSQL

- Menggunakan replikasi logis PostgreSQL dengan Aurora

Menggunakan Aurora Replica

Aurora Replica adalah titik akhir independen dalam klaster Aurora DB, paling baik digunakan untuk menskalakan operasi baca dan meningkatkan ketersediaan. Cluster Aurora DB dapat mencakup hingga 15 Replika Aurora yang terletak di seluruh Zona Ketersediaan Wilayah cluster Aurora DB. AWS

Volume klaster DB terdiri atas beberapa salinan data untuk klaster DB. Namun, data dalam volume klaster direpresentasikan sebagai satu volume logis untuk instans DB penulis primer dan Aurora Replica di klaster DB. Untuk informasi selengkapnya tentang Aurora Replica, lihat Replika Aurora.

Aurora Replica sangat cocok untuk penskalaan baca karena ditujukan sepenuhnya untuk membaca operasi pada volume klaster Anda. Instans DB penulis mengelola operasi tulis. Volume klaster dibagikan di semua instans di klaster DB Aurora PostgreSQL Anda. Dengan demikian, tidak diperlukan kerja tambahan untuk mereplikasi salinan data untuk setiap Aurora Replica.

Dengan Aurora PostgreSQL, saat Aurora Replica dihapus, titik akhir instansnya akan segera dibuang, dan Aurora Replica akan dibuang dari titik akhir pembaca. Jika ada pernyataan yang dijalankan dari Aurora Replica yang sudah dihapus, ada tiga menit masa tenggang. Pernyataan yang ada dapat diselesaikan dengan baik selama masa tenggang. Setelah masa tenggang berakhir, Aurora Replica akan dimatikan dan dihapus.

Cluster Aurora PostgreSQL DB mendukung Aurora Replicas di berbagai Wilayah, menggunakan database global Aurora. AWS Untuk informasi selengkapnya, lihat Menggunakan basis data global Amazon Aurora.

> **Note**
>
> Dengan fitur ketersediaan baca yang ditingkatkan, jika Anda ingin melakukan boot ulang Aurora Replica di klaster DB, Anda harus melakukannya secara manual. Untuk klaster DB yang dibuat sebelum fitur ini melakukan boot ulang, instans DB penulis secara otomatis melakukan boot ulang Aurora Replica. Boot ulang otomatis menetapkan kembali titik entri untuk memastikan konsistensi pembacaan/penulisan di seluruh klaster DB.

Meningkatkan ketersediaan baca Aurora Replica

Aurora PostgreSQL meningkatkan ketersediaan baca di klaster DB dengan terus melayani permintaan baca ketika instans DB penulis dimulai ulang atau ketika Aurora Replica tidak dapat mengikuti lalu lintas tulis.

Fitur ketersediaan baca tersedia secara default pada versi Aurora PostgreSQL berikut:

- Versi 15.2 dan versi 15 yang lebih tinggi
- Versi 14.7 dan versi 14 yang lebih tinggi
- Versi 13.10 dan versi 13 yang lebih tinggi
- Versi 12.14 dan versi 12 yang lebih tinggi

Guna menggunakan fitur ketersediaan baca untuk klaster DB yang dibuat pada salah satu versi ini sebelum peluncuran ini, mulai ulang instans penulis klaster DB.

Saat Anda mengubah parameter statis klaster DB Aurora PostgreSQL, Anda harus memulai ulang instans penulis sehingga perubahan parameter berlaku. Misalnya, Anda harus memulai ulang instans penulis ketika Anda menetapkan nilai `shared_buffers`. Dengan peningkatan ketersediaan Aurora Replica, klaster DB mempertahankan ketersediaan baca selama memulai ulang ini, yang mengurangi dampak perubahan pada instans penulis. Instans pembaca tidak memulai ulang dan terus menanggapi permintaan baca. Untuk menerapkan perubahan parameter statis, mulai ulang setiap individu instans pembaca.

Aurora Replica klaster DB Aurora PostgreSQL dapat pulih dari kesalahan replikasi seperti mulai ulang penulis, failover, replikasi lambat, dan masalah jaringan dengan cepat memulihkan ke status basis data dalam memori setelah terhubung kembali dengan penulis. Pendekatan ini memungkinkan instans Aurora Replica untuk mencapai konsistensi dengan pembaruan penyimpanan terbaru saat basis data klien masih tersedia.

Transaksi yang sedang berlangsung yang bertentangan dengan pemulihan replikasi mungkin menerima kesalahan, tetapi klien dapat mencoba kembali transaksi ini, setelah pembaca mengejar ketinggalan dengan penulis.

Memantau Aurora Replica

Anda dapat memantau Aurora Replica saat pulih dari pemutusan penulis. Gunakan metrik di bawah ini untuk memeriksa informasi terbaru tentang instans pembaca dan untuk melacak transaksi hanya-baca dalam proses.

- `aurora_replica_status`Fungsi ini diperbarui untuk mengembalikan sebagian besar up-to-date informasi untuk instance pembaca ketika masih terhubung. Stempel waktu pembaruan terakhir di `aurora_replica_status` selalu kosong untuk baris yang sesuai dengan instans DB tempat kueri dijalankan. Ini menunjukkan bahwa instans pembaca memiliki data terbaru.
- Ketika replika Aurora terputus dari instans penulis dan menghubungkan kembali, peristiwa basis data berikut dipancarkan:

  ```
  Read replica has been disconnected from the writer instance and
  reconnected.
  ```

- Ketika kueri hanya-baca dibatalkan karena konflik pemulihan, Anda mungkin melihat pesan kesalahan berikut di log kesalahan basis data:

  ```
  Canceling statement due to conflict with recovery.
  ```

Batasan

Batasan berikut berlaku untuk Aurora Replica dengan ketersediaan yang ditingkatkan:

- Replika Aurora DB Global di sekunder Wilayah AWS tidak didukung.
- Aurora Replica tidak mendukung pemulihan replikasi online jika sudah dalam proses dan akan dimulai ulang.
- Aurora Replica akan dimulai ulang ketika instans DB Anda mendekati wraparound ID transaksi. Untuk informasi selengkapnya tentang penyelesaian ID transaksi, lihat Mencegah Kegagalan Penyelesaian ID Transaksi.
- Aurora Replica dapat dimulai kembali ketika proses replikasi diblokir dalam keadaan tertentu.

Memantau replikasi Aurora PostgreSQL

Penskalaan baca dan ketersediaan yang tinggi tergantung dari waktu lag minimal. Anda dapat memantau seberapa jauh Aurora Replica tertinggal dari instance DB penulis dari cluster DB Aurora PostgreSQL Anda dengan memantau metrik Amazon. CloudWatch `ReplicaLag` Karena Aurora Replicas membaca dari volume klaster yang sama dengan instans DB penulis, metrik `ReplicaLag` memiliki makna berbeda untuk klaster DB Aurora PostgreSQL. Metrik `ReplicaLag` untuk Replika Aurora menunjukkan lag untuk cache halaman Replika Aurora dibandingkan dengan yang ada di instans DB penulis.

Untuk informasi selengkapnya tentang pemantauan instans dan CloudWatch metrik RDS, lihat. Memantau metrik di klaster Amazon Aurora

Menggunakan replikasi logis PostgreSQL dengan Aurora

Dengan menggunakan fitur replikasi logis PostgreSQL dengan klaster DB Aurora PostgreSQL, Anda dapat mereplikasi dan menyinkronkan tabel individual daripada seluruh instans basis data. Replikasi logis menggunakan model penerbit dan berlangganan untuk mereplikasi perubahan dari sumber ke satu atau lebih penerima. Replikasi ini bekerja dengan menggunakan catatan perubahan dari log write-ahead (WAL) PostgreSQL. Sumber, atau penerbit, mengirimkan data WAL untuk tabel yang ditentukan ke satu atau lebih penerima (pelanggan), sehingga mereplikasi perubahan dan menjaga tabel pelanggan disinkronkan dengan tabel penerbit. Kumpulan perubahan dari penerbit diidentifikasi menggunakan publikasi. Pelanggan mendapatkan perubahan dengan membuat langganan yang mendefinisikan koneksi ke basis data penerbit dan publikasinya. Slot replikasi adalah mekanisme yang digunakan dalam skema ini untuk melacak progres langganan.

Untuk klaster DB Aurora PostgreSQL, catatan WAL disimpan di penyimpanan Aurora. Klaster DB Aurora PostgreSQL yang bertindak sebagai penerbit dalam skenario replikasi logis membaca data WAL dari penyimpanan Aurora, mendekodekan, dan mengirimkannya ke pelanggan sehingga perubahan dapat diterapkan ke tabel pada instans itu. Penerbit menggunakan decoder logis untuk memecahkan kode data untuk digunakan oleh pelanggan. Secara default, klaster DB Aurora PostgreSQL menggunakan plugin `pgoutput` PostgreSQL asli saat mengirim data. Decoder logis lainnya tersedia. Misalnya, Aurora PostgreSQL juga mendukung plugin `wal2json` yang mengonversi data WAL ke JSON.

Pada Aurora PostgreSQL versi 14.5, 13.8, 12.12, dan 11.17, Aurora PostgreSQL menambah proses replikasi logis PostgreSQL dengan cache write-through untuk meningkatkan kinerja. Log transaksi WAL di-cache secara lokal, dalam buffer, untuk mengurangi jumlah disk I/O, yaitu membaca dari penyimpanan Aurora selama decoding logis. Cache write-through digunakan secara default setiap kali Anda menggunakan replikasi logis untuk klaster DB Aurora PostgreSQL Anda. Aurora menyediakan beberapa fungsi yang dapat Anda gunakan untuk mengelola cache. Untuk informasi selengkapnya, lihat Mengelola cache write-through replikasi logis Aurora PostgreSQL.

Replikasi logis didukung oleh semua versi PostgreSQL Aurora yang tersedia saat ini. Untuk informasi selengkapnya, lihat Pembaruan Amazon Aurora PostgreSQL di Catatan Rilis untuk Aurora PostgreSQL.

> **Note**
>
> Selain fitur replikasi logis PostgreSQL asli yang diperkenalkan di PostgreSQL 10, Aurora PostgreSQL juga mendukung ekstensi `pglogical`. Untuk informasi selengkapnya, lihat Menggunakan pglogical untuk menyinkronkan data di seluruh instans.

Untuk informasi selengkapnya tentang fitur replikasi logis PostgreSQL, lihat Replikasi logis dan Konsep decoding logis dalam dokumentasi PostgreSQL.

Dalam topik berikut, Anda dapat menemukan informasi tentang cara mengatur replikasi logis antara klaster DB Aurora PostgreSQL.

Topik

- Menyiapkan replikasi logis untuk klaster DB Aurora PostgreSQL Anda
- Menggunakan replikasi logis
- Mengelola cache write-through replikasi logis Aurora PostgreSQL
- Mengelola slot logis untuk Aurora PostgreSQL
- Contoh: Menggunakan replikasi logis dengan klaster DB Aurora PostgreSQL
- Contoh: Replikasi logis menggunakan Aurora PostgreSQL dan AWS Database Migration Service

Menyiapkan replikasi logis untuk klaster DB Aurora PostgreSQL Anda

Menyiapkan replikasi logis membutuhkan `rds_superuser` hak istimewa. Klaster DB Aurora PostgreSQL Anda harus dikonfigurasi untuk menggunakan grup parameter klaster DB kustom sehingga Anda dapat mengatur parameter yang diperlukan sebagaimana dijelaskan dalam prosedur berikut. Untuk informasi selengkapnya, lihat Bekerja dengan grup parameter klaster DB.

Untuk menyiapkan replikasi logis PostgreSQL untuk klaster DB Aurora PostgreSQL Anda

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih klaster DB Aurora PostgreSQL Anda.
3. Buka tab Konfigurasi. Di antara detail Instans, temukan tautan Grup parameter dengan Grup parameter klaster DB untuk Jenis.

4. Pilih tautan untuk membuka parameter khusus yang terkait dengan klaster DB Aurora PostgreSQL Anda.
5. Di bidang pencarian Parameter, ketik `rds` untuk menemukan parameter `rds.logical_replication`. Nilai default untuk parameter ini adalah `0`, artinya fitur dinonaktifkan secara default.
6. Pilih Edit parameter untuk mengakses nilai properti, lalu pilih 1 dari pemilih untuk mengaktifkan fitur tersebut. Tergantung penggunaan yang Anda harapkan, Anda mungkin juga perlu mengubah pengaturan parameter berikut. Namun, dalam banyak kasus, nilai default sudah cukup.

   - `max_replication_slots` – Tetapkan parameter ini ke nilai yang setidaknya sama dengan jumlah total publikasi dan langganan replikasi logis yang Anda rencanakan. Jika Anda menggunakan AWS DMS, parameter ini harus sama setidaknya dengan tugas pengambilan data perubahan yang direncanakan dari klaster, ditambah publikasi dan langganan replikasi logis.
   - `max_wal_senders` dan `max_logical_replication_workers` – Tetapkan parameter ini ke nilai yang setidaknya sama dengan jumlah slot replikasi logis yang ingin Anda aktifkan, atau jumlah tugas AWS DMS aktif untuk pengambilan data perubahan. Membiarkan slot replikasi logis tidak aktif mencegah vakum menghilangkan tuple usang dari tabel, jadi kami sarankan Anda memantau slot replikasi dan menghapus slot yang tidak aktif sesuai kebutuhan.
   - `max_worker_processes` – Tetapkan parameter ini ke nilai yang setidaknya sama dengan total nilai `max_logical_replication_workers`, `autovacuum_max_workers`, dan `max_parallel_workers`. Pada kelas instans DB kecil, proses pekerja latar belakang dapat memengaruhi beban kerja aplikasi, jadi pantau kinerja basis data jika Anda menetapkan `max_worker_processes` lebih tinggi dari nilai default. (Nilai default adalah hasil dari `GREATEST(${DBInstanceVCPU*2},8}`, yang berarti bahwa, secara default, ini adalah 8 atau dua kali setara CPU dari kelas instans DB, mana pun yang lebih besar).

   > **Note**
   >
   > Anda dapat mengubah nilai parameter dalam grup parameter DB buatan pelanggan; Anda tidak dapat mengubah nilai parameter dalam grup parameter DB default.

7. Pilih Save perubahan.
8. Boot ulang instans penulis klaster DB Aurora PostgreSQL Anda sehingga perubahan diterapkan. Di konsol Amazon RDS, pilih instans DB utama klaster dan pilih Boot Ulang dari menu Tindakan.

9. Ketika instans tersedia, Anda dapat memverifikasi bahwa replikasi logis diaktifkan sebagai berikut.

   a. Gunakan `psql` untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL.

   ```
   psql --host=your-db-cluster-instance-1.aws-region.rds.amazonaws.com --port=5432
    --username=postgres --password --dbname=labdb
   ```

   b. Verifikasi bahwa replikasi logis telah diaktifkan dengan menggunakan perintah berikut.

   ```
   labdb=> SHOW rds.logical_replication;
    rds.logical_replication
   -------------------------
    on
   (1 row)
   ```

   c. Verifikasi bahwa `wal_level` diatur ke `logical`.

   ```
   labdb=> SHOW wal_level;
     wal_level
   -----------
    logical
   (1 row)
   ```

Untuk contoh menggunakan replikasi logis untuk menjaga tabel basis data disinkronkan dengan perubahan dari sumber klaster DB Aurora PostgreSQL, lihat Contoh: Menggunakan replikasi logis dengan klaster DB Aurora PostgreSQL.

Menggunakan replikasi logis

Setelah Anda menyelesaikan tugas replikasi, hentikan proses replikasi, letakkan slot replikasi, dan nonaktifkan replikasi logis. Sebelum meletakkan slot, pastikan bahwa slot tidak diperlukan lagi. Slot replikasi aktif tidak dapat diletakkan.

Untuk menonaktifkan replikasi logis

1. Letakkan semua slot replikasi.

   Untuk meletakkan semua slot replikasi, hubungkan ke penerbit dan jalankan perintah SQL berikut.

```
SELECT pg_drop_replication_slot(slot_name)
  FROM pg_replication_slots
 WHERE slot_name IN (SELECT slot_name FROM pg_replication_slots);
```

Slot replikasi tidak dapat aktif saat Anda menjalankan perintah ini.

2. Ubah grup parameter klaster DB kustom yang terkait dengan penerbit seperti yang dijelaskan dalamMenyiapkan replikasi logis untuk klaster DB Aurora PostgreSQL Anda, tetapi atur parameter `rds.logical_replication` ke 0.

   Untuk informasi selengkapnya tentang grup parameter kustom, lihat Mengubah parameter dalam grup parameter klaster DB.

3. Mulai ulang klaster DB Aurora PostgreSQL penerbit agar perubahan parameter `rds.logical_replication` dapat diterapkan.

Mengelola cache write-through replikasi logis Aurora PostgreSQL

Secara default, Aurora PostgreSQL versi 14.5, 13.8, 12.12, dan 11.17 serta yang lebih tinggi menggunakan cache write-through untuk meningkatkan kinerja replikasi logis. Tanpa cache write-through, Aurora PostgreSQL menggunakan lapisan penyimpanan Aurora dalam implementasi proses replikasi logis PostgreSQL asli. Aurora PostgreSQL melakukannya dengan menulis data WAL ke penyimpanan dan kemudian membaca data kembali dari penyimpanan untuk memecahkan kode dan mengirim (mereplikasi) ke targetnya (pelanggan). Hal ini dapat mengakibatkan kemacetan selama replikasi logis untuk klaster DB Aurora PostgreSQL.

Cache write-through mengurangi kebutuhan untuk menggunakan lapisan penyimpanan Aurora. Aurora PostgreSQL menggunakan buffer untuk menyimpan aliran WAL logis daripada selalu menulis dan membaca dari lapisan penyimpanan Aurora, sehingga dapat digunakan selama proses replikasi, daripada selalu menarik dari disk. Buffer ini adalah cache asli PostgreSQL yang digunakan oleh replikasi logis, diidentifikasi dalam parameter klaster DB Aurora PostgreSQL sebagai `rds.logical_wal_cache`. Secara default, cache ini menggunakan 1/32 dari pengaturan cache buffer klaster DB Aurora PostgreSQL (`shared_buffers`), tetapi tidak kurang dari 64 kB atau lebih dari ukuran satu segmen WAL, biasanya 16 MB.

Saat Anda menggunakan replikasi logis dengan klaster DB Aurora PostgreSQL Anda (untuk versi yang mendukung cache write-through), Anda dapat memantau rasio hit cache untuk melihat seberapa baik kerjanya untuk kasus penggunaan. Untuk melakukannya, hubungkan ke instans

tulis klaster DB Aurora PostgreSQL Anda menggunakan `psql`, lalu gunakan fungsi Aurora `aurora_stat_logical_wal_cache` seperti yang ditunjukkan pada contoh berikut.

```
SELECT * FROM aurora_stat_logical_wal_cache();
```

Fungsi mengembalikan output seperti berikut ini.

```
name       | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
-----------+------------+-----------+------------+-----------+----------+--------------
test_slot1 | 79183      | 24        | 0          | 24        | 100.00%  | 2022-08-05
 17:39...
test_slot2 |            | 1         | 0          |  1        | 100.00%  | 2022-08-05
 17:34...
(2 rows)
```

Nilai-nilai `last_reset_timestamp` telah dipersingkat untuk keterbacaan. Untuk informasi selengkapnya tentang fungsi ini, lihat aurora_stat_logical_wal_cache.

Aurora PostgreSQL menyediakan dua fungsi berikut untuk memantau cache write-through.

- Fungsi `aurora_stat_logical_wal_cache` – Untuk dokumentasi referensi, lihat aurora_stat_logical_wal_cache.
- Fungsi `aurora_stat_reset_wal_cache` – Untuk dokumentasi referensi, lihat aurora_stat_reset_wal_cache.

Jika Anda menemukan bahwa ukuran cache WAL yang disesuaikan secara otomatis tidak cukup untuk beban kerja, Anda dapat mengubah nilai `rds.logical_wal_cache` secara manual, dengan memodifikasi parameter dalam grup parameter klaster DB kustom Anda. Perhatikan bahwa setiap nilai positif kurang dari 32 kB diperlakukan sebagai 32 kB. Untuk informasi selengkapnya tentang `wal_buffers`, lihat Log Write Ahead di dokumentasi PostgreSQL.

Mengelola slot logis untuk Aurora PostgreSQL

Aktivitas streaming ditangkap dalam tampilan `pg_replication_origin_status`. Untuk melihat isi tampilan ini, Anda dapat menggunakan fungsi `pg_show_replication_origin_status()`, seperti yang ditunjukkan berikut:

```
SELECT * FROM pg_show_replication_origin_status();
```

Anda bisa mendapatkan daftar slot logis Anda dengan menggunakan kueri SQL berikut.

```
SELECT * FROM pg_replication_slots;
```

Untuk meletakkan slot logis, gunakan `pg_drop_replication_slot` dengan nama slot, seperti yang ditunjukkan dalam perintah berikut.

```
SELECT pg_drop_replication_slot('test_slot');
```

Contoh: Menggunakan replikasi logis dengan klaster DB Aurora PostgreSQL

Prosedur berikut menunjukkan cara memulai replikasi logis antara dua klaster DB Aurora PostgreSQL. Baik penerbit dan pelanggan harus dikonfigurasi untuk replikasi logis seperti yang dijelaskan dalam Menyiapkan replikasi logis untuk klaster DB Aurora PostgreSQL Anda.

Klaster DB Aurora PostgreSQL yang merupakan penerbit yang ditunjuk juga harus mengizinkan akses ke slot replikasi. Untuk melakukannya, ubah grup keamanan yang terkait dengan cloud publik virtual (VPC) klaster DB Aurora PostgreSQL berdasarkan layanan Amazon VPC. Izinkan akses masuk dengan menambahkan grup keamanan yang terkait dengan VPC pelanggan ke grup keamanan penerbit. Untuk informasi selengkapnya, lihat Mengontrol lalu lintas ke sumber daya menggunakan grup keamanan di Panduan Pengguna Amazon VPC.

Dengan langkah-langkah awal ini selesai, Anda dapat menggunakan perintah PostgreSQL `CREATE PUBLICATION` pada penerbit dan `CREATE SUBSCRIPTION` pada pelanggan, sebagaimana dijelaskan dalam prosedur berikut.

Untuk memulai replikasi logis antara dua klaster DB Aurora PostgreSQL.

Langkah-langkah ini mengasumsikan bahwa klaster DB Aurora PostgreSQL Anda memiliki instans penulis dengan basis data untuk membuat tabel contoh.

1. Di klaster DB Aurora PostgreSQL penerbit

   a. Buat tabel menggunakan pernyataan SQL berikut.

   ```
   CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
   ```

   b. Masukkan data ke dalam basis data penerbit dengan menggunakan pernyataan SQL berikut.

```
INSERT INTO LogicalReplicationTest VALUES (generate_series(1,10000));
```

c. Verifikasi bahwa data ada dalam tabel dengan menggunakan pernyataan SQL berikut.

```
SELECT count(*) FROM LogicalReplicationTest;
```

d. Buat publikasi untuk tabel ini dengan menggunakan pernyataan `CREATE PUBLICATION`, sebagai berikut.

```
CREATE PUBLICATION testpub FOR TABLE LogicalReplicationTest;
```

2. Di klaster DB Aurora PostgreSQL pelanggan

a. Buat tabel `LogicalReplicationTest` yang sama pada pelanggan yang Anda buat di penerbit, sebagai berikut.

```
CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
```

b. Verifikasi bahwa tabel ini kosong.

```
SELECT count(*) FROM LogicalReplicationTest;
```

c. Buat langganan untuk mendapatkan perubahan dari penerbit. Anda perlu menggunakan detail berikut tentang klaster DB Aurora PostgreSQL penerbit.

- host – Instans DB penulis klaster DB Aurora PostgreSQL penerbit.
- port – Port tempat instans DB penulis mendengarkan. Nilai default untuk PostgreSQL adalah 5432.
- dbname – Nama basis data.

```
CREATE SUBSCRIPTION testsub CONNECTION
   'host=publisher-cluster-writer-endpoint port=5432 dbname=db-name user=user
 password=password'
   PUBLICATION testpub;
```

> **Note**
>
> Tentukan kata sandi selain prompt yang ditampilkan di sini sebagai praktik keamanan terbaik.

Setelah langganan dibuat, slot replikasi logis dibuat di penerbit.

d. Untuk memverifikasi contoh ini di mana data awal direplikasi pada pelanggan, gunakan pernyataan SQL berikut pada basis data pelanggan.

```
SELECT count(*) FROM LogicalReplicationTest;
```

Setiap perubahan lebih lanjut pada penerbit direplikasi ke pelanggan.

Replikasi logis mempengaruhi kinerja. Kami menyarankan Anda menonaktifkan replikasi logis setelah tugas replikasi Anda selesai.

Contoh: Replikasi logis menggunakan Aurora PostgreSQL dan AWS Database Migration Service

Anda dapat menggunakan AWS Database Migration Service (AWS DMS) untuk mereplikasi basis data atau bagian dari basis data. Gunakan AWS DMS untuk memigrasikan data Anda dari basis data Aurora PostgreSQL ke basis data sumber terbuka atau komersial lain. Untuk informasi selengkapnya tentang AWS DMS, lihat Panduan Pengguna AWS Database Migration Service.

Contoh berikut menunjukkan cara menyiapkan replikasi logis dari basis data Aurora PostgreSQL sebagai penerbit, lalu gunakan AWS DMS untuk melakukan migrasi. Contoh ini menggunakan penerbit dan pelanggan sama yang dibuat di Contoh: Menggunakan replikasi logis dengan klaster DB Aurora PostgreSQL.

Untuk menyiapkan replikasi logis dengan AWS DMS, Anda membutuhkan detail tentang penerbit dan pelanggan dari Amazon RDS. Secara khusus, Anda perlu detail tentang instans DB penulis penerbit dan instans DB pelanggan.

Dapatkan informasi berikut untuk instans DB penulis dari penerbit:

- Pengidentifikasi cloud privat virtual (VPC)
- Grup subnet

- Zona Ketersediaan (AZ)
- Grup keamanan VPC
- ID instans DB

Dapatkan informasi berikut untuk instans DB pelanggan:

- ID instans DB
- Mesin sumber

Untuk menggunakan AWS DMS dalam replikasi logis dengan Aurora PostgreSQL

1. Siapkan basis datas penerbit yang akan digunakan dengan AWS DMS.

   Untuk melakukannya, basis data PostgreSQL 10.x dan yang lebih baru mengharuskan Anda untuk menerapkan fungsi pembungkus AWS DMS pada basis data penerbit. Untuk detail tentang langkah ini dan selanjutnya, lihat petunjuknya di Menggunakan PostgreSQL versi 10.x dan yang lebih baru sebagai sumber untuk AWS DMS dalam Panduan Pengguna AWS Database Migration Service.
2. Masuk ke AWS Management Console dan buka konsol AWS DMS di https://console.aws.amazon.com/dms/v2. Di kanan atas, pilih Wilayah AWS yang sama tempat penerbit dan pelanggan berada.
3. Buat instans replikasi AWS DMS.

   Pilih nilai yang sama seperti nilai instans DB penulis dari penerbit Anda. Hal ini termasuk pengaturan berikut:

   - Untuk VPC, pilih VPC yang sama seperti untuk instans DB penulis.
   - Untuk Grup Subnet Replikasi, pilih grup subnet dengan nilai yang sama dengan instans DB penulis. Buat yang baru jika perlu.
   - Untuk Zona ketersediaan, pilih zona yang sama seperti untuk instans DB penulis.
   - Untuk Grup Keamanan VPC, pilih grup yang sama untuk instans DB penulis.
4. Buat titik akhir AWS DMS sebagai sumber.

   Tentukan penerbit sebagai titik akhir sumber dengan menggunakan pengaturan berikut:

   - Untuk Jenis titik akhir, pilih Titik akhir sumber.

- Pilih Pilih Instans DB RDS.
- Untuk Instans RDS, pilih pengidentifikasi DB dari instans DB penulis dari penerbit.
- Untuk Mesin sumber, pilih postgres.

5. Buat titik akhir AWS DMS sebagai target.

   Tentukan pelanggan sebagai titik akhir target dengan menggunakan pengaturan berikut:

   - Untuk Jenis titik akhir, pilih Titik akhir target.
   - Pilih Pilih Instans DB RDS.
   - Untuk Instans RDS, pilih pengidentifikasi DB dari instans DB pelanggan.
   - Pilih nilai untuk Mesin sumber. Misalnya, jika pelanggannya adalah basis data RDS PostgreSQL, pilih postgres. Jika pelanggannya adalah basis data Aurora PostgreSQL, pilih aurora-postgresql.

6. Buat tugas migrasi basis data AWS DMS.

   Gunakan tugas migrasi basis data untuk menentukan tabel basis data yang akan dimigrasi, memetakan data menggunakan skema target, dan membuat tabel baru pada basis data target. Setidaknya, gunakan pengaturan berikut untuk Konfigurasi tugas:

   - Untuk Instans replikasi, pilih instans replikasi yang Anda buat pada langkah sebelumnya.
   - Untuk Titik akhir basis data sumber, pilih sumber penerbit yang Anda buat pada langkah sebelumnya.
   - Untuk Titik akhir basis data target, pilih target pelanggan yang Anda buat pada langkah sebelumnya.

   Detail tugas lainnya tergantung pada proyek migrasi Anda. Untuk informasi selengkapnya tentang cara menentukan semua detail untuk tugas DMS, lihat Bekerja dengan tugas AWS DMS dalam Panduan Pengguna AWS Database Migration Service.

Setelah AWS DMS membuat tugas, migrasi data dimulai dari penerbit ke pelanggan.

Menggunakan Aurora PostgreSQL sebagai Basis Pengetahuan untuk Amazon Bedrock

Dari Aurora PostgreSQL 15.4, 14.9, 13.12, 12.16 versi, Anda dapat menggunakan cluster Aurora PostgreSQL DB sebagai Basis Pengetahuan untuk Amazon Bedrock. Untuk informasi selengkapnya, lihat Membuat penyimpanan vektor di Amazon Aurora. Basis Pengetahuan secara otomatis mengambil data teks tidak terstruktur yang disimpan dalam bucket Amazon S3, mengubahnya menjadi potongan teks dan vektor, dan menyimpannya dalam database PostgreSQL. Dengan aplikasi AI generatif, Anda dapat menggunakan Agen untuk Amazon Bedrock untuk menanyakan data yang disimpan di Pangkalan Pengetahuan dan menggunakan hasil kueri tersebut untuk menambah jawaban yang disediakan oleh model dasar. Alur kerja ini disebut Retrieval Augmented Generation (RAG). Untuk informasi lebih lanjut tentang RAG, lihat Retrieval Augmented Generation (RAG).

Topik

- Prasyarat
- Mempersiapkan Aurora PostgreSQL untuk digunakan sebagai Basis Pengetahuan untuk Amazon Bedrock
- Membuat basis pengetahuan di konsol Bedrock

Prasyarat

Biasakan diri Anda dengan prasyarat berikut untuk menggunakan klaster Aurora PostgreSQL sebagai Basis Pengetahuan untuk Amazon Bedrock. Pada tingkat tinggi, Anda perlu mengkonfigurasi layanan berikut untuk digunakan dengan Bedrock:

- Amazon Aurora PostgreSQL DB cluster dibuat dalam versi berikut:
  - 15.4 dan versi yang lebih tinggi
  - 14.9 dan versi yang lebih tinggi
  - 13.12 dan versi yang lebih tinggi
  - 12.16 dan versi yang lebih tinggi

> **Note**
>
> Anda harus mengaktifkan `pgvector` ekstensi di database target Anda dan menggunakan versi 0.5.0 atau lebih tinggi. Untuk informasi lebih lanjut, lihat pgvector v0.5.0 dengan pengindeksan HNSW.

- Data API
- Pengguna yang dikelola di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager.

Mempersiapkan Aurora PostgreSQL untuk digunakan sebagai Basis Pengetahuan untuk Amazon Bedrock

Anda harus mengikuti langkah-langkah di bawah ini untuk membuat dan mengkonfigurasi cluster Aurora PostgreSQL DB untuk menggunakannya sebagai Basis Pengetahuan untuk Amazon Bedrock.

1. Buat cluster DB PostgreSQL Aurora. Lihat informasi yang lebih lengkap di Membuat dan terhubung ke klaster DB Aurora PostgreSQL
2. Aktifkan API Data saat membuat cluster Aurora PostgreSQL DB. Untuk informasi selengkapnya tentang versi yang didukung, lihatMenggunakan RDS Data API.
3. Perhatikan klaster Aurora PostgreSQL DB Nama Sumber Daya Amazon (ARN) untuk menggunakannya di Amazon Bedrock. Untuk informasi selengkapnya, lihat Nama Sumber Daya Amazon (ARN)
4. Masuk ke database dengan pengguna master Anda dan atur pgvector. Gunakan perintah berikut jika ekstensi tidak diinstal:

   ```
   CREATE EXTENSION IF NOT EXISTS vector;
   ```

   Gunakan `pgvector` 0.5.0 dan versi yang lebih tinggi yang mendukung pengindeksan HNSW. Untuk informasi lebih lanjut, lihat pgvector v0.5.0 dengan pengindeksan HNSW.

   Gunakan perintah berikut untuk memeriksa versi yang `pg_vector` diinstal:

   ```
   postgres=>SELECT extversion FROM pg_extension WHERE extname='vector';
   ```

5. Buat skema tertentu yang dapat digunakan Bedrock untuk menanyakan data. Gunakan perintah berikut untuk membuat skema:

```
CREATE SCHEMA bedrock_integration;
```

6. Buat peran baru yang dapat digunakan Bedrock untuk menanyakan database. Gunakan perintah berikut untuk membuat peran baru:

```
CREATE ROLE bedrock_user WITH PASSWORD password LOGIN;
```

> **Note**
>
> Catat kata sandi ini karena Anda akan menggunakan kata sandi yang sama untuk membuat kata sandi Secrets Manager.

7. Untuk memberikan `bedrock_user` izin untuk mengelola `bedrock_integration` skema, sehingga mereka dapat membuat tabel atau indeks di dalamnya.

```
GRANT ALL ON SCHEMA bedrock_integration to bedrock_user;
```

8. Login sebagai `bedrock_user` dan membuat tabel di`bedrock_integration schema`.

```
CREATE TABLE bedrock_integration.bedrock_kb (id uuid PRIMARY KEY, embedding
 vector(1536), chunks text, metadata json);
```

9. Kami menyarankan Anda untuk membuat indeks dengan operator kosinus yang dapat digunakan batuan dasar untuk menanyakan data.

```
CREATE INDEX on bedrock_integration.bedrock_kb USING hnsw (embedding
 vector_cosine_ops);
```

10. Buat rahasia AWS Secrets Manager database. Untuk informasi selengkapnya, lihat AWS Secrets Manager rahasia database.

Membuat basis pengetahuan di konsol Bedrock

Saat menyiapkan Aurora PostgreSQL untuk digunakan sebagai penyimpanan vektor untuk Pangkalan Pengetahuan, kumpulkan detail berikut yang perlu Anda suplai ke konsol Amazon Bedrock.

- Amazon Aurora DB klaster ARN
- Rahasia ARN
- Nama database (misalnya postgres)
- Nama tabel - Sarankan untuk memberikan nama skema yang memenuhi syarat, yaitu. BUAT TABEL bedrock_integration.bedrock_kb; yang akan membuat tabel bedrock_kb dalam skema bedrock_integration
- Bidang tabel:

  ID: (id)

  Potongan teks (potongan)

  Penyematan vektor (penyematan)

  Metadata (metadata)

Dengan detail ini Anda dapat membuat basis pengetahuan di konsol Bedrock. Untuk informasi selengkapnya, lihat Membuat penyimpanan vektor di Amazon Aurora.

Setelah Aurora ditambahkan sebagai basis pengetahuan, Anda menelan sumber data ke dalamnya. Untuk informasi selengkapnya, lihat Menyerap sumber data Anda ke dalam basis pengetahuan.

Mengintegrasikan Amazon Aurora PostgreSQL dengan layanan AWS lainnya

Amazon Aurora berintegrasi dengan layanan AWS lain sehingga Anda dapat memperluas klaster DB Aurora PostgreSQL Anda untuk menggunakan kemampuan tambahan di AWS Cloud. Klaster DB Aurora PostgreSQL Anda dapat menggunakan layanan AWS untuk melakukan hal berikut:

- Kumpulkan, lihat, dan nilai dengan cepat performa untuk instans DB Aurora PostgreSQL Anda menggunakan Wawasan Performa Amazon RDS. Wawasan Kinerja memperluas fitur pemantauan Amazon RDS yang ada untuk menggambarkan performa basis data Anda dan membantu menganalisis masalah apa pun yang memengaruhinya. Dengan dasbor Wawasan Performa, Anda dapat memvisualisasikan beban basis data dan memfilter beban berdasarkan waktu tunggu, pernyataan SQL, host, atau pengguna. Untuk informasi selengkapnya tentang Wawasan Performa, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

- Konfigurasikan klaster DB PostgreSQL Aurora Anda untuk mempublikasikan data log ke Amazon Logs. CloudWatch CloudWatch Log menyediakan penyimpanan yang sangat tahan lama untuk catatan log Anda. Dengan CloudWatch Log, Anda dapat melakukan analisis real-time dari data log, dan menggunakannya CloudWatch untuk membuat alarm dan melihat metrik. Untuk informasi selengkapnya, lihat Menerbitkan log Aurora PostgreSQL ke Amazon Logs CloudWatch .
- Impor data dari bucket Amazon S3 ke klaster DB Aurora PostgreSQL, atau ekspor data dari klaster DB Aurora PostgreSQL ke bucket Amazon S3. Lihat informasi yang lebih lengkap di Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL dan Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3.
- Tambahkan prediksi berbasis machine learning ke aplikasi basis data menggunakan bahasa SQL. Pembelajaran mesin Aurora menggunakan integrasi yang sangat optimal antara database Aurora dan layanan pembelajaran AWS mesin (ML) dan Amazon SageMaker Comprehend. Untuk informasi selengkapnya, lihat Menggunakan machine learning Amazon Aurora dengan Aurora PostgreSQL.
- Invokasi fungsi AWS Lambda dari klaster DB Aurora PostgreSQL. Untuk melakukannya, gunakan ekstensi PostgreSQL `aws_lambda` yang disediakan dengan Aurora PostgreSQL. Untuk informasi selengkapnya, lihat .
- Integrasikan kueri dari Amazon Redshift dan Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Mulai menggunakan kueri terfederasi ke PostgreSQL dalam Panduan Developer Basis Data Amazon Redshift.

Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL

Anda dapat mengimpor data yang telah disimpan menggunakan Amazon Simple Storage Service ke dalam tabel pada instans klaster DB Aurora PostgreSQL. Untuk melakukannya, instal ekstensi Aurora PostgreSQL `aws_s3` terlebih dahulu. Ekstensi ini menyediakan fungsi yang Anda gunakan untuk mengimpor data dari bucket Amazon S3. Bucket adalah kontainer Amazon S3 untuk objek dan file. Data dapat berada dalam file nilai yang dipisahkan koma (CSV), file teks, atau file terkompresi (gzip). Di bagian berikut ini, Anda dapat mempelajari cara menginstal ekstensi tersebut dan cara mengimpor data dari Amazon S3 ke dalam tabel.

Basis data Anda harus menjalankan PostgreSQL versi 10.7 atau yang lebih tinggi untuk mengimpor dari Amazon S3 ke Aurora PostgreSQL.

Jika Anda tidak memiliki data yang tersimpan di Amazon S3, Anda harus terlebih dahulu membuat bucket dan menyimpan data. Untuk informasi selengkapnya, lihat topik berikut dalam Panduan Pengguna Amazon Simple Storage Service.

- Buat bucket
- Tambahkan objek ke bucket

Impor lintas akun dari Amazon S3 didukung. Untuk informasi selengkapnya, lihat Memberikan izin lintas akun dalam Panduan Pengguna Amazon Simple Storage Service.

Anda dapat menggunakan kunci yang dikelola pelanggan untuk enkripsi saat mengimpor data dari S3. Untuk informasi selengkapnya, lihat Kunci KMS yang disimpan di AWS KMS dalam Panduan Pengguna Amazon Simple Storage Service.

> **Note**
>
> Pengimporan data dari Amazon S3 tidak didukung untuk Aurora Serverless v1. Hal ini didukung untuk Aurora Serverless v2.

Topik

- Menginstal ekstensi aws_s3
- Gambaran umum pengimporan data dari data Amazon S3
- Mengatur akses ke bucket Amazon S3
- Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL
- Referensi fungsi

Menginstal ekstensi aws_s3

Sebelum Anda dapat menggunakan Amazon S3 dengan klaster DB Aurora PostgreSQL, Anda perlu menginstal ekstensi `aws_s3`. Ekstensi ini menyediakan fungsi untuk mengimpor data dari Amazon S3. Ekstensi ini juga dilengkapi dengan fungsi untuk mengekspor data dari instans klaster DB Aurora PostgreSQL ke bucket Amazon S3. Untuk informasi selengkapnya, lihat Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3. Ekstensi `aws_s3` bergantung pada beberapa fungsi pembantu dalam ekstensi `aws_commons`, yang diinstal secara otomatis jika diperlukan.

Untuk menginstal ekstensi **aws_s3**

1. Gunakan psql (atau pgAdmin) untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL sebagai pengguna yang memiliki hak istimewa `rds_superuser`. Jika Anda tetap menggunakan nama default selama proses penyiapan, Anda terhubung sebagai `postgres`.

   ```
   psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password
   ```

2. Untuk menginstal ekstensi, jalankan perintah berikut.

   ```
   postgres=> CREATE EXTENSION aws_s3 CASCADE;
   NOTICE: installing required extension "aws_commons"
   CREATE EXTENSION
   ```

3. Untuk memastikan bahwa ekstensi telah diinstal, Anda dapat menggunakan metacommand `\dx` psql.

   ```
   postgres=> \dx
          List of installed extensions
       Name     | Version |   Schema   |                 Description
   -------------+---------+------------+---------------------------------------------
    aws_commons | 1.2     | public     | Common data types across AWS services
    aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
    plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
   (3 rows)
   ```

Fungsi untuk mengimpor data dari Amazon S3 dan mengekspor data ke Amazon S3 kini dapat digunakan.

Gambaran umum pengimporan data dari data Amazon S3

Untuk mengimpor data S3 ke Aurora PostgreSQL

Pertama, kumpulkan detail yang perlu Anda berikan ke fungsi tersebut. Hal ini termasuk nama tabel pada instans klaster DB Aurora PostgreSQL Anda, dan nama bucket, jalur file, jenis file, serta Wilayah AWS tempat data Amazon S3 disimpan. Untuk informasi selengkapnya, buka Melihat objek dalam Panduan Pengguna Amazon Simple Storage Service.

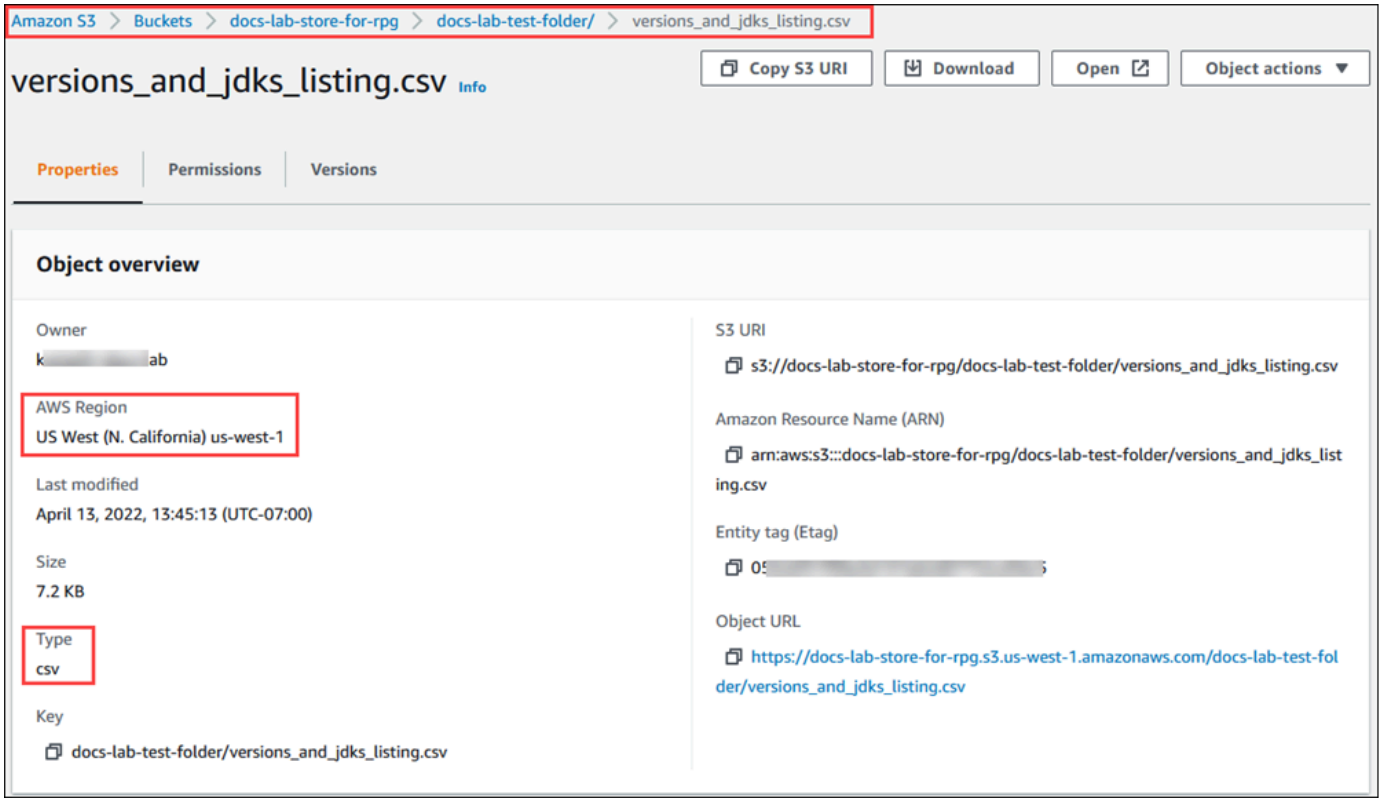
> **Note**
>
> Impor data multibagian dari Amazon S3 saat ini tidak didukung.

1. Dapatkan nama tabel di mana fungsi `aws_s3.table_import_from_s3` adalah untuk mengimpor data. Sebagai contoh, perintah berikut membuat tabel `t1` yang dapat digunakan di langkah selanjutnya.

   ```
   postgres=> CREATE TABLE t1
       (col1 varchar(80),
       col2 varchar(80),
       col3 varchar(80));
   ```

2. Dapatkan detail tentang bucket Amazon S3 dan data yang akan diimpor. Untuk melakukannya, buka konsol Amazon S3 di https://console.aws.amazon.com/s3/, dan pilih Bucket. Temukan bucket yang berisi data Anda dalam daftar. Pilih bucket, buka halaman Ikhtisar objek, lalu pilih Properti.

   Catat nama bucket, jalur, Wilayah AWS, dan jenis file. Anda memerlukan Amazon Resource Name (ARN) nanti untuk mengatur akses ke Amazon S3 melalui peran IAM. Untuk informasi selengkapnya, lihat Mengatur akses ke bucket Amazon S3. Gambar berikut menunjukkan sebuah contoh.



3. Anda dapat memverifikasi jalur ke data di bucket Amazon S3 dengan menggunakan perintah AWS CLI `aws s3 cp`. Jika informasinya benar, perintah ini akan mengunduh salinan file Amazon S3.

```
aws s3 cp s3://sample_s3_bucket/sample_file_path ./
```

4. Siapkan izin di klaster DB Aurora PostgreSQL untuk mengizinkan akses ke file di bucket Amazon S3. Untuk melakukannya, gunakan peran AWS Identity and Access Management (IAM) atau kredensial keamanan. Untuk informasi selengkapnya, lihat Mengatur akses ke bucket Amazon S3.

5. Berikan jalur dan detail objek Amazon S3 lainnya yang dikumpulkan (lihat langkah 2) ke fungsi `create_s3_uri` untuk membuat objek URI Amazon S3. Untuk mempelajari selengkapnya tentang fungsi ini, lihat aws_commons.create_s3_uri. Berikut ini adalah contoh penyusunan objek ini selama sesi psql.

```
postgres=> SELECT aws_commons.create_s3_uri(
   'docs-lab-store-for-rpg',
   'versions_and_jdks_listing.csv',
   'us-west-1'
) AS s3_uri \gset
```

Pada langkah berikutnya, Anda meneruskan objek ini (`aws_commons._s3_uri_1`) ke fungsi `aws_s3.table_import_from_s3` untuk mengimpor data ke tabel.

6. Invokasi fungsi `aws_s3.table_import_from_s3` untuk mengimpor data dari Amazon S3 ke dalam tabel Anda. Untuk informasi referensi, lihat aws_s3.table_import_from_s3. Sebagai contoh, lihat Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL.

Mengatur akses ke bucket Amazon S3

Untuk mengimpor data dari file Amazon S3, berikan izin pada klaster DB Aurora PostgreSQL untuk mengakses bucket Amazon S3 yang berisi file tersebut. Anda dapat menyediakan akses ke bucket Amazon S3 dalam satu dari dua cara, seperti yang dijelaskan dalam topik berikut.

Topik

- Menggunakan peran IAM untuk mengakses bucket Amazon S3
- Menggunakan kredensial keamanan untuk mengakses bucket Amazon S3
- Memecahkan masalah akses ke Amazon S3

Menggunakan peran IAM untuk mengakses bucket Amazon S3

Sebelum Anda memuat data dari file Amazon S3, berikan izin pada klaster DB Aurora PostgreSQL untuk mengakses bucket Amazon S3 yang berisi file tersebut. Dengan cara ini, Anda tidak perlu mengelola informasi kredensial tambahan atau menyediakannya di panggilan fungsi aws_s3.table_import_from_s3.

Untuk melakukannya, buat kebijakan IAM yang memberikan akses ke bucket Amazon S3. Buat peran IAM dan lampirkan kebijakan ke peran tersebut. Kemudian, tetapkan peran IAM ke instans DB Anda.

> **Note**
>
> Anda tidak dapat mengaitkan peran IAM dengan klaster DB Aurora Serverless v1, sehingga langkah berikut tidak berlaku.

Untuk memberi klaster DB Aurora PostgreSQL akses ke Amazon S3 melalui peran IAM

1. Buat kebijakan IAM.

Kebijakan ini memberikan izin pada bucket dan objek yang memungkinkan klaster DB Aurora PostgreSQL Anda mengakses Amazon S3.

Dalam kebijakan, sertakan tindakan yang diperlukan berikut ini untuk memungkinkan transfer file dari bucket Amazon S3 ke Aurora PostgreSQL:

- `s3:GetObject`
- `s3:ListBucket`

Sertakan sumber daya berikut dalam kebijakan untuk mengidentifikasi bucket dan objek Amazon S3 di bucket. Hal ini menunjukkan format Amazon Resource Name (ARN) untuk mengakses Amazon S3.

- arn:aws:s3:::*your-s3-bucket*
- arn:aws:s3:::*your-s3-bucket*/*

Untuk informasi selengkapnya tentang cara membuat kebijakan IAM untuk Aurora PostgreSQL, lihat Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM. Lihat juga Tutorial: Membuat dan melampirkan kebijakan yang dikelola pelanggan pertama Anda dalam Panduan Pengguna IAM.

Perintah AWS CLI berikut membuat kebijakan IAM bernama `rds-s3-import-policy` dengan opsi ini. Perintah tersebut akan memberikan akses ke bucket yang bernama `your-s3-bucket`.

> **Note**
>
> Catat Amazon Resource Name (ARN) dari kebijakan yang ditampilkan oleh perintah ini. Anda memerlukan ARN di langkah berikutnya saat Anda melampirkan kebijakan ke peran IAM.

Example

Untuk Linux, macOS, atau Unix:

```
aws iam create-policy \
   --policy-name rds-s3-import-policy \
```

```
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::your-s3-bucket",
           "arn:aws:s3:::your-s3-bucket/*"
         ]
       }
     ]
   }'
```

Untuk Windows:

```
aws iam create-policy ^
   --policy-name rds-s3-import-policy ^
   --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3import",
         "Action": [
           "s3:GetObject",
           "s3:ListBucket"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::your-s3-bucket",
           "arn:aws:s3:::your-s3-bucket/*"
         ]
       }
     ]
   }'
```

2. Buat peran IAM.

Anda melakukannya agar Aurora PostgreSQL dapat mengambil peran IAM ini untuk mengakses bucket Amazon S3 Anda. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin ke pengguna IAM dalam Panduan Pengguna IAM.

Sebaiknya gunakan kunci konteks kondisi global `aws:SourceArn` dan `aws:SourceAccount` dalam kebijakan berbasis sumber daya untuk membatasi izin layanan ke sumber daya tertentu. Ini adalah perlindungan paling efektif dari masalah confused deputy.

Jika Anda menggunakan kunci konteks kondisi global dan nilai `aws:SourceArn` berisi ID akun, nilai `aws:SourceAccount` dan akun dalam nilai `aws:SourceArn` harus menggunakan ID akun yang sama saat digunakan dalam pernyataan kebijakan yang sama.

- Gunakan `aws:SourceArn` jika Anda menginginkan akses lintas layanan untuk satu sumber daya.
- Gunakan `aws:SourceAccount` jika Anda ingin mengizinkan pengaitan sumber daya apa pun di akun tersebut dengan penggunaan lintas layanan.

Dalam kebijakan, pastikan untuk menggunakan kunci konteks kondisi global `aws:SourceArn` dengan ARN penuh sumber daya. Contoh berikut menunjukkan cara melakukannya menggunakan perintah AWS CLI untuk membuat peran dengan nama `rds-s3-import-role`.

Example

Untuk Linux, macOS, atau Unix:

```
aws iam create-role \
   --role-name rds-s3-import-role \
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
```

```
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                }
            }
        }
    ]
}'
```

Untuk Windows:

```
aws iam create-role ^
   --role-name rds-s3-import-role ^
   --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-
east-1:111122223333:cluster:clustername"
                }
            }
        }
    ]
}'
```

3. Lampirkan kebijakan IAM yang Anda buat ke peran IAM yang Anda buat.

   Perintah AWS CLI berikut melampirkan kebijakan yang dibuat di langkah sebelumnya ke peran bernama `rds-s3-import-role`. Ganti *`your-policy-arn`* dengan kebijakan ARN yang Anda catat di langkah sebelumnya.

   Example

   Untuk Linux, macOS, atau Unix:

```
aws iam attach-role-policy \
   --policy-arn your-policy-arn \
   --role-name rds-s3-import-role
```

Untuk Windows:

```
aws iam attach-role-policy ^
   --policy-arn your-policy-arn ^
   --role-name rds-s3-import-role
```

4. Tambahkan peran IAM ke instans DB.

   Lakukan menggunakan AWS Management Console atau AWS CLI, seperti yang dijelaskan berikut.

Konsol

Untuk menambahkan peran IAM untuk klaster DB PostgreSQL menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih nama klaster DB PostgreSQL untuk menampilkan detailnya.
3. Di tab Konektivitas & keamanan, di bagian Kelola peran IAM, pilih peran yang akan ditambahkan pada bagian Tambahkan peran IAM ke klaster ini.
4. Di bagian Fitur, pilih s3Import.
5. Pilih Tambahkan peran.

AWS CLI

Untuk menambahkan peran IAM untuk klaster DB PostgreSQL menggunakan CLI

- Gunakan perintah berikut untuk menambahkan peran ke klaster DB PostgreSQL bernama `my-db-cluster`. Ganti *`your-role-arn`* dengan ARN peran yang Anda catat pada langkah sebelumnya. Gunakan `s3Import` untuk nilai opsi `--feature-name`.

  Example

  Untuk Linux, macOS, atau Unix:

```
aws rds add-role-to-db-cluster \
   --db-cluster-identifier my-db-cluster \
   --feature-name s3Import \
   --role-arn your-role-arn   \
   --region your-region
```

Untuk Windows:

```
aws rds add-role-to-db-cluster ^
   --db-cluster-identifier my-db-cluster ^
   --feature-name s3Import ^
   --role-arn your-role-arn ^
   --region your-region
```

API RDS

https://docs.aws.amazon.com/AmazonRDS/latest/APIReference/API_AddRoleToDBCluster.html

Menggunakan kredensial keamanan untuk mengakses bucket Amazon S3

Jika Anda mau, Anda dapat menggunakan kredensial keamanan untuk memberikan akses ke bucket Amazon S3 alih-alih memberikan akses ke peran IAM. Anda melakukannya dengan menentukan parameter `credentials` dalam panggilan fungsi aws_s3.table_import_from_s3.

Parameter `credentials` adalah struktur dari jenis `aws_commons._aws_credentials_1`, yang berisi kredensial AWS. Gunakan fungsi aws_commons.create_aws_credentials untuk mengatur kunci akses dan kunci rahasia dalam struktur `aws_commons._aws_credentials_1`, seperti yang ditunjukkan berikut ini.

```
postgres=> SELECT aws_commons.create_aws_credentials(
   'sample_access_key', 'sample_secret_key', '')
AS creds \gset
```

Setelah membuat struktur `aws_commons._aws_credentials_1` , gunakan fungsi aws_s3.table_import_from_s3 dengan parameter `credentials` untuk mengimpor data, seperti yang ditunjukkan berikut.

```
postgres=> SELECT aws_s3.table_import_from_s3(
```

```
   't', '', '(format csv)',
   :'s3_uri',
   :'creds'
);
```

Atau Anda dapat menyertakan panggilan fungsi aws_commons.create_aws_credentials sebaris dalam panggilan fungsi `aws_s3.table_import_from_s3`.

```
postgres=> SELECT aws_s3.table_import_from_s3(
   't', '', '(format csv)',
   :'s3_uri',
   aws_commons.create_aws_credentials('sample_access_key', 'sample_secret_key', '')
);
```

Memecahkan masalah akses ke Amazon S3

Jika Anda mengalami masalah koneksi saat mencoba mengimpor data dari Amazon S3, lihat rekomendasi berikut:

- Memecahkan masalah identitas dan akses Amazon Aurora
- Memecahkan Masalah Amazon S3 dalam Panduan Pengguna Amazon Simple Storage Service
- Pemecahan Masalah Amazon S3 dan IAM dalam Panduan Pengguna IAM

Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL

Anda mengimpor data dari bucket Amazon S3 dengan menggunakan fungsi `table_import_from_s3` aws_s3. Untuk informasi referensi, lihat aws_s3.table_import_from_s3.

> **Note**
>
> Contoh berikut menggunakan metode peran IAM untuk mengizinkan akses ke bucket Amazon S3. Dengan demikian, panggilan fungsi `aws_s3.table_import_from_s3` tidak menyertakan parameter kredensial.

Berikut ini adalah contoh umumnya.

```
postgres=> SELECT aws_s3.table_import_from_s3(
```

```
   't1',
   '',
   '(format csv)',
   :'s3_uri'
);
```

Parameternya adalah sebagai berikut:

- `t1` – Nama untuk tabel dalam klaster DB PostgreSQL tempat tujuan penyalinan data.
- `''` – Daftar opsional kolom dalam tabel basis data. Anda dapat menggunakan parameter ini untuk menunjukkan kolom data S3 yang akan dimasukkan dan kolom tabel tempat memasukkan kolom data S3 tersebut. Jika tidak ada kolom yang ditentukan, semua kolom disalin ke tabel. Untuk contoh cara menggunakan daftar kolom, lihat Mengimpor file Amazon S3 yang menggunakan pemisah kustom.
- `(format csv)` – Argumen PostgreSQL COPY. Proses penyalinan menggunakan argumen dan format perintah PostgreSQL COPY untuk mengimpor data. Pilihan formatnya mencakup nilai yang dipisahkan dengan koma (CSV), seperti yang ditunjukkan dalam contoh ini, serta teks dan biner. Default-nya adalah teks.
- `s3_uri` – Struktur yang berisi informasi yang mengidentifikasi file Amazon S3. Untuk contoh cara menggunakan fungsi aws_commons.create_s3_uri untuk membuat struktur `s3_uri`, lihat Gambaran umum pengimporan data dari data Amazon S3.

Untuk informasi selengkapnya tentang fungsi ini, lihat aws_s3.table_import_from_s3.

Fungsi `aws_s3.table_import_from_s3` menampilkan teks. Untuk menentukan jenis file lain yang akan diimpor dari bucket Amazon S3, lihat salah satu contoh berikut.

> **Note**
>
> Mengimpor file 0 byte akan menyebabkan kesalahan.

Topik

- Mengimpor file Amazon S3 yang menggunakan pemisah kustom
- Mengimpor file (gzip) terkompresi Amazon S3
- Mengimpor file Amazon S3 yang dienkode

Mengimpor file Amazon S3 yang menggunakan pemisah kustom

Contoh berikut menunjukkan cara mengimpor file yang menggunakan pemisah kustom. Hal ini juga menunjukkan cara mengontrol lokasi penempatan data dalam tabel basis data menggunakan parameter `column_list` dari fungsi aws_s3.table_import_from_s3.

Untuk contoh ini, asumsikan bahwa informasi berikut ini diatur ke dalam kolom yang dipisahkan tanda pipa di file Amazon S3.

```
1|foo1|bar1|elephant1
2|foo2|bar2|elephant2
3|foo3|bar3|elephant3
4|foo4|bar4|elephant4
...
```

Untuk mengimpor file yang menggunakan pemisah kustom

1. Buat tabel di basis data untuk data yang diimpor.

   ```
   postgres=> CREATE TABLE test (a text, b text, c text, d text, e text);
   ```

2. Gunakan form berikut dari fungsi aws_s3.table_import_from_s3 untuk mengimpor data dari file Amazon S3.

   Anda dapat memasukkan panggilan fungsi aws_commons.create_s3_uri sebaris dalam panggilan fungsi `aws_s3.table_import_from_s3` untuk menentukan file.

   ```
   postgres=> SELECT aws_s3.table_import_from_s3(
      'test',
      'a,b,d,e',
      'DELIMITER ''|''',
      aws_commons.create_s3_uri('sampleBucket', 'pipeDelimitedSampleFile', 'us-
   east-2')
   );
   ```

Data tersebut sekarang berada dalam tabel di kolom berikut.

```
postgres=> SELECT * FROM test;
a | b | c | d | e
---+------+---+---+------+-----------
```

```
1 | foo1 | | bar1 | elephant1
2 | foo2 | | bar2 | elephant2
3 | foo3 | | bar3 | elephant3
4 | foo4 | | bar4 | elephant4
```

Mengimpor file (gzip) terkompresi Amazon S3

Contoh berikut menunjukkan cara mengimpor file dari Amazon S3 yang dikompresi menggunakan gzip. File yang Anda impor harus memiliki metadata Amazon S3 berikut:

- Kunci: `Content-Encoding`
- Nilai: `gzip`

Jika Anda mengunggah file menggunakan AWS Management Console, metadata biasanya diterapkan oleh sistem. Untuk informasi tentang cara mengunggah file ke Amazon S3 menggunakan AWS Management Console, AWS CLI, atau API, lihat Mengunggah objek dalam Panduan Pengguna Amazon Simple Storage Service.

Untuk informasi selengkapnya tentang metadata Amazon S3 dan detail tentang metadata yang disediakan sistem, lihat Mengedit metadata objek di konsol Amazon S3 dalam Panduan Pengguna Amazon Simple Storage Service.

Impor file gzip ke dalam klaster DB Aurora PostgreSQL seperti yang ditunjukkan berikut ini.

```
postgres=> CREATE TABLE test_gzip(id int, a text, b text, c text, d text);
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_gzip', '', '(format csv)',
 'myS3Bucket', 'test-data.gz', 'us-east-2'
);
```

Mengimpor file Amazon S3 yang dienkode

Contoh berikut menunjukkan cara mengimpor file dari Amazon S3 yang memiliki pengenkodean Windows-1252.

```
postgres=> SELECT aws_s3.table_import_from_s3(
 'test_table', '', 'encoding ''WIN1252''',
 aws_commons.create_s3_uri('sampleBucket', 'SampleFile', 'us-east-2')
);
```

Referensi fungsi

Fungsi

- aws_s3.table_import_from_s3
- aws_commons.create_s3_uri
- aws_commons.create_aws_credentials

aws_s3.table_import_from_s3

Mengimpor data Amazon S3 ke tabel Aurora PostgreSQL. Ekstensi `aws_s3` memberikan fungsi `aws_s3.table_import_from_s3`. Nilai yang ditampilkan berupa teks.

Sintaksis

Parameter yang diperlukan adalah `table_name`, `column_list`, dan `options`. Parameter ini mengidentifikasi tabel basis data dan menentukan cara data disalin ke dalam tabel.

Anda juga dapat menggunakan parameter berikut ini:

- Parameter `s3_info` menentukan file Amazon S3 yang akan diimpor. Saat Anda menggunakan parameter ini, akses ke Amazon S3 disediakan oleh peran IAM untuk klaster DB PostgreSQL.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1
)
```

- Parameter `credentials` menentukan kredensial untuk mengakses Amazon S3. Saat Anda menggunakan parameter ini, Anda tidak menggunakan peran IAM.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   s3_info aws_commons._s3_uri_1,
   credentials aws_commons._aws_credentials_1
)
```

Parameter

table_name

String teks wajib yang berisi nama tabel basis data PostgreSQL sebagai tujuan impor data.

column_list

String teks wajib yang berisi daftar opsional kolom tabel basis data PostgreSQL untuk menyalin data. Jika string kosong, semua kolom tabel digunakan. Sebagai contoh, lihat Mengimpor file Amazon S3 yang menggunakan pemisah kustom.

options

String teks wajib yang berisi argumen untuk perintah `COPY` PostgreSQL. Argumen ini menentukan cara data akan disalin ke dalam tabel PostgreSQL. Untuk detail selengkapnya, lihat dokumentasi COPY PostgreSQL.

s3_info

Jenis komposit `aws_commons._s3_uri_1` yang berisi informasi tentang objek S3 berikut:

- `bucket` – Nama bucket Amazon S3 yang berisi file.
- `file_path` – Nama file Amazon S3 termasuk jalur file.
- `region` – Wilayah AWS tempat file berada. Untuk daftar nama Wilayah AWS dan nilai terkait, lihat Wilayah dan Zona Ketersediaan.

credentials

Jenis komposit `aws_commons._aws_credentials_1` yang berisi kredensial berikut yang akan digunakan untuk operasi impor:

- Kunci akses
- Kunci rahasia
- Token sesi

Untuk informasi tentang cara membuat struktur komposit `aws_commons._aws_credentials_1`, lihat aws_commons.create_aws_credentials.

Sintaksis alternatif

Untuk memudahkan pengujian, Anda dapat menggunakan serangkaian parameter yang diperluas, bukan parameter `s3_info` dan `credentials`. Berikut ini adalah variasi sintaksis tambahan untuk fungsi `aws_s3.table_import_from_s3`:

- Alih-alih menggunakan parameter `s3_info` untuk mengidentifikasi file Amazon S3, gunakan kombinasi parameter `bucket`, `file_path`, dan `region`. Dengan form fungsi ini, akses ke Amazon S3 disediakan oleh peran IAM pada instans DB PostgreSQL.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text
)
```

- Alih-alih menggunakan parameter `credentials` untuk menentukan akses Amazon S3, gunakan kombinasi parameter `access_key`, `session_key`, dan `session_token`.

```
aws_s3.table_import_from_s3 (
   table_name text,
   column_list text,
   options text,
   bucket text,
   file_path text,
   region text,
   access_key text,
   secret_key text,
   session_token text
)
```

Parameter alternatif

bucket

String teks yang berisi nama bucket Amazon S3 yang berisi file.

file_path

String teks yang berisi nama file Amazon S3 beserta jalur file.

region

String teks yang mengidentifikasi lokasi Wilayah AWS file. Untuk daftar nama Wilayah AWS dan nilai terkait, lihat Wilayah dan Zona Ketersediaan.

access_key

String teks yang berisi kunci akses yang akan digunakan dalam operasi impor. Default-nya adalah NULL.

secret_key

String teks yang berisi kunci rahasia yang akan digunakan dalam operasi impor. Default-nya adalah NULL.

session_token

(Opsional) String teks yang berisi kunci sesi yang akan digunakan untuk operasi impor. Default-nya adalah NULL.

aws_commons.create_s3_uri

Membuat struktur `aws_commons._s3_uri_1` untuk menyimpan informasi file Amazon S3. Gunakan hasil dari fungsi `aws_commons.create_s3_uri` di parameter `s3_info` dari fungsi aws_s3.table_import_from_s3.

Sintaksis

```
aws_commons.create_s3_uri(
   bucket text,
   file_path text,
   region text
)
```

Parameter

bucket

String teks wajib berisi nama bucket Amazon S3 untuk file.

file_path

String teks wajib yang berisi nama file Amazon S3 beserta jalurnya.

region

String teks wajib yang berisi Wilayah AWS tempat file berada. Untuk daftar nama Wilayah AWS dan nilai terkait, lihat Wilayah dan Zona Ketersediaan.

aws_commons.create_aws_credentials

Mengatur kunci akses dan kunci rahasia dalam struktur `aws_commons._aws_credentials_1`. Gunakan hasil dari fungsi `aws_commons.create_aws_credentials` di parameter `credentials` dari fungsi aws_s3.table_import_from_s3.

Sintaksis

```
aws_commons.create_aws_credentials(
   access_key text,
   secret_key text,
   session_token text
)
```

Parameter

access_key

String teks wajib yang berisi kunci akses yang digunakan untuk mengimpor file Amazon S3. Default-nya adalah NULL.

secret_key

String teks wajib yang berisi kunci rahasia yang digunakan untuk mengimpor file Amazon S3. Default-nya adalah NULL.

session_token

String teks opsional yang berisi token sesi yang akan digunakan untuk mengimpor file Amazon S3. Default-nya adalah NULL. Jika Anda memberikan `session_token` opsional, Anda dapat menggunakan kredensial sementara.

Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3

Anda dapat mengueri data dari klaster DB Aurora PostgreSQL dan mengekspornya langsung ke file yang disimpan dalam bucket Amazon S3. Untuk melakukannya, instal ekstensi Aurora PostgreSQL `aws_s3` terlebih dahulu. Ekstensi ini memberi Anda fungsi yang Anda gunakan untuk mengekspor hasil kueri ke Amazon S3. Berikut ini, Anda dapat mengetahui cara menginstal ekstensi dan cara mengekspor data ke Amazon S3.

Anda dapat mengekspor dari instans DB Aurora Serverless v2 atau yang tersedia. Langkah-langkah ini tidak didukung untuk Aurora Serverless v1.

> **Note**
>
> Ekspor lintas akun ke Amazon S3 tidak didukung.

Semua versi Aurora PostgreSQL yang tersedia saat ini mendukung ekspor data ke Amazon Simple Storage Service. Untuk informasi versi terperinci, lihat Pembaruan Amazon Aurora PostgreSQL dalam Catatan Rilis untuk Aurora PostgreSQL.

Jika Anda belum menyiapkan bucket untuk ekspor, lihat topik berikut, Panduan Pengguna Amazon Simple Storage Service.

- Menyiapkan Amazon S3
- Membuat bucket

Secara default, data yang diekspor dari Aurora PostgreSQL ke Amazon S3 menggunakan enkripsi sisi server dengan. Kunci yang dikelola AWS Anda dapat menggunakan kunci terkelola pelanggan yang telah Anda buat. Jika Anda menggunakan enkripsi bucket, bucket Amazon S3 harus dienkripsi dengan AWS Key Management Service (AWS KMS) key (SSE-KMS). Saat ini, bucket yang dienkripsi dengan kunci terkelola Amazon S3 (SSE-S3) tidak didukung.

> **Note**
>
> Anda dapat menyimpan data snapshot cluster DB dan DB ke Amazon S3 menggunakan AWS Management Console AWS CLI,, atau Amazon RDS API. Untuk informasi selengkapnya, lihat Mengekspor data snapshot klaster DB ke Amazon S3.

Topik

- Menginstal ekstensi aws_s3
- Ikhtisar ekspor data ke Amazon S3
- Menentukan jalur file Amazon S3 tujuan ekspor
- Menyiapkan akses ke bucket Amazon S3
- Mengekspor data kueri menggunakan fungsi aws_s3.query_export_to_s3
- Memecahkan masalah akses ke Amazon S3
- Referensi fungsi

Menginstal ekstensi aws_s3

Sebelum Anda dapat menggunakan Amazon Simple Storage Service dengan klaster DB Aurora PostgreSQL, Anda perlu menginstal ekstensi `aws_s3`. Ekstensi ini memberikan fungsi untuk mengekspor data dari instans penulis klaster DB Aurora PostgreSQL ke bucket Amazon S3. Ini juga menyediakan fungsi untuk mengimpor data dari Amazon S3. Untuk informasi selengkapnya, lihat Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL. Ekstensi `aws_s3` bergantung pada beberapa fungsi pembantu dalam ekstensi `aws_commons`, yang diinstal secara otomatis bila diperlukan.

Untuk menginstal ekstensi **`aws_s3`**

1. Gunakan psql (atau pgAdmin) untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL sebagai pengguna yang memiliki hak istimewa `rds_superuser`. Jika Anda menyimpan nama default selama proses penyiapan, Anda terhubung sebagai `postgres`.

   ```
   psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password
   ```

2. Untuk menginstal ekstensi, jalankan perintah berikut.

   ```
   postgres=> CREATE EXTENSION aws_s3 CASCADE;
   NOTICE: installing required extension "aws_commons"
   CREATE EXTENSION
   ```

3. Untuk memverifikasi bahwa ekstensi sudah diinstal, Anda dapat menggunakan metacommand psql `\dx`.

   ```
   postgres=> \dx
          List of installed extensions
       Name     | Version |   Schema   |                Description
   -------------+---------+------------+-------------------------------------------
    aws_commons | 1.2     | public     | Common data types across AWS services
    aws_s3      | 1.1     | public     | AWS S3 extension for importing data from S3
    plpgsql     | 1.0     | pg_catalog | PL/pgSQL procedural language
   (3 rows)
   ```

Fungsi untuk mengimpor data dari Amazon S3 dan mengekspor data ke Amazon S3 kini dapat digunakan.

Verifikasi bahwa versi Aurora PostgreSQL Anda mendukung ekspor ke Amazon S3

Anda dapat memverifikasi bahwa versi RDS for PostgreSQL Anda mendukung ekspor ke Amazon S3 dengan menggunakan perintah `describe-db-engine-versions`. Contoh berikut memeriksa untuk melihat apakah versi 10.14 dapat mengekspor ke Amazon S3.

```
aws rds describe-db-engine-versions --region us-east-1 \
--engine aurora-postgresql --engine-version 10.14 | grep s3Export
```

Jika output-nya menyertakan string `"s3Export"`, berarti mesinnya mendukung ekspor Amazon S3. Jika tidak, mesin tidak mendukungnya.

Ikhtisar ekspor data ke Amazon S3

Untuk mengekspor data yang disimpan dalam basis data Aurora PostgreSQL ke bucket Amazon S3, gunakan prosedur berikut.

Untuk mengekspor data Aurora PostgreSQL ke S3

1. Identifikasi jalur file Amazon S3 yang akan digunakan untuk mengekspor data. Untuk detail tentang proses ini, lihat Menentukan jalur file Amazon S3 tujuan ekspor.
2. Berikan izin untuk mengakses bucket Amazon S3.

   Untuk mengekspor data ke file Amazon S3, beri klaster DB Aurora PostgreSQL izin untuk mengakses bucket Amazon S3 yang akan digunakan untuk penyimpanan data yang diekspor. Berikut adalah langkah-langkahnya:

   1. Buat kebijakan IAM yang memberikan akses ke bucket Amazon S3 tempat tujuan ekspor.
   2. Buat peran IAM.
   3. Lampirkan kebijakan yang Anda buat ke peran yang Anda buat.
   4. Tambahkan peran IAM ini ke klaster DB .

   Untuk detail tentang proses ini, lihat Menyiapkan akses ke bucket Amazon S3.
3. Identifikasi kueri basis data untuk mendapatkan data. Ekspor data kueri dengan memanggil fungsi `aws_s3.query_export_to_s3`.

Setelah menyelesaikan tugas persiapan sebelumnya, gunakan fungsi aws_s3.query_export_to_s3 untuk mengekspor hasil kueri ke Amazon S3. Untuk detail tentang proses ini, lihat Mengekspor data kueri menggunakan fungsi aws_s3.query_export_to_s3.

Menentukan jalur file Amazon S3 tujuan ekspor

Tentukan informasi berikut untuk mengidentifikasi lokasi di Amazon S3 tempat Anda ingin mengekspor data:

- Nama bucket – Bucket adalah kontainer untuk objek atau file Amazon S3.

  Untuk informasi selengkapnya tentang menyimpan data dengan Amazon S3, lihat Membuat bucket dan Melihat objek dalam Panduan Pengguna Amazon Simple Storage Service.
- Jalur file – Jalur file mengidentifikasi tempat penyimpanan data yang diekspor dalam bucket Amazon S3. Jalur file terdiri atas:
  - Awalan jalur opsional yang mengidentifikasi jalur folder virtual.
  - Awalan file yang mengidentifikasi satu atau beberapa file yang akan disimpan. Ekspor yang lebih besar disimpan dalam beberapa file, masing-masing berukuran maksimum sekitar 6 GB. Nama file tambahan memiliki awalan file yang sama, tetapi dengan penambahan `_part`*XX*. *XX* mewakili 2, lalu 3, dan seterusnya.

  Misalnya, jalur file dengan folder `exports` dan awalan file `query-1-export` adalah `/exports/query-1-export`.
- AWS Wilayah (opsional) - AWS Wilayah tempat bucket Amazon S3 berada.

  > **Note**
  >
  > Saat ini, AWS Wilayah harus sama dengan wilayah DB cluster DB yang mengekspor.

  Untuk daftar nama AWS Wilayah dan nilai terkait, lihatWilayah dan Zona Ketersediaan.

Untuk menyimpan informasi file Amazon S3 tentang lokasi penyimpanan file yang diekspor, Anda dapat menggunakan fungsi aws_commons.create_s3_uri untuk membuat struktur komposit `aws_commons._s3_uri_1` sebagai berikut.

```
psql=> SELECT aws_commons.create_s3_uri(
   'sample-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

Kemudian, berikan nilai `s3_uri_1` ini sebagai parameter untuk memanggil fungsi aws_s3.query_export_to_s3. Sebagai contoh, lihat Mengekspor data kueri menggunakan fungsi aws_s3.query_export_to_s3.

Menyiapkan akses ke bucket Amazon S3

Untuk mengekspor data ke Amazon S3, berikan izin kepada klaster DB PostgreSQL Anda untuk mengakses bucket Amazon S3 yang akan dimasuki file.

Untuk melakukannya, gunakan prosedur berikut.

Untuk memberi klaster DB PostgreSQL akses ke Amazon S3 melalui peran IAM

1. Buat kebijakan IAM.

   Kebijakan ini memberikan izin kepada bucket dan objek yang memungkinkan klaster DB PostgreSQL Anda mengakses Amazon S3.

   Sebagai bagian dari pembuatan kebijakan ini, lakukan langkah-langkah berikut:

   a. Sertakan tindakan yang diperlukan berikut dalam kebijakan untuk mengizinkan transfer file dari klaster DB PostgreSQL ke bucket Amazon S3:

      - `s3:PutObject`
      - `s3:AbortMultipartUpload`

   b. Sertakan Amazon Resource Name (ARN) yang mengidentifikasi bucket dan objek Amazon S3 dalam bucket. Format ARN untuk mengakses Amazon S3 adalah: `arn:aws:s3:::`*`your-s3-bucket`*`/*`

   Untuk informasi selengkapnya tentang cara membuat kebijakan IAM untuk Aurora PostgreSQL, lihat Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM. Lihat juga Tutorial: Membuat dan melampirkan kebijakan yang dikelola pelanggan pertama Anda di Panduan Pengguna IAM.

AWS CLI Perintah berikut membuat kebijakan IAM bernama `rds-s3-export-policy` dengan opsi ini. Ini memberikan akses ke bucket bernama `your-s3-bucket`.

> **Warning**
>
> Sebaiknya Anda menyiapkan basis data Anda dengan VPC privat yang memiliki kebijakan titik akhir yang dikonfigurasi untuk mengakses bucket tertentu. Untuk informasi selengkapnya, lihat Menggunakan kebijakan titik akhir untuk Amazon S3 di Panduan Pengguna Amazon VPC.
>
> Kami sangat menyarankan Anda agar tidak membuat kebijakan dengan akses semua sumber daya. Akses ini dapat menjadi ancaman bagi data keamanan. Jika Anda membuat kebijakan yang memberi akses `S3:PutObject` ke semua sumber daya menggunakan `"Resource":"*"`, pengguna yang memiliki hak istimewa ekspor dapat mengekspor data ke semua bucket di akun Anda. Selain itu, pengguna dapat mengekspor data ke bucket yang dapat ditulis secara publik Wilayah AWS Anda.

Setelah Anda membuat kebijakan, catat Amazon Resource Name (ARN) dari kebijakan tersebut. Anda memerlukan ARN ini untuk langkah berikutnya ketika Anda melampirkan kebijakan ke peran IAM.

```
aws iam create-policy  --policy-name rds-s3-export-policy  --policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Sid": "s3export",
         "Action": [
           "s3:PutObject",
           "s3:AbortMultipartUpload"
         ],
         "Effect": "Allow",
         "Resource": [
           "arn:aws:s3:::your-s3-bucket/*"
         ]
       }
     ]
   }'
```

2. Buat peran IAM.

Lakukan langkah ini agar Aurora PostgreSQL dapat mengambil peran IAM ini atas nama Anda untuk mengakses bucket Amazon S3. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin ke pengguna IAM dalam Panduan Pengguna IAM.

Sebaiknya gunakan kunci konteks kondisi global `aws:SourceArn` dan `aws:SourceAccount` dalam kebijakan berbasis sumber daya untuk membatasi izin layanan ke sumber daya tertentu. Ini adalah cara paling efektif untuk melindungi dari masalah deputi yang membingungkan.

Jika Anda menggunakan kunci konteks kondisi global dan nilai `aws:SourceArn` berisi ID akun, nilai `aws:SourceAccount` dan akun dalam nilai `aws:SourceArn` harus menggunakan ID akun yang sama saat digunakan dalam pernyataan kebijakan yang sama.

- Gunakan `aws:SourceArn` jika Anda menginginkan akses lintas layanan untuk satu sumber daya.
- Gunakan `aws:SourceAccount` jika Anda ingin mengizinkan sumber daya apa pun di akun tersebut dikaitkan dengan penggunaan lintas layanan.

Dalam kebijakan, pastikan untuk menggunakan kunci konteks kondisi global `aws:SourceArn` dengan ARN penuh sumber daya. Contoh berikut menunjukkan bagaimana melakukannya dengan menggunakan AWS CLI perintah untuk membuat peran bernama`rds-s3-export-role`.

Example

Untuk Linux, macOS, atau Unix:

```
aws iam create-role  \
    --role-name rds-s3-export-role  \
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
```

```
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-east-1:111122223333:db:dbname"
                }
            }
        }
      ]
   }'
```

Untuk Windows:

```
aws iam create-role  ^
    --role-name rds-s3-export-role  ^
    --assume-role-policy-document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
            "Service": "rds.amazonaws.com"
          },
         "Action": "sts:AssumeRole",
         "Condition": {
             "StringEquals": {
                "aws:SourceAccount": "111122223333",
                "aws:SourceArn": "arn:aws:rds:us-east-1:111122223333:db:dbname"
                }
            }
       }
     ]
   }'
```

3. Lampirkan kebijakan IAM yang Anda buat ke peran IAM yang Anda buat.

   AWS CLI Perintah berikut melampirkan kebijakan yang dibuat sebelumnya ke peran bernama `rds-s3-export-role`. Ganti *`your-policy-arn`* dengan ARN kebijakan yang Anda catat di langkah sebelumnya.

   ```
   aws iam attach-role-policy  --policy-arn your-policy-arn  --role-name rds-s3-
   export-role
   ```

4. Tambahkan peran IAM ke klaster DB. Anda melakukannya dengan menggunakan AWS Management Console atau AWS CLI, seperti yang dijelaskan berikut.

Konsol

Untuk menambahkan peran IAM untuk klaster DB PostgreSQL menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih nama klaster DB PostgreSQL untuk menampilkan detailnya.
3. Di tab Konektivitas & keamanan, di bagian Kelola peran IAM, pilih peran yang akan ditambahkan pada bagian Tambahkan peran IAM ke instans ini.
4. Di bagian Fitur, pilih s3Export.
5. Pilih Tambahkan peran.

AWS CLI

Untuk menambahkan peran IAM untuk klaster DB PostgreSQL menggunakan CLI

- Gunakan perintah berikut untuk menambahkan peran ke klaster DB PostgreSQL bernama `my-db-cluster`. Ganti *`your-role-arn`* dengan ARN peran yang Anda catat pada langkah sebelumnya. Gunakan `s3Export` untuk nilai opsi `--feature-name`.

  Example

  Untuk Linux, macOS, atau Unix:

```
aws rds add-role-to-db-cluster \
   --db-cluster-identifier my-db-cluster \
   --feature-name s3Export \
   --role-arn your-role-arn   \
   --region your-region
```

  Untuk Windows:

```
aws rds add-role-to-db-cluster ^
   --db-cluster-identifier my-db-cluster ^
   --feature-name s3Export ^
   --role-arn your-role-arn ^
   --region your-region
```

Mengekspor data kueri menggunakan fungsi aws_s3.query_export_to_s3

Ekspor data PostgreSQL Anda ke Amazon S3 dengan memanggil fungsi aws_s3.query_export_to_s3.

Topik

- Prasyarat
- Memanggil aws_s3.query_export_to_s3
- Mengekspor ke file CSV yang menggunakan pembatas kustom
- Mengekspor ke file biner dengan pengodean

Prasyarat

Sebelum menggunakan fungsi `aws_s3.query_export_to_s3`, pastikan untuk melengkapi prasyarat berikut:

- Instal ekstensi PostgreSQL yang diperlukan seperti yang dijelaskan di Ikhtisar ekspor data ke Amazon S3.
- Tentukan tempat untuk mengekspor data Anda ke Amazon S3 seperti yang dijelaskan di Menentukan jalur file Amazon S3 tujuan ekspor.
- Pastikan bahwa klaster DB memiliki akses ekspor ke Amazon S3 seperti yang dijelaskan di Menyiapkan akses ke bucket Amazon S3.

Contoh berikut menggunakan tabel basis data yang disebut `sample_table`. Contoh ini mengekspor data ke dalam bucket bernama `sample-bucket`. Contoh tabel dan data dibuat dengan pernyataan SQL berikut di psql.

```
psql=> CREATE TABLE sample_table (bid bigint PRIMARY KEY, name varchar(80));
psql=> INSERT INTO sample_table (bid,name) VALUES (1, 'Monday'), (2,'Tuesday'), (3,
 'Wednesday');
```

Memanggil aws_s3.query_export_to_s3

Berikut ini adalah cara-cara dasar untuk memanggil fungsi aws_s3.query_export_to_s3.

Contoh ini menggunakan variabel `s3_uri_1` untuk mengidentifikasi struktur berisi informasi yang mengidentifikasi file Amazon S3. Gunakan fungsi aws_commons.create_s3_uri untuk membuat struktur.

```
psql=> SELECT aws_commons.create_s3_uri(
   'sample-bucket',
   'sample-filepath',
   'us-west-2'
) AS s3_uri_1 \gset
```

Meskipun parameter bervariasi untuk dua panggilan fungsi `aws_s3.query_export_to_s3` berikut, hasilnya sama untuk contoh ini. Semua baris dari tabel `sample_table` diekspor ke dalam bucket yang disebut `sample-bucket`.

```
psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1');

psql=> SELECT * FROM aws_s3.query_export_to_s3('SELECT * FROM
 sample_table', :'s3_uri_1', options :='format text');
```

Parameternya dijelaskan sebagai berikut:

- `'SELECT * FROM sample_table'` – Parameter pertama adalah string teks wajib yang berisi kueri SQL. Mesin PostgreSQL menjalankan kueri ini. Hasil kueri disalin ke bucket S3 yang diidentifikasi dalam parameter lain.
- `:'s3_uri_1'` – Parameter ini adalah struktur yang mengidentifikasi file Amazon S3. Contoh ini menggunakan variabel untuk mengidentifikasi struktur yang dibuat sebelumnya. Anda dapat membuat struktur dengan menyertakan baris panggilan fungsi `aws_commons.create_s3_uri` sebaris dalam panggilan fungsi `aws_s3.query_export_to_s3` sebagai berikut.

  ```
  SELECT * from aws_s3.query_export_to_s3('select * from sample_table',
     aws_commons.create_s3_uri('sample-bucket', 'sample-filepath', 'us-west-2')
  );
  ```

- `options :='format text'` – Parameter `options` adalah string teks opsional yang berisi argumen `COPY` PostgreSQL. Proses penyalinan menggunakan argumen dan format perintah PostgreSQL COPY.

Jika file yang ditentukan tidak ada dalam bucket Amazon S3, file tersebut akan dibuat. Jika file sudah ada, file tersebut akan ditimpa. Sintaks untuk mengakses data yang diekspor di Amazon S3 adalah sebagai berikut.

```
s3-region://bucket-name[/path-prefix]/file-prefix
```

Ekspor yang lebih besar disimpan dalam beberapa file, masing-masing berukuran maksimum sekitar 6 GB. Nama file tambahan memiliki awalan file yang sama, tetapi dengan penambahan _part*XX*. *XX* mewakili 2, lalu 3, dan seterusnya. Sebagai contoh, misalkan Anda menentukan jalur tempat Anda menyimpan file data sebagai berikut.

```
s3-us-west-2://my-bucket/my-prefix
```

Jika ekspor harus membuat tiga file data, bucket Amazon S3 berisi file data berikut.

```
s3-us-west-2://my-bucket/my-prefix
s3-us-west-2://my-bucket/my-prefix_part2
s3-us-west-2://my-bucket/my-prefix_part3
```

Untuk referensi selengkapnya tentang fungsi ini dan cara lain untuk memanggilnya, lihat aws_s3.query_export_to_s3. Untuk informasi selengkapnya tentang cara mengakses file di Amazon S3, buka Melihat objek dalam Panduan Pengguna Amazon Simple Storage Service.

Mengekspor ke file CSV yang menggunakan pembatas kustom

Contoh berikut menunjukkan cara memanggil fungsi aws_s3.query_export_to_s3 untuk mengekspor data ke file yang menggunakan pembatas kustom. Contoh ini menggunakan argumen perintah PostgreSQL COPY untuk menetapkan format nilai yang dipisahkan koma (CSV) dan pembatas titik dua (:).

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format csv, delimiter $$:$$');
```

Mengekspor ke file biner dengan pengodean

Contoh berikut menunjukkan cara memanggil fungsi aws_s3.query_export_to_s3 untuk mengekspor data ke file biner yang memiliki pengodean Windows-1253.

```
SELECT * from aws_s3.query_export_to_s3('select * from basic_test', :'s3_uri_1',
 options :='format binary, encoding WIN1253');
```

Memecahkan masalah akses ke Amazon S3

Jika Anda mengalami masalah koneksi saat mencoba mengekspor data ke Amazon S3, pertama pastikan aturan akses keluar untuk grup keamanan VPC yang terkait dengan instans DB Anda mengizinkan konektivitas jaringan. Khususnya, grup keamanan harus memiliki aturan yang mengizinkan instans DB mengirim lalu lintas TCP ke port 443 dan ke alamat IPv4 mana pun (0.0.0.0/0). Untuk informasi selengkapnya, lihat Berikan akses ke klaster DB dalam VPC dengan membuat grup keamanan.

Lihat juga rekomendasi berikut:

- Memecahkan masalah identitas dan akses Amazon Aurora
- Memecahkan Masalah Amazon S3 di Panduan Pengguna Amazon Simple Storage Service
- Memecahkan Masalah Amazon S3 dan IAM di Panduan Pengguna IAM

Referensi fungsi

Fungsi

- aws_s3.query_export_to_s3
- aws_commons.create_s3_uri

aws_s3.query_export_to_s3

Mengekspor hasil kueri PostgreSQL ke bucket Amazon S3. Ekstensi `aws_s3` memberikan fungsi `aws_s3.query_export_to_s3`.

Dua parameter yang dibutuhkan adalah `query` dan `s3_info`. Parameter ini menentukan kueri yang akan diekspor dan mengidentifikasi bucket Amazon S3 tempat tujuan ekspor. Parameter opsional yang disebut `options` disediakan untuk menentukan berbagai parameter ekspor. Sebagai contoh penggunaan fungsi `aws_s3.query_export_to_s3`, lihat Mengekspor data kueri menggunakan fungsi aws_s3.query_export_to_s3.

Sintaksis

```
aws_s3.query_export_to_s3(
    query text,
    s3_info aws_commons._s3_uri_1,
    options text,
```

```
    kms_key text
)
```

Parameter input

query

String teks yang diperlukan yang berisi kueri SQL yang dijalankan mesin PostgreSQL. Hasil kueri ini disalin ke bucket S3 yang diidentifikasi dalam parameter `s3_info`.

s3_info

Jenis komposit `aws_commons._s3_uri_1` yang berisi informasi tentang objek S3 berikut:

- `bucket` – Nama bucket Amazon S3 yang akan diisi file.
- `file_path` – Nama dan jalur file Amazon S3.
- `region`— AWS Wilayah tempat ember berada. Untuk daftar nama AWS Wilayah dan nilai terkait, lihatWilayah dan Zona Ketersediaan.

  Saat ini, nilai ini harus AWS Wilayah yang sama dengan DB cluster DB yang mengekspor. Defaultnya adalah AWS Wilayah DB cluster DB yang mengekspor.

Untuk membuat struktur komposit `aws_commons._s3_uri_1`, lihat fungsi aws_commons.create_s3_uri.

options

String teks opsional yang berisi argumen untuk perintah `COPY` PostgreSQL. Argumen ini menentukan cara menyalin data saat diekspor. Untuk detail selengkapnya, lihat Dokumentasi PostgreSQL COPY.

teks kms_key

String teks opsional yang berisi kunci KMS yang dikelola pelanggan dari bucket S3 untuk mengekspor data.

Parameter input alternatif

Untuk memudahkan pengujian, Anda dapat menggunakan serangkaian parameter yang diperluas, bukan parameter `s3_info`. Berikut ini adalah variasi sintaks tambahan untuk fungsi `aws_s3.query_export_to_s3`.

Alih-alih menggunakan parameter `s3_info` untuk mengidentifikasi file Amazon S3, gunakan kombinasi parameter `bucket`, `file_path`, dan `region`.

```
aws_s3.query_export_to_s3(
    query text,
    bucket text,
    file_path text,
    region text,
    options text,
    kms_key text
)
```

query

String teks yang diperlukan yang berisi kueri SQL yang dijalankan mesin PostgreSQL. Hasil kueri ini disalin ke bucket S3 yang diidentifikasi dalam parameter `s3_info`.

bucket

String teks yang diperlukan yang berisi nama bucket Amazon S3 yang berisi file.

file_path

String teks yang diperlukan yang berisi nama file Amazon S3 beserta jalurnya.

region

String teks opsional yang berisi AWS Wilayah tempat bucket berada. Untuk daftar nama AWS Wilayah dan nilai terkait, lihatWilayah dan Zona Ketersediaan.

Saat ini, nilai ini harus AWS Wilayah yang sama dengan DB cluster DB yang mengekspor. Defaultnya adalah AWS Wilayah DB cluster DB yang mengekspor.

options

String teks opsional yang berisi argumen untuk perintah `COPY` PostgreSQL. Argumen ini menentukan cara menyalin data saat diekspor. Untuk detail selengkapnya, lihat Dokumentasi PostgreSQL COPY.

teks kms_key

String teks opsional yang berisi kunci KMS yang dikelola pelanggan dari bucket S3 untuk mengekspor data.

Parameter output

```
aws_s3.query_export_to_s3(
    OUT rows_uploaded bigint,
    OUT files_uploaded bigint,
    OUT bytes_uploaded bigint
)
```

rows_uploaded

Jumlah baris tabel yang berhasil diunggah ke Amazon S3 untuk kueri tertentu.

files_uploaded

Jumlah file yang diunggah ke Amazon S3. File dibuat dalam ukuran kira-kira 6 GB. Setiap file tambahan yang dibuat memiliki `_part`*XX* yang ditambahkan pada namanya. *XX* mewakili 2, kemudian 3, dan seterusnya sesuai kebutuhan.

bytes_uploaded

Jumlah total byte yang diunggah ke Amazon S3.

Contoh-contoh

```
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-
bucket', 'sample-filepath');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-
bucket', 'sample-filepath','us-west-2');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-
bucket', 'sample-filepath','us-west-2','format text');
```

aws_commons.create_s3_uri

Membuat struktur `aws_commons._s3_uri_1` untuk menyimpan informasi file Amazon S3. Gunakan hasil dari fungsi `aws_commons.create_s3_uri` dalam parameter `s3_info` dari fungsi aws_s3.query_export_to_s3. Untuk contoh penggunaan fungsi `aws_commons.create_s3_uri`, lihat Menentukan jalur file Amazon S3 tujuan ekspor.

Sintaksis

```
aws_commons.create_s3_uri(
   bucket text,
```

```
    file_path text,
    region text
)
```

Parameter input

bucket

String teks yang diperlukan yang berisi nama bucket Amazon S3 untuk file tersebut.

file_path

String teks yang diperlukan yang berisi nama file Amazon S3 beserta jalurnya.

region

String teks yang diperlukan yang berisi AWS Wilayah tempat file tersebut berada. Untuk daftar nama AWS Wilayah dan nilai terkait, lihatWilayah dan Zona Ketersediaan.

AWS Lambda adalah layanan komputasi berbasis peristiwa yang memungkinkan Anda menjalankan kode tanpa menyediakan atau mengelola server. Misalnya, Anda dapat menggunakan fungsi Lambda untuk memproses pemberitahuan peristiwa dari basis data, atau memuat data dari file setiap kali file baru diunggah ke Amazon S3. Untuk mempelajari lebih lanjut tentang Lambda, lihat Apa itu? AWS Lambda di Panduan AWS Lambda Pengembang.

> **Note**
>
> AWS Lambda Fungsi pemanggilan didukung di Aurora PostgreSQL 11.9 dan yang lebih tinggi (termasuk). Aurora Serverless v2

Berikut ini, Anda dapat menemukan ringkasan langkah-langkah yang diperlukan.

Untuk informasi selengkapnya tentang fungsi Lambda, lihat Mulai menggunakan Lambda dan Dasar-dasar AWS Lambda di Panduan Developer AWS Lambda .

Topik

- Langkah 1: Konfigurasikan Aurora PostgreSQL DB cluster Anda untuk instance PostgreSQL DB untuk koneksi keluar ke AWS Lambda
- AWS Lambda

- Langkah 3: Instal ekstensi aws_lambda untuk klaster DB Aurora PostgreSQL
- Langkah 4: Gunakan fungsi pembantu Lambda dengan klaster DB Aurora PostgreSQL (Opsional)
- Langkah 5: Invokasi fungsi Lambda dari klaster DB Aurora PostgreSQL Anda
- Langkah 6: Berikan pengguna lain izin untuk menginvokasi fungsi Lambda
- Contoh: Menginvokasi fungsi Lambda dari klaster DB Aurora PostgreSQL
- Pesan kesalahan fungsi Lambda
- AWS Lambdafungsi dan referensi parameter

Langkah 1: Konfigurasikan Aurora PostgreSQL DB cluster Anda untuk instance PostgreSQL DB untuk koneksi keluar ke AWS Lambda

Fungsi Lambda selalu berjalan di dalam VPC Amazon yang dimiliki oleh layanan. AWS Lambda Lambda menerapkan akses jaringan dan aturan keamanan untuk VPC ini dan mempertahankan serta memantau VPC secara otomatis. Klaster DB Aurora PostgreSQL Anda mengirimkan lalu lintas jaringan ke VPC layanan Lambda. Cara Anda mengonfigurasi ini bergantung pada apakah instans DB primer klaster DB Aurora Anda bersifat publik atau pribadi.

- Public Aurora PostgreSQL DB cluster utama cluster DB adalah publik jika terletak di subnet publik di VPC Anda, dan jika properti “” adalah. PubliclyAccessible `true` Untuk menemukan nilai properti ini, Anda dapat menggunakan describe-db-instances AWS CLI perintah. Atau, Anda dapat menggunakan AWS Management Console untuk membuka tab Konektivitas & keamanan dan memeriksa apakah opsi Dapat diakses publik adalah Ya. Untuk memverifikasi bahwa instans ini ada di subnet publik VPC, Anda dapat menggunakan AWS Management Console atau AWS CLI.

  Untuk mengatur akses ke Lambda, Anda menggunakan AWS Management Console atau AWS CLI untuk membuat aturan keluar pada grup keamanan VPC Anda. Aturan outbound menentukan bahwa TCP dapat menggunakan port 443 untuk mengirim paket ke alamat IPv4 (0.0.0.0/0) mana pun.
- Private Aurora PostgreSQL DB cluster “” instance ada atau berada di subnet pribadi. PubliclyAccessible `false` Agar instans dapat berfungsi dengan Lambda, Anda dapat menggunakan gateway Network Address Translation (NAT). Untuk informasi selengkapnya, silakan lihat Gateway NAT. Atau, Anda dapat mengonfigurasi VPC dengan titik akhir VPC untuk Lambda. Untuk informasi selengkapnya, lihat Titik akhir VPC di Panduan Pengguna Amazon VPC. Titik akhir ini merespons panggilan yang dilakukan oleh klaster DB Aurora PostgreSQL ke fungsi Lambda Anda.

VPC Anda sekarang dapat berinteraksi dengan AWS Lambda VPC di tingkat jaringan. Selanjutnya, konfigurasikan izin menggunakan IAM.

AWS Lambda

Menginvokasi fungsi Lambda dari klaster DB Aurora PostgreSQL memerlukan hak istimewa tertentu. Untuk mengonfigurasi hak istimewa yang diperlukan, sebaiknya Anda membuat kebijakan IAM yang memungkinkan invokasi fungsi Lambda, menetapkan kebijakan ke peran, dan kemudian menerapkan peran tersebut ke klaster DB. Pendekatan ini memberikan hak istimewa klaster DB untuk menginvokasi fungsi Lambda yang ditentukan atas nama Anda. Langkah-langkah berikut menunjukkan cara melakukannya dengan menggunakan AWS CLI.

Untuk mengonfigurasi izin IAM untuk menggunakan klaster dengan Lambda

1. (ID pernyataan (Sid) adalah deskripsi opsional untuk pernyataan kebijakan Anda dan tidak berpengaruh pada penggunaan.) Kebijakan ini memberi klaster DB Aurora izin minimum yang diperlukan untuk menginvokasi fungsi Lambda yang ditentukan.

   ```
   aws iam create-policy  --policy-name rds-lambda-policy --policy-document '{
       "Version": "2012-10-17",
       "Statement": [
           {
           "Sid": "AllowAccessToExampleFunction",
           "Effect": "Allow",
           "Action": "lambda:InvokeFunction",
           "Resource": "arn:aws:lambda:aws-region:444455556666:function:my-function"
           }
       ]
   }'
   ```

   Sebagai alternatif, Anda dapat menggunakan kebijakan `AWSLambdaRole` yang ditentukan sebelumnya yang memungkinkan Anda menginvokasi fungsi Lambda apa pun. Untuk informasi selengkapnya, lihat Kebijakan IAM berbasis identitas untuk Lambda

2. Gunakan AWS CLI perintah create-role untuk membuat peran IAM yang dapat diasumsikan oleh kebijakan saat runtime.

   ```
   aws iam create-role  --role-name rds-lambda-role --assume-role-policy-document '{
       "Version": "2012-10-17",
       "Statement": [
           {
   ```

```
        "Effect": "Allow",
        "Principal": {
            "Service": "rds.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
}'
```

3. Terapkan kebijakan ke peran dengan menggunakan attach-role-policy AWS CLI perintah.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::444455556666:policy/rds-lambda-policy \
    --role-name rds-lambda-role --region aws-region
```

4. https://awscli.amazonaws.com/v2/documentation/api/latest/reference/rds/add-role-to-db-cluster.html AWS CLI Langkah terakhir ini memungkinkan pengguna basis data klaster DB menginvokasi fungsi Lambda.

```
aws rds add-role-to-db-cluster \
       --db-cluster-identifier my-cluster-name \
       --feature-name Lambda \
       --role-arn  arn:aws:iam::444455556666:role/rds-lambda-role   \
       --region aws-region
```

Setelah menyelesaikan konfigurasi VPC dan IAM, Anda sekarang dapat menginstal ekstensi `aws_lambda`. (Perhatikan bahwa Anda dapat menginstal ekstensi kapan saja, tetapi sebelum menyiapkan dukungan VPC dan hak istimewa IAM yang benar, ekstensi `aws_lambda` tidak menambahkan apa pun ke kapabilitas klaster DB Aurora PostgreSQL.)

Langkah 3: Instal ekstensi **`aws_lambda`** untuk klaster DB Aurora PostgreSQL

Ekstensi ini memberi klaster DB Aurora PostgreSQL kemampuan untuk memanggil fungsi Lambda dari PostgreSQL.

Untuk menginstal ekstensi **`aws_lambda`** di klaster DB Aurora PostgreSQL Anda

Gunakan baris perintah `psql` PostgreSQL atau alat pgAdmin untuk terhubung ke klaster DB Aurora PostgreSQL .

1. Hubungkan ke klaster DB Aurora PostgreSQL sebagai pengguna dengan hak istimewa `rds_superuser`. Pengguna `postgres` default ditampilkan dalam contoh.

```
psql -h cluster-instance.444455556666.aws-region.rds.amazonaws.com -U postgres -p
 5432
```

2. Instal ekstensi `aws_lambda`. Ekstensi `aws_commons` juga diperlukan. Ini memberikan fungsi pembantu `aws_lambda` dan berbagai ekstensi Aurora lainnya untuk PostgreSQL. Jika belum ada di klaster DB Aurora PostgreSQL , ekstensi akan diinstal dengan `aws_lambda` seperti yang ditunjukkan sebagai berikut.

```
CREATE EXTENSION IF NOT EXISTS aws_lambda CASCADE;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
```

Ekstensi `aws_lambda` diinstal di instans DB primer klaster DB Aurora PostgreSQL. Anda sekarang dapat membuat struktur kemudahan untuk menginvokasi fungsi Lambda.

Langkah 4: Gunakan fungsi pembantu Lambda dengan klaster DB Aurora PostgreSQL (Opsional)

Anda dapat menggunakan fungsi pembantu di ekstensi `aws_commons` untuk menyiapkan entitas yang dapat diinvokasi dengan lebih mudah dari PostgreSQL. Untuk melakukannya, Anda harus memiliki informasi berikut tentang fungsi Lambda:

- Nama fungsi – Nama, Amazon Resource Name (ARN), versi, atau alias fungsi Lambda. Kebijakan IAM yang dibuat Langkah 2: Konfigurasikan IAM untuk klaster dan Lambda memerlukan ARN, jadi kami sarankan Anda menggunakan ARN fungsi Anda.
- AWS Wilayah

Untuk menyimpan informasi nama fungsi Lambda, Anda menggunakan fungsi aws_commons.create_lambda_function_arn. Fungsi pembantu ini menciptakan struktur komposit `aws_commons._lambda_function_arn_1` dengan detail yang dibutuhkan oleh fungsi invokasi. Berikut ini, Anda dapat menemukan tiga pendekatan alternatif untuk menyiapkan struktur komposit ini.

```
SELECT aws_commons.create_lambda_function_arn(
```

```
   'my-function',
   'aws-region'
) AS aws_lambda_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   '111122223333:function:my-function',
   'aws-region'
) AS lambda_partial_arn_1 \gset
```

```
SELECT aws_commons.create_lambda_function_arn(
   'arn:aws:lambda:aws-region:111122223333:function:my-function'
) AS lambda_arn_1 \gset
```

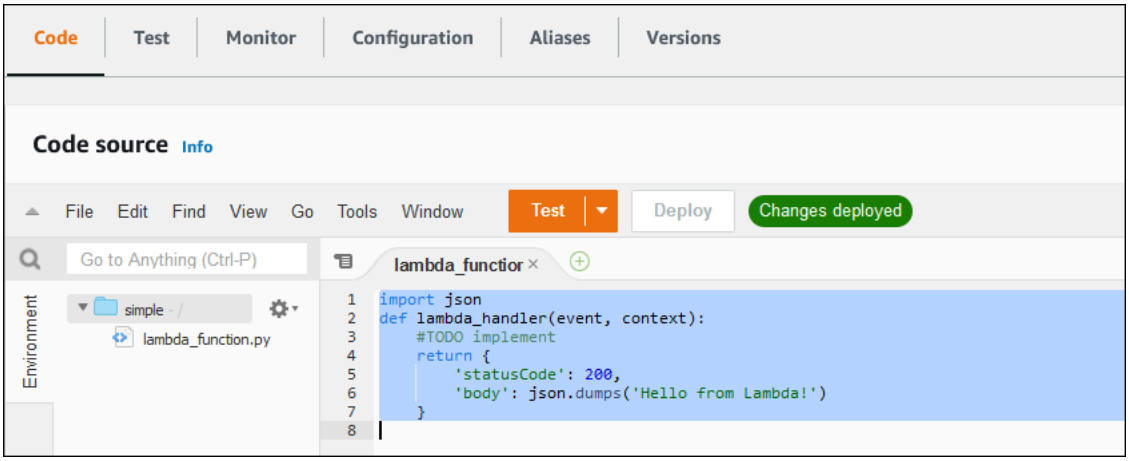
Salah satu dari nilai ini dapat digunakan dalam panggilan ke fungsi aws_lambda.invoke. Sebagai contoh, lihat Langkah 5: Invokasi fungsi Lambda dari klaster DB Aurora PostgreSQL Anda.

Langkah 5: Invokasi fungsi Lambda dari klaster DB Aurora PostgreSQL Anda

Fungsi `aws_lambda.invoke` berperilaku sinkron atau asinkron, bergantung pada `invocation_type`. Dua alternatif untuk parameter ini adalah `RequestResponse` (default) dan `Event`, sebagai berikut.

- **`RequestResponse`** – Jenis invokasi ini sinkron. Ini adalah perilaku default saat panggilan dilakukan tanpa menentukan jenis invokasi. Payload respons mencakup hasil dari fungsi `aws_lambda.invoke`. Gunakan jenis invokasi ini jika alur kerja Anda perlu menerima hasil dari fungsi Lambda sebelum melanjutkan.
- **`Event`** – Jenis invokasi ini asinkron. Respons tidak mencakup payload yang berisi hasil. Gunakan jenis invokasi ini jika alur kerja Anda tidak memerlukan hasil dari fungsi Lambda untuk melanjutkan pemrosesan.

Sebagai pengujian sederhana terhadap pengaturan Anda, Anda dapat terhubung ke instans DB menggunakan `psql` dan menginvokasi contoh fungsi dari baris perintah. Misalkan Anda memiliki salah satu fungsi dasar yang disiapkan pada layanan Lambda, seperti fungsi Python sederhana yang diperlihatkan pada tangkapan layar berikut.



Untuk menginvokasi contoh fungsi

1. Hubungkan ke instans DB primer menggunakan `psql` atau pgAdmin.

```
psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. Invokasi fungsi menggunakan ARN-nya.

```
SELECT * from
 aws_lambda.invoke(aws_commons.create_lambda_function_arn('arn:aws:lambda:aws-
region:444455556666:function:simple', 'us-west-1'), '{"body": "Hello from
 Postgres!"}'::json );
```

   Respons-nya terlihat sebagai berikut.

```
status_code |                         payload                         |
 executed_version | log_result
-------------+---------------------------------------------------------
+------------------+------------
         200 | {"statusCode": 200, "body": "\"Hello from Lambda!\""} | $LATEST
     |
(1 row)
```

Jika upaya invokasi tidak berhasil, lihat Pesan kesalahan fungsi Lambda .

Langkah 6: Berikan pengguna lain izin untuk menginvokasi fungsi Lambda

Dalam langkah ini, hanya Anda sebagai `rds_superuser` yang dapat menginvokasi fungsi Lambda. Untuk mengizinkan pengguna lain menginvokasi fungsi apa pun yang Anda buat, Anda harus memberi mereka izin.

Untuk memberi pengguna lain izin untuk menginvokasi fungsi Lambda

1. Hubungkan ke instans DB primer menggunakan `psql` atau pgAdmin.

```
psql -h cluster.444455556666.aws-region.rds.amazonaws.com -U postgres -p 5432
```

2. Jalankan perintah SQL berikut:

```
postgres=>  GRANT USAGE ON SCHEMA aws_lambda TO db_username;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA aws_lambda TO db_username;
```

Contoh: Menginvokasi fungsi Lambda dari klaster DB Aurora PostgreSQL

Berikut ini, Anda dapat menemukan beberapa contoh pemanggilan fungsi aws_lambda.invoke. Sebagian besar contoh menggunakan struktur komposit `aws_lambda_arn_1` yang Anda buat di Langkah 4: Gunakan fungsi pembantu Lambda dengan klaster DB Aurora PostgreSQL (Opsional) untuk menyederhanakan penerusan detail fungsi. Untuk contoh panggilan asinkron, lihat Contoh: Invokasi fungsi Lambda asinkron (Event). Semua contoh lain yang tercantum menggunakan panggilan sinkron.

Untuk mempelajari lebih lanjut tentang jenis invokasi Lambda, lihat Menginvokasi fungsi Lambda di Panduan Developer AWS Lambda . Untuk informasi selengkapnya tentang `aws_lambda_arn_1`, lihat aws_commons.create_lambda_function_arn.

Daftar contoh

- Contoh: Synchronous (RequestResponse) pemanggilan fungsi Lambda
- Contoh: Invokasi fungsi Lambda asinkron (Event)
- Contoh: Menangkap log eksekusi Lambda dalam respons fungsi
- Contoh: Menyertakan konteks klien dalam fungsi Lambda
- Contoh: Menginvokasi fungsi Lambda versi spesifik

Contoh: Synchronous (RequestResponse) pemanggilan fungsi Lambda

Berikut ini adalah dua contoh dari invokasi fungsi Lambda sinkron. Hasil dari panggilan fungsi `aws_lambda.invoke` ini sama.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
```

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse');
```

Parameternya dijelaskan sebagai berikut:

- `:'aws_lambda_arn_1'` – Parameter ini mengidentifikasi struktur komposit yang dibuat di[Langkah 4: Gunakan fungsi pembantu Lambda dengan klaster DB Aurora PostgreSQL (Opsional)](), dengan fungsi pembantu `aws_commons.create_lambda_function_arn`. Anda juga dapat membuat struktur ini sebaris dalam panggilan `aws_lambda.invoke` Anda sebagai berikut.

  ```
  SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-function',
   'aws-region'),
  '{"body": "Hello from Postgres!"}'::json
  );
  ```

- `'{"body": "Hello from PostgreSQL!"}'::json` – Payload JSON untuk diteruskan ke fungsi Lambda.
- `'RequestResponse'` – Jenis invokasi Lambda.

Contoh: Invokasi fungsi Lambda asinkron (Event)

Berikut ini adalah contoh invokasi fungsi Lambda asinkron. Jenis invokasi `Event` menjadwalkan invokasi fungsi Lambda dengan payload input yang ditentukan dan segera kembali. Gunakan jenis invokasi `Event` di alur kerja tertentu yang tidak bergantung pada hasil fungsi Lambda.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'Event');
```

Contoh: Menangkap log eksekusi Lambda dalam respons fungsi

Anda dapat menyertakan 4 KB terakhir log eksekusi di respons fungsi dengan menggunakan parameter `log_type` dalam panggilan fungsi `aws_lambda.invoke` Anda. Secara default, parameter ini diatur ke `None`, tetapi Anda dapat menentukan `Tail` untuk menangkap hasil log eksekusi Lambda dalam respons, seperti yang ditunjukkan berikut.

```
SELECT *, select convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

Atur parameter `log_type` fungsi aws_lambda.invoke ke `Tail` untuk menyertakan log eksekusi dalam respons. Nilai default untuk parameter `log_type` adalah `None`.

`log_result` yang ditampilkan string yang dienkode `base64`. Anda dapat mendekode kontennya menggunakan kombinasi fungsi PostgreSQL `decode` dan `convert_from`.

Untuk informasi selengkapnya tentang `log_type`, lihat aws_lambda.invoke.

Contoh: Menyertakan konteks klien dalam fungsi Lambda

Fungsi `aws_lambda.invoke` memiliki parameter `context` yang dapat Anda gunakan untuk meneruskan informasi yang terpisah dari payload, seperti yang diperlihatkan di bawah.

```
SELECT *, convert_from(decode(log_result, 'base64'), 'utf-8') as log FROM
 aws_lambda.invoke(:'aws_lambda_arn_1', '{"body": "Hello from Postgres!"}'::json,
 'RequestResponse', 'Tail');
```

Untuk menyertakan konteks klien, gunakan objek JSON untuk parameter `context` fungsi aws_lambda.invoke.

Untuk informasi selengkapnya tentang parameter `context`, lihat referensi aws_lambda.invoke.

Contoh: Menginvokasi fungsi Lambda versi spesifik

Anda dapat menentukan versi tertentu dari fungsi Lambda dengan menyertakan parameter `qualifier` dengan panggilan `aws_lambda.invoke`. Berikut ini, Anda dapat menemukan contoh yang melakukan ini menggunakan '*`custom_version`*' sebagai alias untuk versi.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json, 'RequestResponse', 'None', NULL, 'custom_version');
```

Anda juga dapat menyediakan pengualifikasi fungsi Lambda dengan detail nama fungsi sebagai berikut.

```
SELECT * FROM aws_lambda.invoke(aws_commons.create_lambda_function_arn('my-
function:custom_version', 'us-west-2'),
'{"body": "Hello from Postgres!"}'::json);
```

Untuk informasi selengkapnya tentang `qualifier` dan parameter lainnya, lihat referensi aws_lambda.invoke.

Pesan kesalahan fungsi Lambda

Dalam daftar berikut, Anda dapat menemukan informasi tentang pesan kesalahan, dengan kemungkinan penyebab dan solusi.

- Masalah konfigurasi VPC

  Masalah konfigurasi VPC dapat memunculkan pesan kesalahan berikut saat mencoba menghubungkan:

  ```
  ERROR:  invoke API failed
  DETAIL: AWS Lambda client returned 'Unable to connect to endpoint'.
  CONTEXT:  SQL function "invoke" statement 1
  ```

  Penyebab umum kesalahan ini adalah grup keamanan VPC tidak dikonfigurasi dengan benar. Pastikan Anda memiliki aturan keluar untuk TCP yang terbuka pada di 443 grup keamanan VPC Anda, sehingga VPC Anda dapat terhubung ke VPC Lambda.
- Kurangnya izin yang diperlukan untuk menginvokasi fungsi Lambda

  Jika Anda melihat salah satu pesan galat berikut, berarti pengguna (peran) yang menginvokasi fungsi ini tidak memiliki izin yang tepat.

  ```
  ERROR:  permission denied for schema aws_lambda
  ```

  ```
  ERROR:  permission denied for function invoke
  ```

  Pengguna (peran) harus diberi izin khusus untuk menginvokasi fungsi Lambda. Untuk informasi selengkapnya, lihat Langkah 6: Berikan pengguna lain izin untuk menginvokasi fungsi Lambda.
- Penanganan kesalahan yang tidak tepat dalam fungsi Lambda

Jika fungsi Lambda menampilkan pengecualian selama pemrosesan permintaan, berarti `aws_lambda.invoke` gagal dengan kesalahan PostgreSQL seperti berikut.

```
SELECT * FROM aws_lambda.invoke('aws_lambda_arn_1', '{"body": "Hello from
 Postgres!"}'::json);
ERROR:  lambda invocation failed
DETAIL:  "arn:aws:lambda:us-west-2:555555555555:function:my-function" returned error
 "Unhandled", details: "<Error details string>".
```

Pastikan untuk menangani kesalahan dalam fungsi Lambda Anda atau di aplikasi PostgreSQL Anda.

AWS Lambdafungsi dan referensi parameter

Berikut ini adalah referensi untuk fungsi dan parameter yang akan digunakan untuk memanggil Lambda dengan Aurora .

Fungsi dan parameter

- aws_lambda.invoke
- aws_commons.create_lambda_function_arn
- parameter aws_lambda

aws_lambda.invoke

Menjalankan fungsi Lambda untuk klaster DB Aurora PostgreSQL .

Untuk detail lebih lanjut tentang memanggil fungsi Lambda, lihat juga Invokasi di Panduan Developer AWS Lambda.

Sintaksis

JSON

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSON,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
```

```
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSON,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSON DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSON,
OUT executed_version TEXT,
OUT log_result TEXT)
```

JSONB

```
aws_lambda.invoke(
IN function_name TEXT,
IN payload JSONB,
IN region TEXT DEFAULT NULL,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT)
```

```
aws_lambda.invoke(
IN function_name aws_commons._lambda_function_arn_1,
IN payload JSONB,
IN invocation_type TEXT DEFAULT 'RequestResponse',
IN log_type TEXT DEFAULT 'None',
IN context JSONB DEFAULT NULL,
IN qualifier VARCHAR(128) DEFAULT NULL,
```

```
OUT status_code INT,
OUT payload JSONB,
OUT executed_version TEXT,
OUT log_result TEXT
)
```

Parameter input

function_name

Nama yang mengidentifikasi fungsi Lambda. Nilai tersebut dapat berupa nama fungsi, sebuah ARN, atau ARN parsial. Untuk daftar format yang memungkinkan, lihat Format nama fungsi Lambda dalam Panduan Developer AWS Lambda.

payload

Input untuk fungsi Lambda. Formatnya dapat berupa JSON atau JSONB. Untuk informasi selengkapnya, lihat Jenis JSON dalam dokumentasi PostgreSQL.

region

(Opsional) Wilayah Lambda untuk fungsi tersebut. Secara default, Aurora menyelesaikan Wilayah AWS dari ARN penuh di `function_name` atau menggunakan Wilayah instans DB Aurora PostgreSQL . Jika nilai Wilayah ini bertentangan dengan nilai yang disediakan dalam ARN `function_name`, pesan kesalahan akan muncul.

invocation_type

Jenis invokasi fungsi Lambda. Nilai ini peka huruf besar/kecil. Kemungkinan nilainya termasuk yang berikut ini:

- `RequestResponse` – Default. Jenis invokasi untuk fungsi Lambda bersifat sinkron dan menampilkan payload respons dalam hasilnya. Gunakan jenis invokasi `RequestResponse` ketika alur kerja Anda bergantung pada penerimaan hasil fungsi Lambda dengan segera.
- `Event` – Jenis invokasi untuk fungsi Lambda ini bersifat asinkron dan segera kembali tanpa menampilkan payload. Gunakan jenis invokasi `Event` ketika Anda tidak membutuhkan hasil dari fungsi Lambda sebelum alur kerja Anda berlanjut.
- `DryRun` – Jenis invokasi ini menguji akses tanpa menjalankan fungsi Lambda.

log_type

Jenis log Lambda untuk ditampilkan dalam parameter output `log_result`. Nilai ini peka huruf besar/kecil. Kemungkinan nilainya termasuk yang berikut ini:

- Ekor – Parameter output `log_result` yang ditampilkan akan mencakup 4 KB terakhir log eksekusi.
- Tidak Ada – Tidak ada informasi log Lambda yang ditampilkan.

context

Konteks klien dalam format JSON atau JSONB. Kolom yang akan digunakan termasuk `custom` dan `env`.

qualifier

Pengualifikasi yang mengidentifikasi versi fungsi Lambda yang akan diinvokasi. Jika nilai ini bertentangan dengan nilai yang disediakan dalam ARN `function_name`, pesan kesalahan akan muncul.

Parameter output

status_code

Kode respons status HTTP. Untuk informasi selengkapnya, lihat Elemen respons invokasi Lambda di Panduan Developer AWS Lambda.

payload

Informasi yang ditampilkan dari fungsi Lambda yang berjalan. Formatnya berupa JSON atau JSONB.

executed_version

Versi fungsi Lambda yang berjalan.

log_result

Informasi log eksekusi yang ditampilkan jika nilai `log_type` adalah `Tail` ketika fungsi Lambda diinvokasi. Hasilnya berisi 4 KB terakhir log eksekusi yang dikodekan dalam Base64.

aws_commons.create_lambda_function_arn

Membuat struktur `aws_commons._lambda_function_arn_1` untuk menyimpan informasi nama fungsi Lambda. Gunakan hasil fungsi `aws_commons.create_lambda_function_arn` dalam parameter `function_name` dari fungsi aws_lambda.invoke aws_lambda.invoke.

Sintaksis

```
aws_commons.create_lambda_function_arn(
    function_name TEXT,
    region TEXT DEFAULT NULL
    )
    RETURNS aws_commons._lambda_function_arn_1
```

Parameter input

function_name

String teks yang diperlukan berisi nama fungsi Lambda. Nilai tersebut dapat berupa nama fungsi, ARN penuh, atau ARN parsial.

region

String teks opsional yang berisi Wilayah AWS tempat fungsi Lambda berada. Untuk daftar nama Wilayah dan nilai terkait, lihat Wilayah dan Zona Ketersediaan.

parameter aws_lambda

Dalam tabel ini, Anda dapat menemukan parameter yang terkait dengan `aws_lambda` fungsi tersebut.

Parameter	Deskripsi
`aws_lambda.connect_timeout_ms`	Ini adalah parameter dinamis dan menetapkan waktu tunggu maksimum saat menghubungkan ke AWS Lambda. Nilai defaultnya adalah`1000`. Nilai yang diizinkan untuk parameter ini adalah 1 - 900000.
`aws_lambda.request_timeout_ms`	Ini adalah parameter dinamis dan menetapkan waktu tunggu maksimum sambil menunggu respons dari AWS Lambda. Nilai defaultnya adalah`3000`. Nilai yang diizinkan untuk parameter ini adalah 1 - 900000.
`aws_lambda.endpoint_override`	Menentukan endpoint yang dapat digunakan untuk terhubung ke LambdaAWS. String kosong memilih titik akhir AWS Lambda default untuk wilayah tersebut. Anda harus me-restart database agar perubahan parameter statis ini berlaku.

Menerbitkan log Aurora PostgreSQL ke Amazon Logs CloudWatch

Anda dapat mengonfigurasi klaster DB PostgreSQL Aurora untuk mengekspor data log ke Amazon Logs secara teratur. CloudWatch Ketika Anda melakukannya, peristiwa dari log PostgreSQL cluster PostgreSQL Aurora PostgreSQL DB Anda secara otomatis dipublikasikan ke Amazon, sebagai Amazon Logs. CloudWatch CloudWatch Di CloudWatch, Anda dapat menemukan data log yang diekspor dalam grup Log untuk cluster DB Aurora PostgreSQL Anda. Grup log berisi satu atau beberapa log stream yang berisi peristiwa dari log PostgreSQL dari setiap instans di klaster.

Menerbitkan CloudWatch log ke Log memungkinkan Anda menyimpan catatan log PostgreSQL klaster Anda dalam penyimpanan yang sangat tahan lama. Dengan data log yang tersedia di CloudWatch Log, Anda dapat mengevaluasi dan meningkatkan operasi klaster Anda. Anda juga dapat menggunakan CloudWatch untuk membuat alarm dan melihat metrik. Untuk mempelajari selengkapnya, lihat Memantau peristiwa log di Amazon CloudWatch.

> **Note**
>
> Menerbitkan log PostgreSQL Anda CloudWatch ke Log menghabiskan penyimpanan, dan Anda dikenakan biaya untuk penyimpanan itu. Pastikan untuk menghapus CloudWatch Log apa pun yang tidak lagi Anda butuhkan.

Menonaktifkan opsi log ekspor untuk klaster Aurora PostgreSQL DB yang ada tidak memengaruhi data apa pun yang sudah disimpan di Log. CloudWatch Log yang ada tetap tersedia di CloudWatch Log berdasarkan pengaturan penyimpanan log Anda. Untuk mempelajari lebih lanjut tentang CloudWatch Log, lihat Apa itu CloudWatch Log Amazon?

Aurora PostgreSQL mendukung penerbitan log ke Log untuk versi berikut. CloudWatch

- 14.3 dan versi 14 yang lebih tinggi
- 13.3 dan 13 versi yang lebih tinggi
- 12.8 dan versi 12 yang lebih tinggi
- 11.12 dan versi 11 yang lebih tinggi

Mengaktifkan opsi untuk menerbitkan log ke Amazon CloudWatch

Untuk mempublikasikan log PostgreSQL klaster Aurora PostgreSQL DB Anda ke Log, pilih opsi ekspor Log untuk klaster. CloudWatch Anda dapat memilih pengaturan Ekspor log saat Anda

membuat klaster DB Aurora PostgreSQL. Atau, Anda dapat mengubah klaster ini nanti. Saat Anda memodifikasi klaster yang ada, log PostgreSQL dari setiap instance dipublikasikan CloudWatch ke cluster sejak saat itu. Untuk Aurora PostgreSQL, PostgreSQL log () adalah satu-satunya log yang dipublikasikan ke Amazon. `postgresql.log` CloudWatch

Anda dapat menggunakan AWS Management Console, AWS CLI, atau API RDS untuk mengaktifkan fitur Ekspor log untuk klaster DB Aurora PostgreSQL Anda.

Konsol

Anda memilih opsi ekspor Log untuk mulai menerbitkan log PostgreSQL dari cluster Aurora PostgreSQL DB Anda ke Log. CloudWatch

Untuk mengaktifkan fitur Ekspor log dari konsol

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB PostgreSQL Aurora yang data lognya ingin Anda publikasikan ke Log. CloudWatch
4. Pilih Ubah.
5. Di bagian Ekspor log, pilih Log PostgreSQL.
6. Pilih Lanjutkan, lalu pilih Ubah klaster di halaman ringkasan.

AWS CLI

Anda dapat mengaktifkan opsi ekspor log untuk mulai menerbitkan log Aurora PostgreSQL ke Amazon Logs dengan file. CloudWatch AWS CLI Untuk melakukannya, jalankan modify-db-clusterAWS CLIperintah dengan opsi berikut:

- `--db-cluster-identifier` – Pengidentifikasi klaster DB.
- `--cloudwatch-logs-export-configuration`—Pengaturan konfigurasi untuk jenis log yang akan disetel untuk diekspor ke CloudWatch Log untuk cluster DB.

Anda juga dapat menerbitkan log Aurora PostgreSQL dengan menjalankan salah satu dari perintah AWS CLI berikut:

- create-db-cluster

- restore-db-cluster-from-s3
- restore-db-cluster-from-snapshot
- restore-db-cluster-to-point-in-time

Jalankan salah satu perintah AWS CLI ini dengan opsi berikut:

- `--db-cluster-identifier` – Pengidentifikasi klaster DB.
- `--engine` – Mesin basis data.
- `--enable-cloudwatch-logs-exports`—Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk diekspor ke CloudWatch Log untuk cluster DB.

Opsi lain mungkin diperlukan tergantung pada perintah AWS CLI yang Anda jalankan.

Example

Perintah berikut membuat cluster Aurora PostgreSQL DB untuk mempublikasikan file log ke Log. CloudWatch

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --engine aurora-postgresql \
    --enable-cloudwatch-logs-exports postgresql
```

Untuk Windows:

```
aws rds create-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --engine aurora-postgresql ^
    --enable-cloudwatch-logs-exports postgresql
```

Example

Perintah berikut memodifikasi cluster Aurora PostgreSQL DB yang ada untuk mempublikasikan file log ke Log. CloudWatch Nilai `--cloudwatch-logs-export-configuration` adalah objek JSON. Kunci untuk objek ini adalah `EnableLogTypes`, dan nilainya adalah `postgresql`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier my-db-cluster \
    --cloudwatch-logs-export-configuration '{"EnableLogTypes":["postgresql"]}'
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier my-db-cluster ^
    --cloudwatch-logs-export-configuration '{\"EnableLogTypes\":[\"postgresql\"]}'
```

> **Note**
>
> Saat menggunakan command prompt Windows, pastikan untuk meng-escape tanda kutip ganda (“) dalam kode JSON dengan memberikan garis miring terbalik (\) di depannya.

Example

Contoh berikut memodifikasi cluster Aurora PostgreSQL DB yang ada untuk menonaktifkan penerbitan file log ke Log. CloudWatch Nilai `--cloudwatch-logs-export-configuration` adalah objek JSON. Kunci untuk objek ini adalah `DisableLogTypes`, dan nilainya adalah `postgresql`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbinstance \
    --cloudwatch-logs-export-configuration '{"DisableLogTypes":["postgresql"]}'
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbinstance ^
    --cloudwatch-logs-export-configuration "{\"DisableLogTypes\":[\"postgresql\"]}"
```

> **Note**
>
> Saat menggunakan command prompt Windows, Anda harus meng-escape tanda kutip ganda (") dalam kode JSON dengan memberikan garis miring terbalik (\) di depannya.

API RDS

Anda dapat mengaktifkan opsi ekspor log untuk mulai menerbitkan log Aurora PostgreSQL dengan API RDS. Untuk melakukannya, jalankan operasi ModifyDBCluster dengan opsi berikut:

- `DBClusterIdentifier` – Pengidentifikasi klaster DB.
- `CloudwatchLogsExportConfiguration`— Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk ekspor ke CloudWatch Log untuk cluster DB.

Anda juga dapat menerbitkan log Aurora PostgreSQL dengan API RDS dengan menjalankan salah satu dari operasi API RDS berikut:

- CreateDBCluster
- DipulihkanB S3 ClusterFrom
- DipulihkanB ClusterFromSnapshot
- DipulihkanB ClusterToPointInTime

Jalankan tindakan API RDS dengan parameter berikut:

- `DBClusterIdentifier` – Pengidentifikasi klaster DB.
- `Engine` – Mesin basis data.
- `EnableCloudwatchLogsExports`—Pengaturan konfigurasi untuk jenis log yang akan diaktifkan untuk diekspor ke CloudWatch Log untuk cluster DB.

Parameter lain mungkin diperlukan tergantung pada perintah AWS CLI yang Anda jalankan.

Memantau peristiwa log di Amazon CloudWatch

Dengan peristiwa log Aurora PostgreSQL yang diterbitkan dan tersedia sebagai Amazon CloudWatch Logs, Anda dapat melihat dan memantau peristiwa menggunakan Amazon. CloudWatch Untuk informasi selengkapnya tentang pemantauan, lihat Melihat data log yang dikirim ke CloudWatch Log.

Saat Anda mengaktifkan Ekspor log, grup log baru secara otomatis dibuat menggunakan awalan `/aws/rds/cluster/` dengan nama Aurora PostgreSQL Anda dan jenis log, seperti dalam pola berikut.

```
/aws/rds/cluster/your-cluster-name/postgresql
```

Sebagai contoh, misalkan cluster Aurora PostgreSQL DB bernama mengekspor lognya ke Amazon Logs. `docs-lab-apg-small` CloudWatch Nama grup lognya di Amazon CloudWatch ditampilkan sebagai berikut.

```
/aws/rds/cluster/docs-lab-apg-small/postgresql
```

Jika ada grup log yang sudah memiliki nama yang ditentukan, Aurora akan menggunakan grup log tersebut untuk mengekspor data log untuk klaster DB Aurora. Setiap instans DB di klaster DB Aurora PostgreSQL mengunggah log PostgreSQL-nya ke grup log sebagai log stream unik. Anda dapat memeriksa grup log dan aliran lognya menggunakan berbagai alat grafis dan analitis yang tersedia di Amazon CloudWatch.

Misalnya, Anda dapat mencari informasi dalam peristiwa log dari klaster Aurora PostgreSQL DB, dan memfilter peristiwa dengan menggunakan konsol CloudWatch Log, API, atau Logs. AWS CLI CloudWatch Untuk informasi selengkapnya, Mencari dan memfilter data log di Panduan Pengguna Amazon CloudWatch Logs.

Secara default, grup log baru dibuat menggunakan Tidak pernah kedaluwarsa untuk periode retensinya. Anda dapat menggunakan konsol CloudWatch Logs, theAWS CLI, atau CloudWatch Logs API untuk mengubah periode penyimpanan log. Untuk mempelajari selengkapnya, lihat Mengubah penyimpanan data CloudWatch log di Log di Panduan Pengguna CloudWatch Log Amazon.

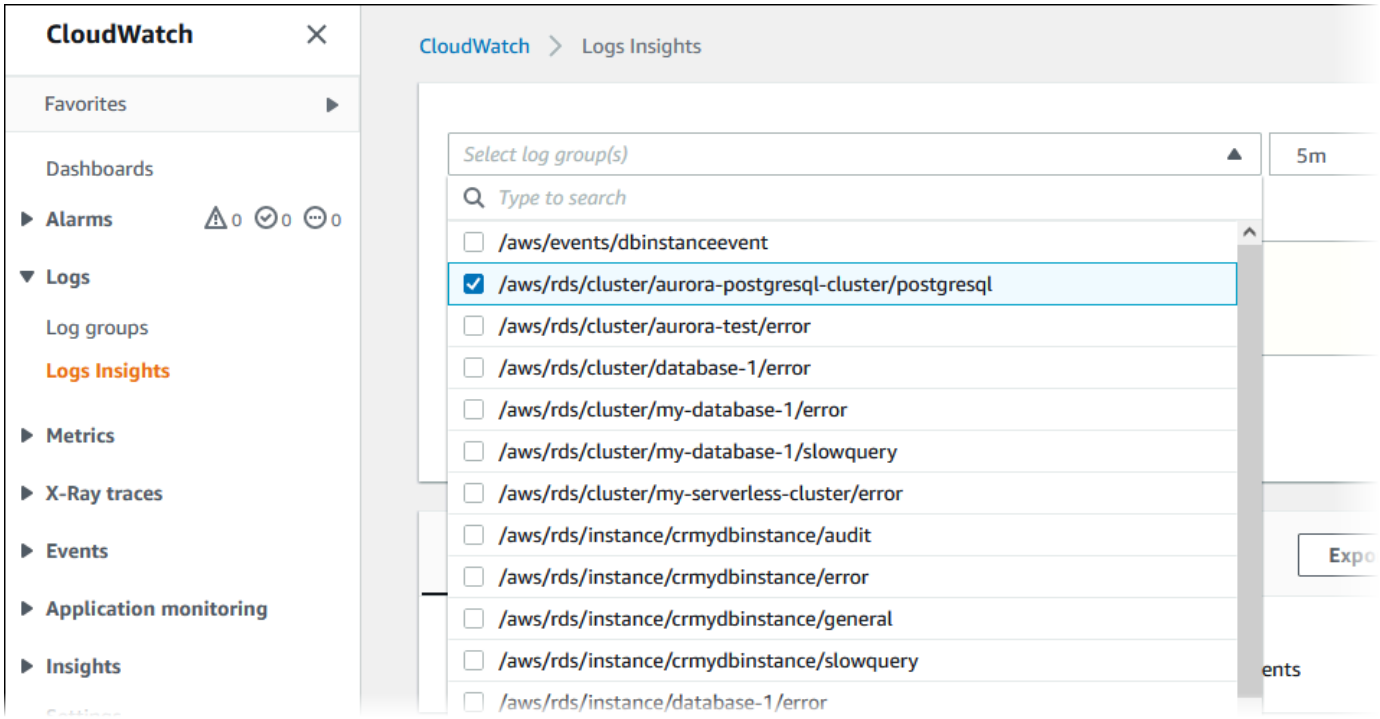
> **Tip**
>
> Anda dapat menggunakan konfigurasi otomatis, seperti AWS CloudFormation, untuk membuat grup log dengan periode retensi log, filter metrik, dan izin akses yang telah ditentukan sebelumnya.

Menganalisis log PostgreSQL menggunakan Wawasan Log CloudWatch

Dengan log PostgreSQL dari klaster DB PostgreSQL Aurora yang CloudWatch diterbitkan sebagai Log, Anda dapat CloudWatch menggunakan Wawasan Log untuk mencari dan menganalisis data log Anda secara interaktif di Log Amazon. CloudWatch CloudWatch Wawasan Log mencakup bahasa kueri, kueri sampel, dan alat lain untuk menganalisis data log sehingga Anda dapat mengidentifikasi potensi masalah dan memverifikasi perbaikan. Untuk mempelajari selengkapnya, lihat Menganalisis data CloudWatch log dengan Wawasan Log di Panduan Pengguna CloudWatch Log Amazon. CloudWatch Log Amazon

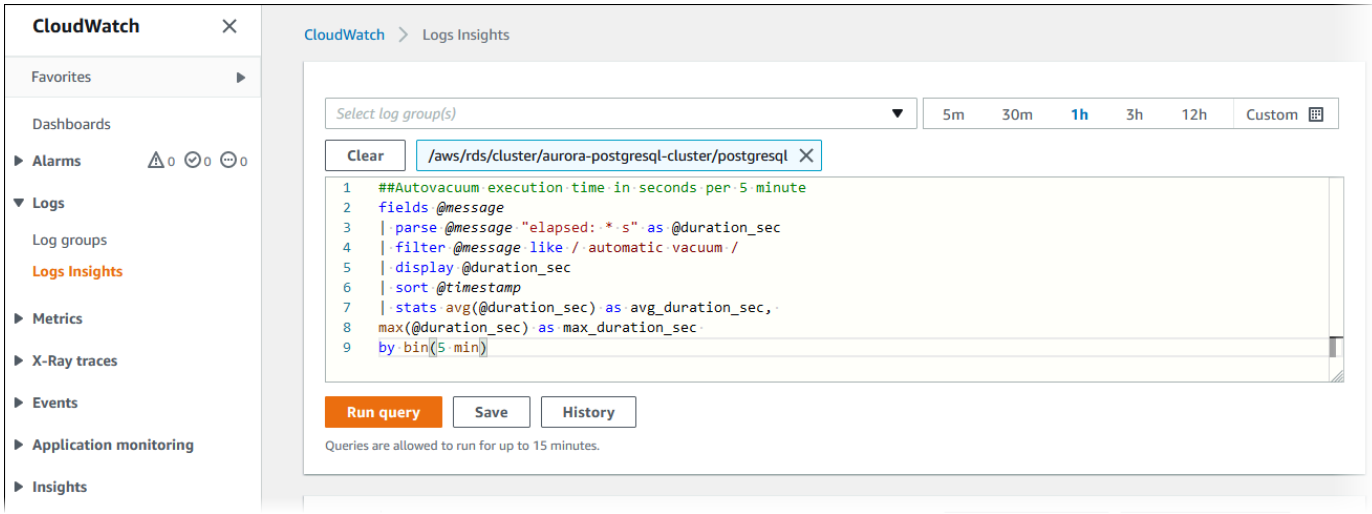
Untuk menganalisis log PostgreSQL dengan Wawasan Log CloudWatch

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Di panel navigasi, buka Log dan pilih Wawasan log.
3. Di opsi Pilih grup log, pilih grup log untuk klaster DB Aurora PostgreSQL Anda.

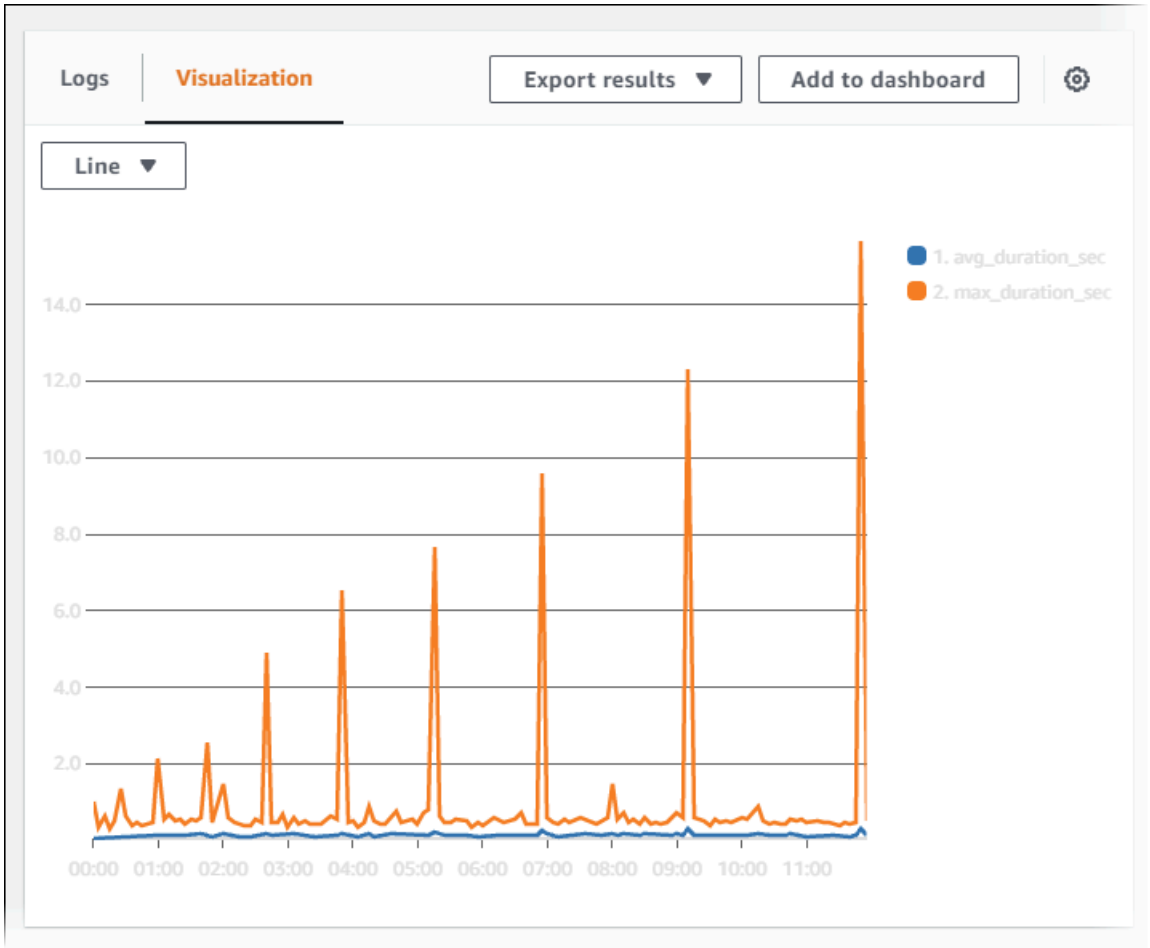


4. Di editor kueri, hapus kueri yang saat ini ditampilkan, masukkan yang berikut ini, lalu pilih Jalankan kueri.

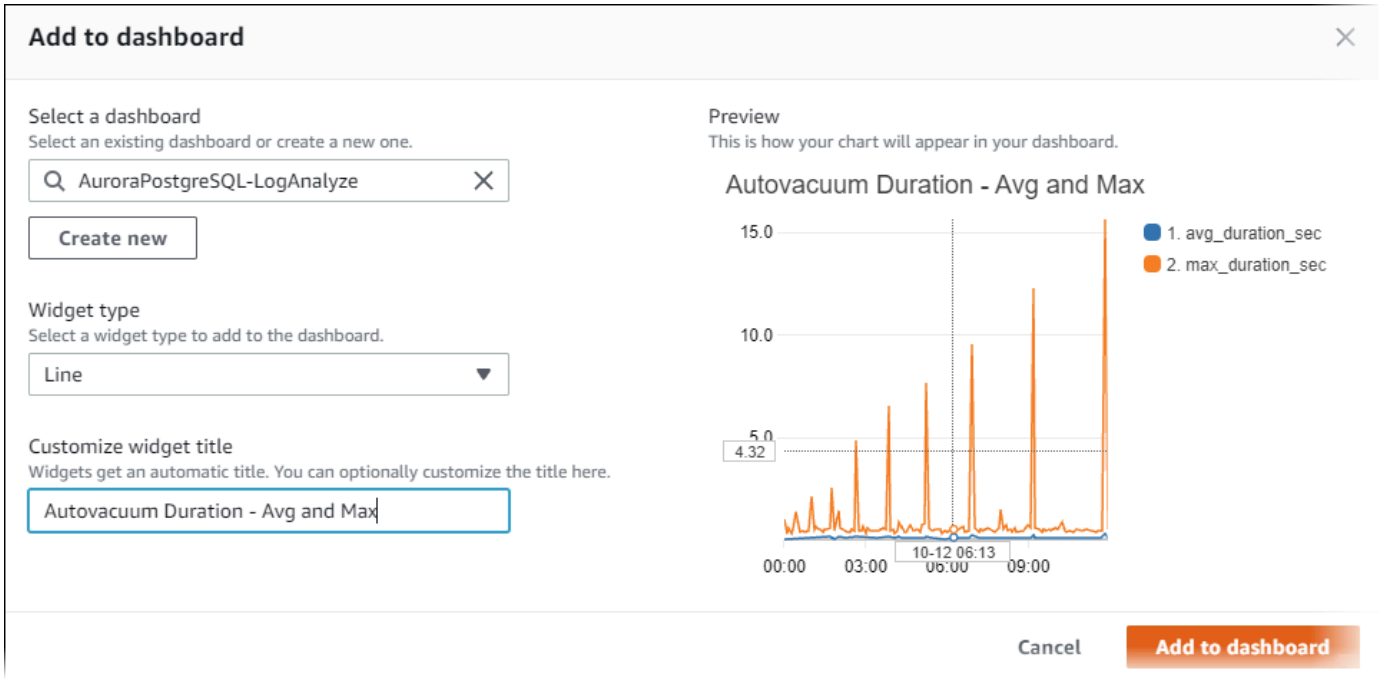
```
##Autovacuum execution time in seconds per 5 minute
fields @message
| parse @message "elapsed: * s" as @duration_sec
| filter @message like / automatic vacuum /
| display @duration_sec
| sort @timestamp
| stats avg(@duration_sec) as avg_duration_sec,
max(@duration_sec) as max_duration_sec
by bin(5 min)
```



5. Pilih tab Visualisasi.



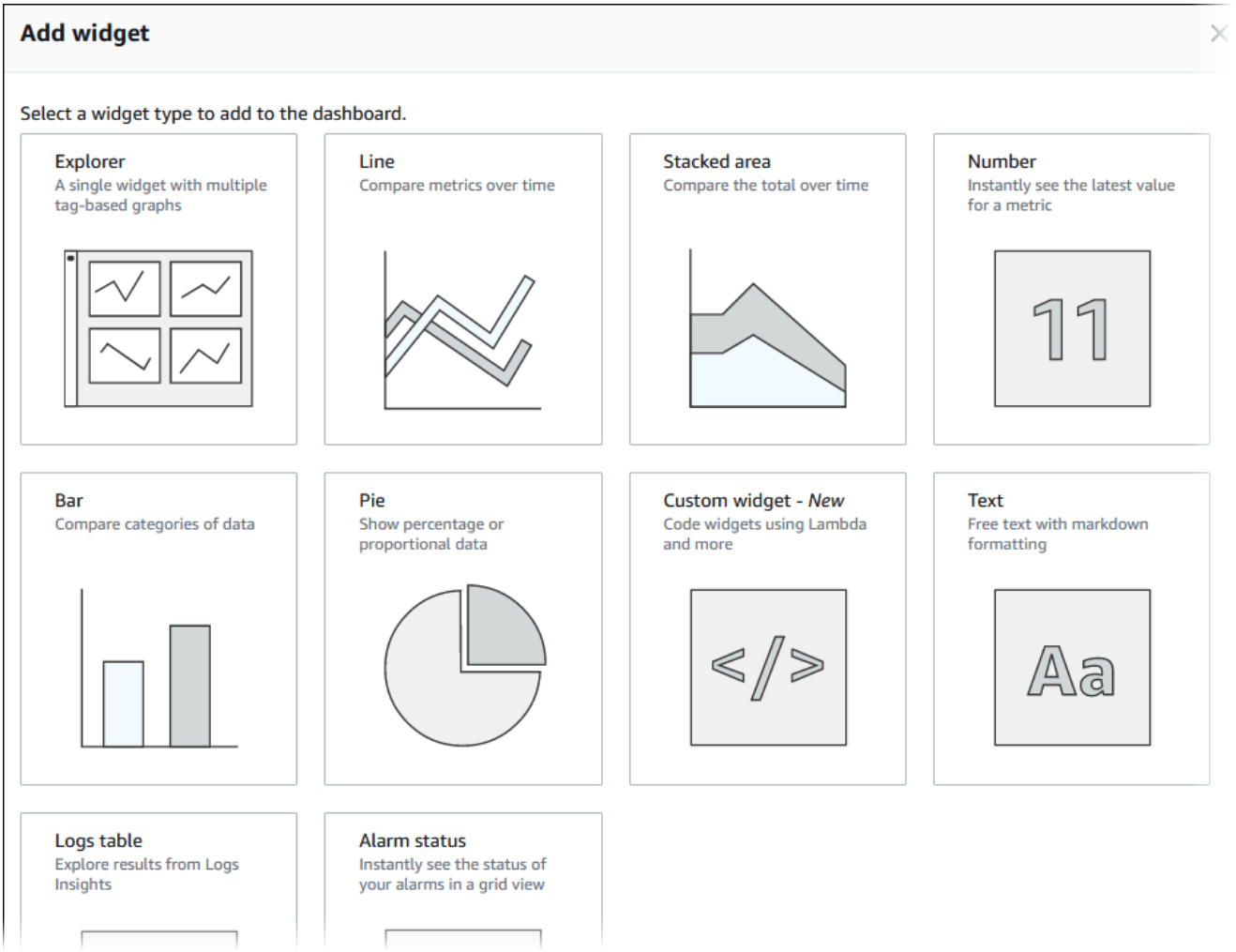
6. Pilih Tambahkan ke dasbor.
7. Di Pilih dasbor, pilih dasbor atau masukkan nama untuk membuat dasbor baru.
8. Dalam Jenis widget, pilih jenis widget untuk visualisasi Anda.



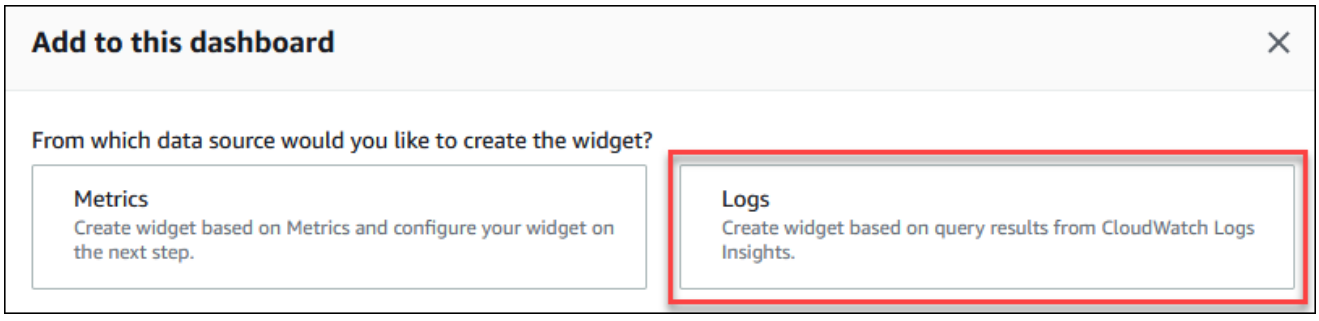
9. (Opsional) Tambahkan lebih banyak widget berdasarkan hasil kueri log Anda.

   a. Pilih Tambahkan widget.

   b. Pilih jenis widget, seperti Baris.



c. Di jendela Tambahkan ke dasbor ini, pilih Log.

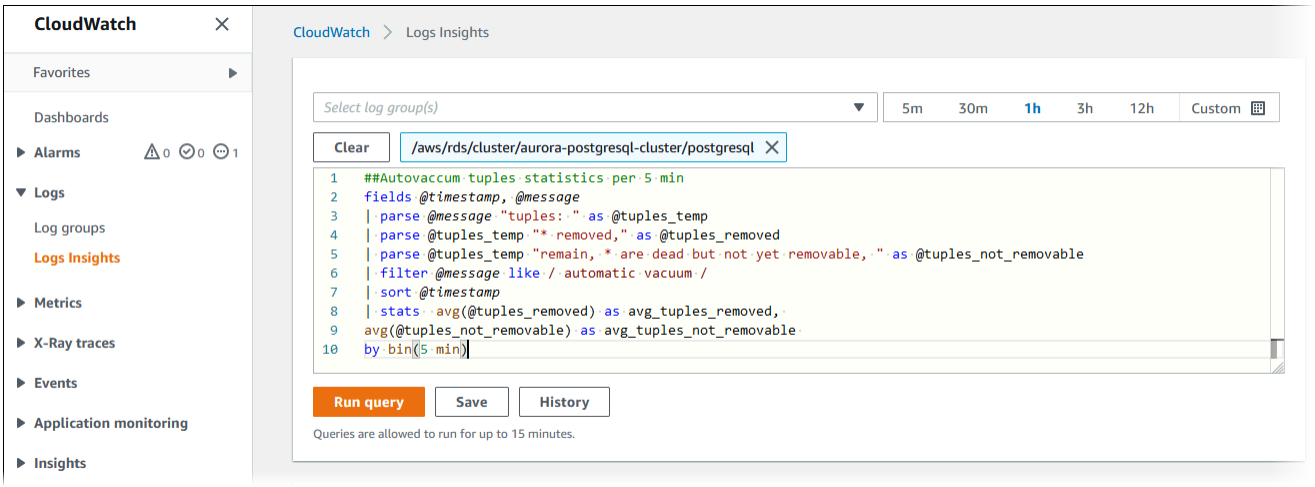


d. Di opsi Pilih grup log, pilih grup log untuk klaster DB Anda.

e. Di editor kueri, hapus kueri yang saat ini ditampilkan, masukkan yang berikut ini, lalu pilih Jalankan kueri.

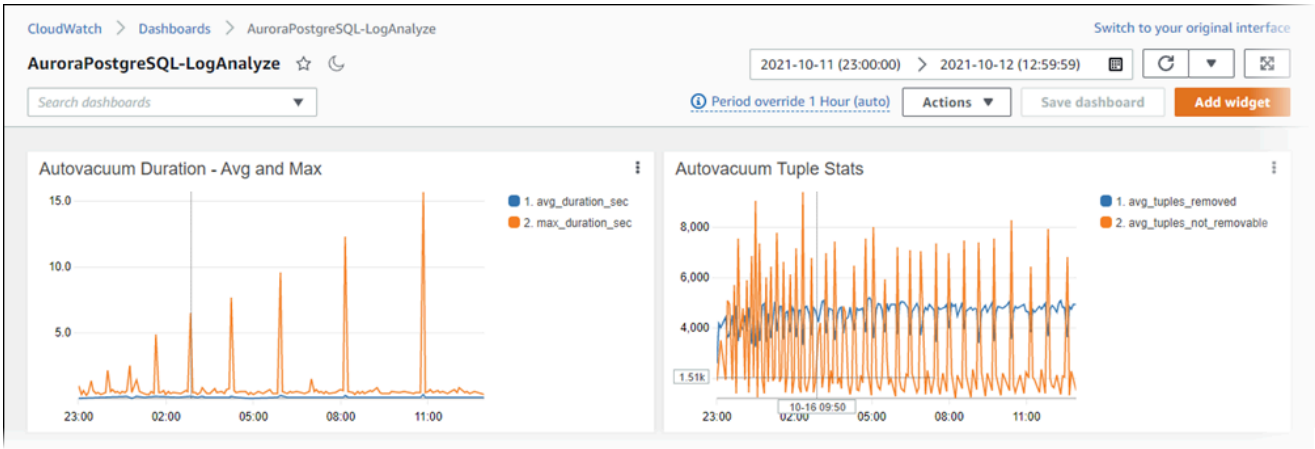
```
##Autovacuum tuples statistics per 5 min
fields @timestamp, @message
| parse @message "tuples: " as @tuples_temp
```

```
| parse @tuples_temp "* removed," as @tuples_removed
| parse @tuples_temp "remain, * are dead but not yet removable, " as
 @tuples_not_removable
| filter @message like / automatic vacuum /
| sort @timestamp
| stats  avg(@tuples_removed) as avg_tuples_removed,
avg(@tuples_not_removable) as avg_tuples_not_removable
by bin(5 min)
```



f. Pilih Buat widget.

Dasbor Anda akan terlihat seperti gambar berikut.



Memantau rencana eksekusi kueri untuk Aurora PostgreSQL

Anda dapat memantau rencana eksekusi kueri di instans Aurora PostgreSQL DB Anda untuk mendeteksi rencana eksekusi yang berkontribusi pada pemuatan basis data saat ini dan untuk melacak statistik kinerja rencana eksekusi dari waktu ke waktu menggunakan parameter. `aurora_compute_plan_id` Setiap kali kueri dijalankan, rencana eksekusi yang digunakan oleh kueri diberi pengenal dan pengidentifikasi yang sama digunakan oleh eksekusi berikutnya dari rencana yang sama.

Dihidupkan `aurora_compute_plan_id` secara default di grup parameter DB dari Aurora PostgreSQL versi 14.10, 15.5, dan versi yang lebih tinggi. Penetapan pengidentifikasi rencana adalah perilaku default dan dapat dimatikan dengan menyetel `aurora_compute_plan_id` ke OFF di grup parameter.

Pengidentifikasi rencana ini digunakan dalam beberapa utilitas yang melayani tujuan yang berbeda.

Topik

- Mengakses rencana eksekusi kueri menggunakan fungsi Aurora
- Referensi parameter untuk rencana eksekusi kueri Aurora PostgreSQL

Mengakses rencana eksekusi kueri menggunakan fungsi Aurora

Dengan`aurora_compute_plan_id`, Anda dapat mengakses rencana eksekusi menggunakan fungsi-fungsi berikut:

- aurora_stat_activity
- aurora_stat_plans

Untuk informasi lebih lanjut tentang fungsi-fungsi ini, lihatReferensi fungsi Aurora PostgreSQL.

Referensi parameter untuk rencana eksekusi kueri Aurora PostgreSQL

Anda dapat memantau rencana eksekusi kueri menggunakan parameter di bawah ini dalam grup parameter DB.

Parameter

- aurora_compute_plan_id

- aurora_stat_plans.minutes_until_recapture
- aurora_stat_plans.calls_until_recapture
- aurora_stat_plans.with_costs
- aurora_stat_plans.with_analysis
- aurora_stat_plans.with_timing
- aurora_stat_plans.with_buffers
- aurora_stat_plans.with_wal
- aurora_stat_plans.with_trigger

> **Note**
>
> Konfigurasi untuk `aurora_stat_plans.with_*` parameter hanya berlaku untuk rencana yang baru ditangkap.

aurora_compute_plan_id

Setel `off` untuk mencegah pengenal rencana ditetapkan.

Default	Nilai yang diizinkan	Deskripsi
on	0 (nonaktif)	Setel `off` untuk mencegah pengenal rencana ditetapkan.
	1 (aktif)	Setel on untuk menetapkan pengenal rencana.

aurora_stat_plans.minutes_until_recapture

Jumlah menit yang harus dilewati sebelum rencana direbut kembali. Defaultnya adalah 0 yang akan menonaktifkan pengambilan kembali rencana. Ketika `aurora_stat_plans.calls_until_recapture` ambang batas dilewati, rencana akan direbut kembali.

Default	Nilai yang diizinkan	Deskripsi
0	0-1073741823	Atur jumlah menit yang harus dilewati sebelum rencana direbut kembali.

aurora_stat_plans.calls_until_recapture

Jumlah panggilan ke rencana sebelum ditangkap kembali. Defaultnya adalah 0 yang akan menonaktifkan pengambilan kembali paket setelah sejumlah panggilan. Ketika `aurora_stat_plans.minutes_until_recapture` ambang batas dilewati, rencana akan direbut kembali.

Default	Nilai yang diizinkan	Deskripsi
0	0-1073741823	Tetapkan jumlah panggilan sebelum rencana ditangkap kembali.

aurora_stat_plans.with_costs

Menangkap rencana EXPLORE dengan perkiraan biaya. Nilai yang diizinkan adalah on dan`off`. Default-nya adalah on.

Default	Nilai yang diizinkan	Deskripsi
on	0 (nonaktif)	Tidak menunjukkan perkiraan biaya dan baris untuk setiap node paket.
	1 (aktif)	Menunjukkan perkiraan biaya dan baris untuk setiap node paket.

aurora_stat_plans.with_analysis

Mengontrol rencana EXPLY dengan ANALYSIS. Mode ini hanya digunakan saat pertama kali rencana ditangkap. Nilai yang diizinkan adalah on dan`off`. Default-nya adalah `off`.

Default	Nilai yang diizinkan	Deskripsi
off	0 (nonaktif)	Tidak termasuk statistik waktu berjalan aktual untuk rencana tersebut.
	1 (aktif)	Termasuk statistik waktu berjalan aktual untuk rencana tersebut.

aurora_stat_plans.with_timing

Waktu rencana akan ditangkap dalam penjelasan saat ANALISIS digunakan. Default-nya adalah on.

Default	Nilai yang diizinkan	Deskripsi
on	0 (nonaktif)	Tidak termasuk waktu start up aktual dan waktu yang dihabiskan di setiap node rencana.
	1 (aktif)	Termasuk waktu start up aktual dan waktu yang dihabiskan di setiap node rencana.

aurora_stat_plans.with_buffers

Statistik penggunaan buffer rencana akan ditangkap dalam penjelasan saat ANALYZE digunakan. Default-nya adalah `off`.

Default	Nilai yang diizinkan	Deskripsi
off	0 (nonaktif)	Tidak termasuk informasi tentang penggunaan buffer.
	1 (aktif)	Termasuk informasi tentang penggunaan buffer.

aurora_stat_plans.with_wal

Statistik penggunaan plan wal akan ditangkap dalam penjelasan saat ANALYZE digunakan. Default-nya adalah `off`.

Default	Nilai yang diizinkan	Deskripsi
off	0 (nonaktif)	Tidak termasuk informasi tentang pembuatan catatan WAL.
	1 (aktif)	Termasuk informasi tentang pembuatan catatan WAL.

aurora_stat_plans.with_trigger

Statistik eksekusi pemicu rencana akan ditangkap dalam penjelasan saat `ANALYZE` digunakan. Default-nya adalah `off`.

Default	Nilai yang diizinkan	Deskripsi
off	0 (nonaktif)	Tidak termasuk statistik eksekusi pemicu.
	1 (aktif)	Termasuk statistik eksekusi pemicu.

Mengelola rencana eksekusi kueri untuk Aurora PostgreSQL

Manajemen rencana kueri Aurora PostgreSQL adalah fitur opsional yang dapat Anda gunakan dengan klaster DB Amazon Aurora PostgreSQL-Compatible Edition. Fitur ini dikemas sebagai ekstensi `apg_plan_mgmt` yang dapat Anda instal di klaster DB Aurora PostgreSQL Anda. Manajemen rencana kueri memungkinkan Anda mengelola rencana eksekusi kueri yang dihasilkan oleh pengoptimisasi untuk aplikasi SQL Anda. Ekstensi AWS `apg_plan_mgmt` dibangun di atas fungsionalitas pemrosesan kueri native dari mesin basis data PostgreSQL.

Di bagian berikut ini, Anda dapat menemukan informasi tentang fitur manajemen rencana kueri Aurora PostgreSQL, cara mengaturnya, dan cara menggunakannya dengan klaster DB Aurora PostgreSQL Anda. Sebelum memulai, sebaiknya Anda meninjau catatan rilis apa pun untuk versi spesifik ekstensi `apg_plan_mgmt` yang tersedia untuk versi Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Versi ekstensi apg_plan_mgmt Aurora PostgreSQL dalam Catatan Rilis untuk Aurora PostgreSQL.

Topik

- Gambaran umum manajemen rencana kueri Aurora PostgreSQL
- Praktik terbaik untuk manajemen rencana kueri Aurora PostgreSQL
- Memahami manajemen rencana kueri Aurora PostgreSQL
- Mengambil rencana eksekusi Aurora PostgreSQL
- Menggunakan rencana terkelola Aurora PostgreSQL
- Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans
- Mengelola rencana eksekusi Aurora PostgreSQL
- Referensi untuk manajemen rencana kueri Aurora PostgreSQL
- Fitur lanjutan dalam Manajemen Rencana Kueri

Gambaran umum manajemen rencana kueri Aurora PostgreSQL

Manajemen rencana kueri Aurora PostgreSQL dirancang untuk memastikan stabilitas rencana terlepas dari perubahan pada basis data yang dapat menyebabkan regresi rencana kueri. Regresi rencana kueri terjadi ketika pengoptimal memilih rencana yang kurang optimal untuk pernyataan SQL yang diberikan setelah perubahan sistem atau basis data. Perubahan pada statistik, pembatasan, pengaturan lingkungan, pengikatan parameter kueri, dan upgrade ke mesin basis data PostgreSQL semuanya dapat menyebabkan regresi rencana.

Dengan manajemen rencana kueri Aurora PostgreSQL, Anda dapat mengendalikan bagaimana dan kapan rencana eksekusi kueri akan berubah. Manfaat manajemen rencana kueri Aurora PostgreSQL meliputi yang berikut ini.

- Meningkatkan stabilitas rencana dengan memaksa pengoptimal untuk memilih dari sejumlah kecil rencana yang sudah diketahui bagus.
- Mengoptimalkan rencana secara terpusat lalu mendistribusikan rencana terbaik secara global.
- Mengidentifikasi indeks yang tidak digunakan dan menilai dampak dari membuat atau menghapus indeks.
- Secara otomatis mendeteksi rencana biaya minimum baru yang ditemukan oleh pengoptimal.
- Mencoba fitur-fitur pengoptimal baru dengan risiko lebih rendah karena Anda dapat memilih untuk hanya menyetujui perubahan rencana yang meningkatkan performa.

Anda dapat menggunakan alat yang disediakan oleh manajemen rencana kueri secara proaktif untuk menentukan rencana terbaik untuk kueri tertentu. Atau, Anda dapat menggunakan manajemen rencana kueri untuk bereaksi terhadap perubahan keadaan dan menghindari regresi rencana. Untuk informasi selengkapnya, lihat Praktik terbaik untuk manajemen rencana kueri Aurora PostgreSQL.

Topik

- Pernyataan SQL yang didukung
- Batasan manajemen rencana kueri
- Terminologi manajemen rencana kueri
- Versi manajemen rencana kueri Aurora PostgreSQL
- Mengaktifkan manajemen rencana kueri Aurora PostgreSQL
- Meng-upgrade manajemen rencana kueri Aurora PostgreSQL
- Menonaktifkan manajemen rencana kueri Aurora PostgreSQL

Pernyataan SQL yang didukung

Manajemen rencana kueri mendukung jenis pernyataan SQL berikut.

- Pernyataan SELECT, INSERT, UPDATE, atau DELETE, terlepas dari kompleksitasnya.
- Pernyataan yang disiapkan. Untuk informasi selengkapnya, lihat PREPARE dalam dokumentasi PostgreSQL.

- Pernyataan dinamis, termasuk yang dijalankan dalam mode segera. Untuk informasi selengkapnya, lihat Dynamic SQL dan EXECUTE IMMEDIATE dalam dokumentasi PostgreSQL.
- Perintah dan pernyataan SQL tersemat. Untuk informasi selengkapnya, lihat Embedded SQL Commands dalam dokumentasi PostgreSQL.
- Pernyataan di dalam fungsi bernama. Untuk informasi selengkapnya, lihat CREATE FUNCTION dalam dokumentasi PostgreSQL.
- Pernyataan yang berisi tabel temp.
- Pernyataan di dalam prosedur dan DO-block.

Anda dapat menggunakan manajemen rencana kueri dengan `EXPLAIN` dalam mode manual untuk menangkap rencana tanpa benar-benar menjalankannya. Untuk informasi selengkapnya, lihat Menganalisis rencana yang dipilih pengoptimisasi. Untuk mempelajari selengkapnya tentang mode manajemen rencana kueri (manual, otomatis), lihat Mengambil rencana eksekusi Aurora PostgreSQL.

Manajemen rencana kueri Aurora PostgreSQL mendukung semua fitur bahasa PostgreSQL, termasuk tabel yang dipartisi, pewarisan, keamanan tingkat baris, dan ekspresi tabel umum (CTE) rekursif. Untuk mempelajari selengkapnya tentang fitur bahasa PostgreSQL ini, lihat Table Partitioning, Row Security Policies, dan WITH Queries (Common Table Expressions) serta topik lainnya dalam dokumentasi PostgreSQL.

Untuk informasi tentang berbagai versi fitur manajemen rencana kueri Aurora PostgreSQL, lihat Aurora PostgreSQL versi ekstensi apg_plan_mgmt dalam Catatan Rilis untuk Aurora PostgreSQL.

Batasan manajemen rencana kueri

Rilis manajemen rencana kueri Aurora PostgreSQL saat ini memiliki batasan berikut.

- Rencana tidak ditangkap untuk pernyataan yang mereferensikan relasi sistem – Pernyataan yang mereferensikan relasi sistem, seperti `pg_class`, tidak ditangkap. Hal ini berlaku berdasarkan desain untuk mencegah sejumlah besar rencana yang dihasilkan sistem yang digunakan secara internal agar tidak ditangkap. Hal ini juga berlaku untuk tabel sistem dalam tampilan.
- Kelas instans DB yang lebih besar mungkin diperlukan untuk klaster DB PostgreSQL Aurora Anda – Bergantung pada beban kerjanya, manajemen rencana kueri mungkin memerlukan kelas instans DB yang memiliki lebih dari 2 vCPU. Jumlah `max_worker_processes` dibatasi oleh ukuran kelas instans DB. Jumlah `max_worker_processes` yang disediakan oleh kelas instans DB 2-vCPU (db.t3.medium, misalnya) mungkin tidak cukup untuk beban kerja tertentu. Kami menyarankan agar

Anda memilih kelas instans DB dengan lebih dari 2 vCPU untuk klaster DB Aurora PostgreSQL Anda jika Anda menggunakan manajemen rencana kueri.

Ketika kelas instans DB tidak dapat mendukung beban kerja, manajemen rencana kueri memunculkan pesan kesalahan seperti berikut ini.

```
WARNING: could not register plan insert background process
HINT: You may need to increase max_worker_processes.
```

Dalam hal ini, Anda harus menaikkan skala klaster DB Aurora PostgreSQL Anda ke ukuran kelas instans DB dengan lebih banyak memori. Untuk informasi selengkapnya, lihat Mesin DB yang didukung untuk kelas instans DB.

- Rencana yang sudah disimpan dalam sesi tidak terpengaruh – Manajemen rencana kueri menyediakan cara untuk memengaruhi rencana kueri tanpa mengubah kode aplikasi. Namun, ketika rencana generik sudah disimpan dalam sesi yang ada dan jika Anda ingin mengubah rencana kueri, maka Anda harus terlebih dahulu mengatur `plan_cache_mode` ke `force_custom_plan` dalam grup parameter klaster DB.
- `queryid` di `apg_plan_mgmt.dba_plans` dan `pg_stat_statements` dapat menyimpang ketika:
  - Objek dihapus dan dibuat ulang setelah disimpan dalam apg_plan_mgmt.dba_plans.
  - Tabel `apg_plan_mgmt.plans` diimpor dari klaster lain.

Untuk informasi tentang berbagai versi fitur manajemen rencana kueri Aurora PostgreSQL, lihat Aurora PostgreSQL versi ekstensi apg_plan_mgmt dalam Catatan Rilis untuk Aurora PostgreSQL.

Terminologi manajemen rencana kueri

Istilah berikut digunakan di seluruh topik ini:

pernyataan terkelola

Pernyataan SQL yang ditangkap oleh pengoptimal dalam manajemen rencana kueri. Pernyataan terkelola memiliki satu atau beberapa rencana eksekusi kueri yang disimpan dalam tampilan `apg_plan_mgmt.dba_plans`.

acuan dasar rencana

Kumpulan rencana yang disetujui untuk pernyataan terkelola tertentu. Artinya, semua rencana untuk pernyataan terkelola yang telah "Disetujui" untuk kolom `status`-nya dalam tampilan `dba_plan`.

riwayat rencana

Kumpulan semua rencana yang ditangkap untuk suatu pernyataan terkelola tertentu. Riwayat rencana berisi semua rencana yang ditangkap untuk pernyataan tersebut, terlepas dari statusnya.

regresi rencana kueri

Kasus ketika pengoptimal memilih rencana yang kurang optimal dibandingkan dengan sebelum perubahan tertentu dilakukan pada lingkungan basis data, seperti versi PostgreSQL baru atau perubahan statistik.

Versi manajemen rencana kueri Aurora PostgreSQL

Manajemen rencana kueri didukung oleh semua rilis Aurora PostgreSQL yang tersedia saat ini. Untuk informasi selengkapnya, lihat Daftar pembaruan Amazon Aurora PostgreSQL dalam Catatan Rilis untuk Aurora PostgreSQL.

Fungsionalitas manajemen rencana kueri ditambahkan ke klaster DB Aurora PostgreSQL Anda saat Anda menginstal ekstensi `apg_plan_mgmt`. Berbagai versi Aurora PostgreSQL mendukung versi ekstensi `apg_plan_mgmt` yang berbeda-beda. Kami menyarankan agar Anda meng-upgrade ekstensi manajemen rencana kueri ke rilis terbaru untuk versi Aurora PostgreSQL Anda.

> **Note**
>
> Untuk catatan rilis untuk setiap versi ekstensi `apg_plan_mgmt`, lihat Aurora PostgreSQL versi ekstensi apg_plan_mgmt dalam Catatan Rilis untuk Aurora PostgreSQL.

Anda dapat mengidentifikasi versi yang berjalan di klaster Anda dengan menghubungkan ke instans menggunakan `psql` lalu menggunakan perintah meta \dx untuk menampilkan daftar ekstensi seperti yang ditunjukkan berikut.

```
labdb=> \dx
```

```
                        List of installed extensions
     Name      | Version |    Schema     |                              Description
---------------+---------+---------------
+-------------------------------------------------------------------
 apg_plan_mgmt | 1.0     | apg_plan_mgmt | Amazon Aurora with PostgreSQL compatibility
 Query Plan Management
 plpgsql       | 1.0     | pg_catalog    | PL/pgSQL procedural language
(2 rows)
```

Output menunjukkan bahwa klaster ini menggunakan ekstensi versi 1.0. Hanya versi `apg_plan_mgmt` tertentu yang tersedia untuk versi Aurora PostgreSQL tertentu. Dalam beberapa kasus, Anda mungkin perlu meng-upgrade klaster DB Aurora PostgreSQL ke rilis minor baru atau menerapkan patch sehingga Anda dapat meng-upgrade ke manajemen rencana kueri versi terbaru. `apg_plan_mgmt` versi 1.0 yang ditampilkan dalam output berasal dari klaster DB Aurora PostgreSQL versi 10.17, yang tidak memiliki `apg_plan_mgmt` versi lebih baru. Dalam hal ini, klaster DB Aurora PostgreSQL harus di-upgrade ke versi PostgreSQL yang lebih baru.

Untuk informasi selengkapnya tentang meng-upgrade klaster DB Aurora PostgreSQL Anda ke PostgreSQL versi baru, lihat Pembaruan Amazon Aurora PostgreSQL.

Untuk mempelajari cara meng-upgrade ekstensi `apg_plan_mgmt`, lihat Meng-upgrade manajemen rencana kueri Aurora PostgreSQL.

Mengaktifkan manajemen rencana kueri Aurora PostgreSQL

Penyiapan manajemen rencana kueri untuk klaster DB Aurora PostgreSQL Anda memerlukan penginstalan ekstensi dan perubahan beberapa pengaturan parameter klaster DB. Anda memerlukan izin `rds_superuser` untuk menginstal ekstensi `apg_plan_mgmt` dan mengaktifkan fitur untuk klaster DB Aurora PostgreSQL.

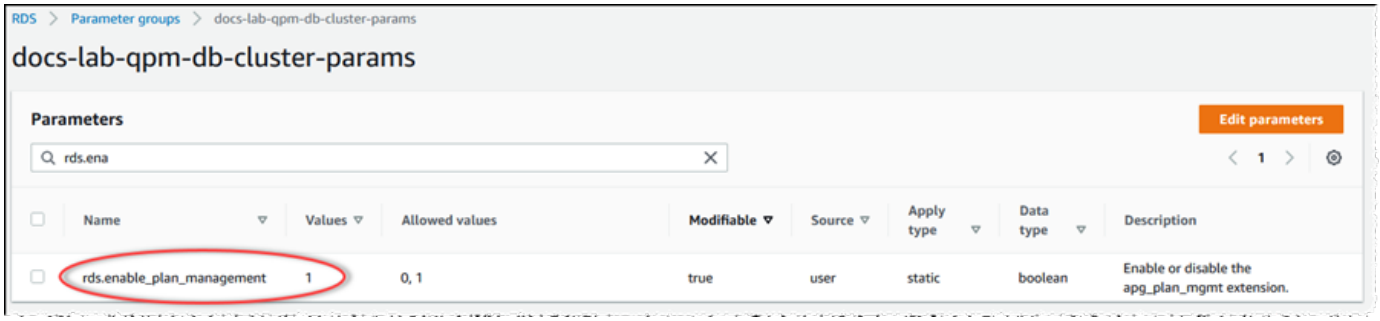
Saat ekstensi ini diinstal, sebuah peran baru, `apg_plan_mgmt`, akan dibuat. Peran ini memungkinkan pengguna basis data melihat, mengelola, dan memelihara rencana kueri. Sebagai administrator dengan hak akses `rds_superuser`, pastikan untuk memberikan peran `apg_plan_mgmt` kepada pengguna basis data sesuai kebutuhan.

Hanya pengguna dengan peran `rds_superuser` yang dapat menyelesaikan prosedur berikut. `rds_superuser` diperlukan untuk membuat ekstensi `apg_plan_mgmt` dan peran `apg_plan_mgmt`. Pengguna harus diberi peran `apg_plan_mgmt` untuk mengelola ekstensi `apg_plan_mgmt`.

Mengaktifkan manajemen rencana kueri untuk klaster DB Aurora PostgreSQL Anda

Langkah-langkah berikut mengaktifkan manajemen rencana kueri untuk semua pernyataan SQL yang dikirimkan ke klaster DB Aurora PostgreSQL. Hal ini dikenal sebagai mode otomatis. Untuk mempelajari selengkapnya tentang perbedaan di antara mode, lihat Mengambil rencana eksekusi Aurora PostgreSQL.

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Buat grup parameter klaster DB kustom untuk klaster DB Aurora PostgreSQL Anda. Anda perlu mengubah parameter tertentu untuk mengaktifkan manajemen rencana kueri dan mengatur perilakunya. Untuk informasi selengkapnya, lihat Membuat grup parameter DB.
3. Buka grup parameter klaster DB kustom dan atur parameter `rds.enable_plan_management` ke 1, seperti yang ditunjukkan pada gambar berikut ini.



   Untuk informasi selengkapnya, lihat Mengubah parameter dalam grup parameter klaster DB.
4. Buat grup parameter DB kustom yang dapat Anda gunakan untuk mengatur parameter rencana kueri di tingkat instans. Untuk informasi selengkapnya, lihat Membuat grup parameter klaster DB.
5. Ubah instans penulis klaster DB Aurora PostgreSQL untuk menggunakan grup parameter DB kustom. Untuk informasi selengkapnya, lihat Memodifikasi instans DB dalam klaster DB.
6. Ubah klaster DB Aurora PostgreSQL untuk menggunakan grup parameter klaster DB kustom. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API.
7. Boot ulang instans DB Anda untuk mengaktifkan pengaturan grup parameter kustom.
8. Hubungkan ke titik akhir instans DB untuk klaster DB Aurora PostgreSQL Anda menggunakan `psql` atau `pgAdmin`. Contoh berikut menggunakan akun `postgres` default untuk peran `rds_superuser`.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=my-db
```

9. Buat ekstensi `apg_plan_mgmt` untuk instans DB Anda, seperti yang ditunjukkan berikut ini.

```
labdb=> CREATE EXTENSION apg_plan_mgmt;
CREATE EXTENSION
```

> **Tip**
>
> Instal ekstensi `apg_plan_mgmt` dalam basis data templat untuk aplikasi Anda. Basis data templat default diberi nama `template1`. Untuk mempelajari selengkapnya, lihat Template Databases dalam dokumentasi PostgreSQL.

10. Ubah parameter `apg_plan_mgmt.capture_plan_baselines` menjadi `automatic`. Pengaturan ini akan menyebabkan pengoptimal menghasilkan rencana untuk setiap pernyataan SQL yang direncanakan atau dijalankan dua kali atau lebih.

> **Note**
>
> Manajemen rencana kueri juga memiliki mode manual yang dapat Anda gunakan untuk pernyataan SQL tertentu. Untuk mempelajari selengkapnya, lihat Mengambil rencana eksekusi Aurora PostgreSQL.

11. Ubah nilai parameter `apg_plan_mgmt.use_plan_baselines` menjadi "on". Parameter ini menyebabkan pengoptimal memilih rencana untuk pernyataan dari acuan dasar rencananya. Untuk mempelajari selengkapnya, lihat Menggunakan rencana terkelola Aurora PostgreSQL.

> **Note**
>
> Anda dapat mengubah nilai salah satu parameter dinamis ini untuk sesi tanpa perlu mem-boot ulang instans.

Ketika pengaturan manajemen rencana kueri Anda selesai, pastikan untuk memberikan peran `apg_plan_mgmt` kepada setiap pengguna basis data yang perlu melihat, mengelola, atau memelihara rencana kueri.

Meng-upgrade manajemen rencana kueri Aurora PostgreSQL

Kami menyarankan agar Anda meng-upgrade ekstensi manajemen rencana kueri ke rilis terbaru untuk versi Aurora PostgreSQL Anda.

1. Hubungkan ke instans penulis klaster DB Aurora PostgreSQL Anda sebagai pengguna yang memiliki hak akses `rds_superuser`. Jika Anda menyimpan nama default saat menyiapkan instans, Anda akan terhubung sebagai `postgres`. Contoh ini menunjukkan cara menggunakan `psql`, tetapi Anda juga dapat menggunakan pgAdmin jika mau.

   ```
   psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password
   ```

2. Jalankan kueri berikut untuk meng-upgrade ekstensi.

   ```
   ALTER EXTENSION apg_plan_mgmt UPDATE TO '2.1';
   ```

3. Gunakan fungsi apg_plan_mgmt.validate_plans untuk memperbarui hash semua rencana. Pengoptimal akan memvalidasi semua rencana yang Disetujui, Tidak Disetujui, dan Ditolak untuk memastikan bahwa rencana tersebut masih layak untuk ekstensi versi baru.

   ```
   SELECT apg_plan_mgmt.validate_plans('update_plan_hash');
   ```

   Untuk mempelajari selengkapnya tentang penggunaan fungsi ini, lihat Memvalidasi rencana.

4. Gunakan fungsi apg_plan_mgmt.reload untuk me-refresh rencana apa pun di memori bersama dengan rencana yang divalidasi dari tampilan dba_plans.

   ```
   SELECT apg_plan_mgmt.reload();
   ```

Untuk mempelajari selengkapnya tentang semua fungsi yang tersedia untuk manajemen rencana kueri, lihat Referensi fungsi untuk manajemen rencana kueri Aurora PostgreSQL.

Menonaktifkan manajemen rencana kueri Aurora PostgreSQL

Anda dapat menonaktifkan manajemen rencana kueri setiap saat dengan menonaktifkan `apg_plan_mgmt.use_plan_baselines` dan `apg_plan_mgmt.capture_plan_baselines`.

```
labdb=> SET apg_plan_mgmt.use_plan_baselines = off;
```

```
labdb=> SET apg_plan_mgmt.capture_plan_baselines = off;
```

Praktik terbaik untuk manajemen rencana kueri Aurora PostgreSQL

Manajemen rencana kueri memungkinkan Anda mengendalikan cara dan waktu perubahan rencana eksekusi kueri. Sebagai DBA, tujuan utama Anda saat menggunakan QPM termasuk mencegah regresi ketika ada perubahan pada basis data Anda, dan mengontrol apakah akan mengizinkan pengoptimisasi menggunakan rencana baru. Di bagian berikut ini, Anda dapat menemukan beberapa praktik terbaik yang direkomendasikan untuk menggunakan manajemen rencana kueri. Pendekatan manajemen rencana yang proaktif dan reaktif berbeda dari segi cara dan waktu persetujuan terhadap penggunaan rencana baru.

Daftar Isi

- Manajemen rencana proaktif untuk membantu mencegah regresi performa
  - Memastikan stabilitas rencana setelah upgrade versi mayor
- Manajemen rencana reaktif untuk mendeteksi dan memperbaiki regresi performa

Manajemen rencana proaktif untuk membantu mencegah regresi performa

Guna mencegah regresi performa rencana terjadi, Anda perlu mengembangkan acuan dasar rencana dengan menjalankan prosedur yang membandingkan performa rencana yang baru ditemukan dengan performa acuan dasar yang sudah ada untuk rencana yang Disetujui, lalu secara otomatis menyetujui rangkaian rencana tercepat sebagai acuan dasar baru. Dengan cara ini, acuan dasar rencana meningkat dari waktu ke waktu seiring rencana yang lebih cepat ditemukan.

1. Di lingkungan pengembangan, temukan pernyataan SQL yang memiliki dampak terbesar pada performa atau throughput sistem. Kemudian, ambil rencana untuk pernyataan tersebut seperti yang dijelaskan dalam Mengambil rencana secara manual untuk pernyataan SQL tertentu dan Mengambil rencana secara otomatis.
2. Ekspor rencana yang diambil dari lingkungan pengembangan lalu impor ke lingkungan produksi. Untuk informasi selengkapnya, lihat Mengekspor dan mengimpor rencana.
3. Dalam produksi, jalankan aplikasi Anda dan terapkan penggunaan rencana terkelola yang disetujui. Untuk informasi selengkapnya, lihat Menggunakan rencana terkelola Aurora PostgreSQL. Saat aplikasi berjalan, tambahkan juga rencana baru saat pengoptimisasi menemukan rencana tersebut. Untuk informasi selengkapnya, lihat Mengambil rencana secara otomatis.

4. Analisis rencana yang belum disetujui dan setujui rencana yang beperforma baik. Untuk informasi selengkapnya, lihat Mengevaluasi performa rencana.
5. Saat aplikasi Anda terus berjalan, pengoptimisasi mulai menggunakan rencana baru sesuai kebutuhan.

Memastikan stabilitas rencana setelah upgrade versi mayor

Setiap versi mayor PostgreSQL menyertakan penyempurnaan dan perubahan pada pengoptimisasi kueri yang dirancang untuk menaikkan performa. Namun, rencana eksekusi kueri yang dihasilkan oleh pengoptimisasi di versi sebelumnya dapat menyebabkan regresi performa di versi yang di-upgrade yang lebih baru. Anda dapat menggunakan manajemen rencana kueri untuk mengatasi masalah performa ini dan untuk memastikan stabilitas rencana setelah upgrade versi mayor.

Pengoptimisasi selalu menggunakan rencana yang Disetujui dengan biaya minimum, meskipun ada lebih dari satu rencana yang Disetujui untuk pernyataan yang sama. Setelah upgrade, pengoptimisasi mungkin menemukan rencana baru, tetapi rencana tersebut akan disimpan sebagai rencana yang Tidak Disetujui. Rencana ini dilakukan hanya jika disetujui menggunakan gaya reaktif manajemen rencana dengan parameter unapproved_plan_execution_threshold. Anda dapat memaksimalkan stabilitas rencana menggunakan gaya proaktif manajemen rencana dengan parameter evolve_plan_baselines. Parameter ini membandingkan performa rencana baru dengan rencana lama dan menyetujui atau menolak rencana yang setidaknya 10% lebih cepat dari rencana terbaik berikutnya.

Setelah upgrade, Anda dapat menggunakan fungsi `evolve_plan_baselines` untuk membandingkan performa rencana sebelum dan sesudah upgrade menggunakan pengikatan parameter kueri Anda. Langkah-langkah berikut mengasumsikan bahwa Anda telah menggunakan rencana terkelola yang disetujui di lingkungan produksi Anda, seperti yang dijelaskan dalam Menggunakan rencana terkelola Aurora PostgreSQL.

1. Sebelum upgrade, jalankan aplikasi Anda bersama pengelola rencana kueri yang berjalan. Saat aplikasi berjalan, tambahkan juga rencana baru saat pengoptimisasi menemukan rencana tersebut. Untuk informasi selengkapnya, lihat Mengambil rencana secara otomatis.
2. Evaluasi performa setiap rencana. Untuk informasi selengkapnya, lihat Mengevaluasi performa rencana.
3. Setelah upgrade, analisis kembali rencana Anda yang disetujui menggunakan fungsi `evolve_plan_baselines`. Bandingkan performa sebelum dan sesudah menggunakan pengikatan parameter kueri Anda. Jika rencana baru cepat, Anda dapat menambahkannya ke

rencana yang disetujui. Jika lebih cepat dari rencana lain untuk pengikatan parameter yang sama, maka Anda dapat menandai rencana yang lebih lambat sebagai Ditolak.

Untuk informasi selengkapnya, lihat Menyetujui rencana yang lebih baik. Untuk informasi referensi tentang fungsi ini, lihat apg_plan_mgmt.evolve_plan_baselines.

Untuk informasi selengkapnya, lihat artikel Memastikan performa yang konsisten setelah upgrade versi mayor dengan Manajemen Rencana Kueri Amazon Aurora PostgreSQL-Compatible Edition.

> **Note**
>
> Saat Anda melakukan upgrade versi mayor menggunakan replikasi logis atau AWS DMS, pastikan Anda mereplikasi skema `apg_plan_mgmt` untuk memastikan rencana yang sudah ada disalin ke instans yang di-upgrade. Untuk informasi selengkapnya tentang replikasi logis, lihat Menggunakan replikasi logis untuk melakukan upgrade versi mayor untuk Aurora PostgreSQL.

Manajemen rencana reaktif untuk mendeteksi dan memperbaiki regresi performa

Dengan memantau aplikasi Anda saat berjalan, Anda dapat mendeteksi rencana yang menyebabkan regresi performa. Saat Anda mendeteksi regresi, Anda dapat secara manual menolak atau memperbaiki rencana yang buruk dengan mengikuti langkah-langkah berikut:

1. Saat aplikasi Anda berjalan, terapkan penggunaan rencana terkelola dan tambahkan secara otomatis rencana yang baru ditemukan sebagai rencana yang tidak disetujui. Untuk informasi selengkapnya, lihat Menggunakan rencana terkelola Aurora PostgreSQL dan Mengambil rencana secara otomatis.
2. Pantau regresi performa pada aplikasi Anda yang sedang berjalan.
3. Ketika Anda menemukan regresi rencana, atur status rencana menjadi `rejected`. Saat pengoptimisasi kembali menjalankan pernyataan SQL, pengoptimisasi akan secara otomatis mengabaikan rencana yang ditolak dan menggunakan rencana berbeda yang telah disetujui. Untuk informasi selengkapnya, lihat Menolak atau menonaktifkan rencana yang lebih lambat.

   Dalam beberapa kasus, Anda mungkin lebih memilih untuk memperbaiki rencana yang buruk daripada menolak, menonaktifkan, atau menghapusnya. Gunakan ekstensi `pg_hint_plan` untuk bereksperimen dalam meningkatkan rencana. Dengan `pg_hint_plan`, Anda menggunakan

komentar khusus untuk meminta pengoptimisasi mengganti caranya dalam membuat rencana. Untuk informasi selengkapnya, lihat Memperbaiki rencana menggunakan pg_hint_plan.

Memahami manajemen rencana kueri Aurora PostgreSQL

Dengan manajemen rencana kueri diaktifkan untuk klaster DB Aurora PostgreSQL Anda, pengoptimisasi akan menghasilkan dan menyimpan rencana eksekusi kueri untuk pernyataan SQL apa pun yang diproses lebih dari sekali. Pengoptimisasi selalu mengatur status rencana yang pertama kali dibuat dari suatu pernyataan terkelola ke `Approved`, dan menyimpannya di tampilan `dba_plans`.

Rangkaian rencana yang disetujui yang disimpan untuk suatu pernyataan terkelola disebut sebagai acuan dasar rencana. Saat aplikasi berjalan, pengoptimisasi dapat membuat rencana tambahan untuk pernyataan terkelola. Pengoptimisasi mengatur rencana tambahan yang diambil ke status `Unapproved`.

Kemudian, Anda dapat memutuskan apakah rencana `Unapproved` berjalan dengan baik dan mengubahnya menjadi `Approved`, `Rejected`, atau `Preferred`. Untuk melakukannya, Anda menggunakan fungsi `apg_plan_mgmt.evolve_plan_baselines` atau fungsi `apg_plan_mgmt.set_plan_status`.

Ketika pengoptimisasi menghasilkan rencana untuk pernyataan SQL, manajemen rencana kueri menyimpan rencana ini dalam tabel `apg_plan_mgmt.plans`. Pengguna basis data yang telah diberi peran `apg_plan_mgmt` dapat melihat detail rencana dengan mengkueri tampilan `apg_plan_mgmt.dba_plans`. Misalnya, kueri berikut menampilkan daftar detail untuk rencana yang saat ini ada dalam tampilan untuk klaster DB PostgreSQL Aurora non-produksi.

- `sql_hash` – Pengidentifikasi untuk pernyataan SQL yang merupakan nilai hash untuk teks yang dinormalisasi dari pernyataan SQL.
- `plan_hash` – Pengidentifikasi unik untuk rencana yang merupakan kombinasi dari `sql_hash` dan hash rencana.
- `status` – Status rencana. Pengoptimisasi dapat menjalankan rencana yang disetujui.
- `enabled` – Menunjukkan apakah rencana siap digunakan (true) atau tidak (false).
- `plan_outline` – Representasi rencana yang digunakan untuk membuat ulang rencana eksekusi sebenarnya. Operator dalam struktur pohon akan dipetakan ke operator dalam output EXPLAIN.

Tampilan `apg_plan_mgmt.dba_plans` memiliki lebih banyak kolom yang berisi semua detail rencana, misalnya, waktu rencana terakhir digunakan. Untuk detail lengkap, lihat Referensi untuk tampilan apg_plan_mgmt.dba_plans.

Normalisasi dan hash SQL

Dalam tampilan `apg_plan_mgmt.dba_plans`, Anda dapat mengidentifikasi pernyataan terkelola dengan nilai hash SQL. Hash SQL dihitung pada representasi ternormalisasi dari pernyataan SQL yang menghilangkan beberapa perbedaan, seperti nilai-nilai literal.

Proses normalisasi untuk setiap pernyataan SQL mempertahankan spasi dan huruf besar/kecil, sehingga Anda masih dapat membaca dan memahami inti dari pernyataan SQL tersebut. Normalisasi akan menghapus atau mengganti item berikut.

- Komentar blok di depan
- Kata kunci EXPLAIN dan opsi EXPLAIN, serta EXPLAIN ANALYZE
- Spasi di belakang
- Semua literal

Sebagai contoh, amati pernyataan berikut.

```
/*Leading comment*/ EXPLAIN SELECT /* Query 1 */ * FROM t WHERE x > 7 AND y = 1;
```

Pengoptimisasi menormalkan pernyataan ini seperti yang ditunjukkan berikut.

```
SELECT /* Query 1 */ * FROM t WHERE x > CONST AND y = CONST;
```

Normalisasi memungkinkan hash SQL yang sama digunakan untuk pernyataan SQL serupa yang mungkin hanya berbeda dalam hal nilai literal atau parameternya. Dengan kata lain, bisa ada lebih dari satu rencana untuk hash SQL yang sama, dengan sebuah rencana berbeda yang optimal dalam kondisi yang berbeda-beda.

> **Note**
>
> Pernyataan SQL tunggal yang digunakan dengan berbagai skema akan memiliki rencana yang berbeda-beda karena terikat pada skema tertentu saat runtime. Perencana menggunakan statistik untuk pengikatan skema untuk memilih rencana yang optimal.

Untuk mempelajari selengkapnya tentang cara pengoptimisasi memilih rencana, lihat Menggunakan rencana terkelola Aurora PostgreSQL. Di bagian tersebut, Anda dapat mempelajari cara menggunakan `EXPLAIN` dan `EXPLAIN ANALYZE` untuk melihat pratinjau rencana sebelum benar-benar digunakan. Untuk detailnya, lihat Menganalisis rencana yang dipilih pengoptimisasi. Untuk gambar yang menguraikan proses untuk memilih rencana, lihat Bagaimana cara pengoptimisasi memilih rencana yang akan dijalankan.

Mengambil rencana eksekusi Aurora PostgreSQL

Manajemen rencana kueri Aurora PostgreSQL menawarkan dua mode berbeda untuk mengambil rencana eksekusi kueri, yaitu otomatis atau manual. Anda memilih mode dengan mengatur nilai `apg_plan_mgmt.capture_plans_baselines` ke `automatic` atau ke `manual`. Anda dapat mengambil rencana eksekusi untuk pernyataan SQL tertentu menggunakan pengambilan rencana manual. Alternatifnya, Anda dapat mengambil semua rencana (atau rencana yang paling lambat) yang dijalankan dua kali atau lebih saat aplikasi Anda berjalan menggunakan pengambilan rencana otomatis.

Saat mengambil rencana, pengoptimisasi selalu mengatur status rencana yang pertama kali diambil untuk suatu pernyataan terkelola menjadi `approved`. Pengoptimisasi selalu mengatur status setiap rencana tambahan yang diambil untuk suatu pernyataan terkelola menjadi `unapproved`. Namun, kadang-kadang lebih dari satu rencana dapat disimpan dengan status `approved`. Hal ini dapat terjadi ketika beberapa rencana dibuat untuk sebuah pernyataan secara paralel dan sebelum rencana pertama untuk pernyataan tersebut ditetapkan.

Untuk mengontrol jumlah maksimum rencana yang dapat diambil dan disimpan di tampilan `dba_plans`, atur parameter `apg_plan_mgmt.max_plans` dalam grup parameter tingkat instans DB Anda. Perubahan pada paramater `apg_plan_mgmt.max_plans` mengharuskan instans DB di-boot ulang agar nilai baru dapat diterapkan. Untuk informasi selengkapnya, lihat parameter apg_plan_mgmt.max_plans.

Mengambil rencana secara manual untuk pernyataan SQL tertentu

Jika Anda memiliki kumpulan pernyataan SQL yang diketahui untuk dikelola, masukkan pernyataan tersebut ke dalam file skrip SQL lalu ambil rencana secara manual. Hal berikut ini menunjukkan contoh psql tentang cara mengambil rencana kueri secara manual untuk kumpulan pernyataan SQL.

```
psql> SET apg_plan_mgmt.capture_plan_baselines = manual;
psql> \i my-statements.sql
```

```
psql> SET apg_plan_mgmt.capture_plan_baselines = off;
```

Setelah mengambil rencana untuk setiap pernyataan SQL, pengoptimisasi menambahkan baris baru ke tampilan `apg_plan_mgmt.dba_plans`.

Kami menyarankan Anda menggunakan pernyataan EXPLAIN atau EXPLAIN EXECUTE di file skrip SQL. Pastikan Anda memasukkan cukup variasi dalam nilai parameter untuk mengambil semua rencana yang diinginkan.

Jika Anda mengetahui rencana yang lebih baik daripada rencana berbiaya minimum dari pengoptimisasi, Anda mungkin dapat memaksa pengoptimisasi untuk menggunakan rencana yang lebih baik. Untuk melakukannya, tentukan satu atau beberapa petunjuk pengoptimal. Untuk informasi selengkapnya, lihat Memperbaiki rencana menggunakan pg_hint_plan. Untuk membandingkan performa rencana `unapproved` dan `approved` serta menyetujui, menolak, atau menghapusnya, lihat Mengevaluasi performa rencana.

Mengambil rencana secara otomatis

Gunakan pengambilan rencana otomatis untuk situasi seperti berikut ini:

- Anda tidak mengetahui pernyataan SQL spesifik yang ingin Anda kelola.
- Anda memiliki ratusan atau ribuan pernyataan SQL untuk dikelola.
- Aplikasi Anda menggunakan API klien. Misalnya, JDBC menggunakan pernyataan yang disiapkan tanpa nama atau pernyataan mode massal yang tidak dapat dinyatakan dalam psql.

Untuk mengambil rencana secara otomatis

1. Aktifkan pengambilan rencana otomatis dengan mengatur `apg_plan_mgmt.capture_plan_baselines` ke `automatic` dalam grup parameter tingkat instans DB. Untuk informasi selengkapnya, lihat Memodifikasi parameter dalam grup parameter DB.
2. Boot ulang instans DB Anda.
3. Saat aplikasi berjalan, pengoptimisasi mengambil rencana untuk setiap pernyataan SQL yang berjalan setidaknya dua kali.

   Saat aplikasi berjalan dengan pengaturan parameter manajemen rencana kueri default, pengoptimisasi mengambil rencana untuk setiap pernyataan SQL yang berjalan setidaknya dua

kali. Dengan mengambil semua rencana saat menggunakan pengaturan default, overhead run-time yang diperlukan akan sangat kecil dan hal ini dapat dilakukan dalam produksi.

Untuk menonaktifkan pengambilan rencana otomatis

- Tetapkan parameter `apg_plan_mgmt.capture_plan_baselines` ke `off` dari grup parameter tingkat instans DB.

Untuk mengukur performa rencana yang belum disetujui dan menyetujui, menolak, atau menghapusnya, lihat Mengevaluasi performa rencana.

Menggunakan rencana terkelola Aurora PostgreSQL

Agar pengoptimisasi menggunakan rencana yang diambil untuk pernyataan terkelola Anda, atur parameter `apg_plan_mgmt.use_plan_baselines` ke `true`. Berikut ini adalah contoh instans lokal.

```
SET apg_plan_mgmt.use_plan_baselines = true;
```

Sementara aplikasi berjalan, pengaturan ini menyebabkan pengoptimisasi menggunakan rencana berbiaya minimum, pilihan, atau disetujui yang valid dan diaktifkan untuk setiap pernyataan terkelola.

Menganalisis rencana yang dipilih pengoptimisasi

Jika parameter `apg_plan_mgmt.use_plan_baselines` diatur ke `true`, Anda dapat menggunakan pernyataan SQL EXPLAIN ANALYZE untuk membuat pengoptimisasi menampilkan rencana yang akan digunakan jika pengoptimisasi akan menjalankan pernyataan tersebut. Berikut adalah contohnya.

```
EXPLAIN ANALYZE EXECUTE rangeQuery (1,10000);
```

```
                                                      QUERY PLAN
------------------------------------------------------------------------------
 Aggregate  (cost=393.29..393.30 rows=1 width=8) (actual time=7.251..7.251 rows=1
 loops=1)
   ->  Index Only Scan using t1_pkey on t1 t  (cost=0.29..368.29 rows=10000 width=0)
 (actual time=0.061..4.859 rows=10000 loops=1)
Index Cond: ((id >= 1) AND (id <= 10000))
         Heap Fetches: 10000
 Planning time: 1.408 ms
 Execution time: 7.291 ms
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1984047223, Plan Hash: 512153379
```

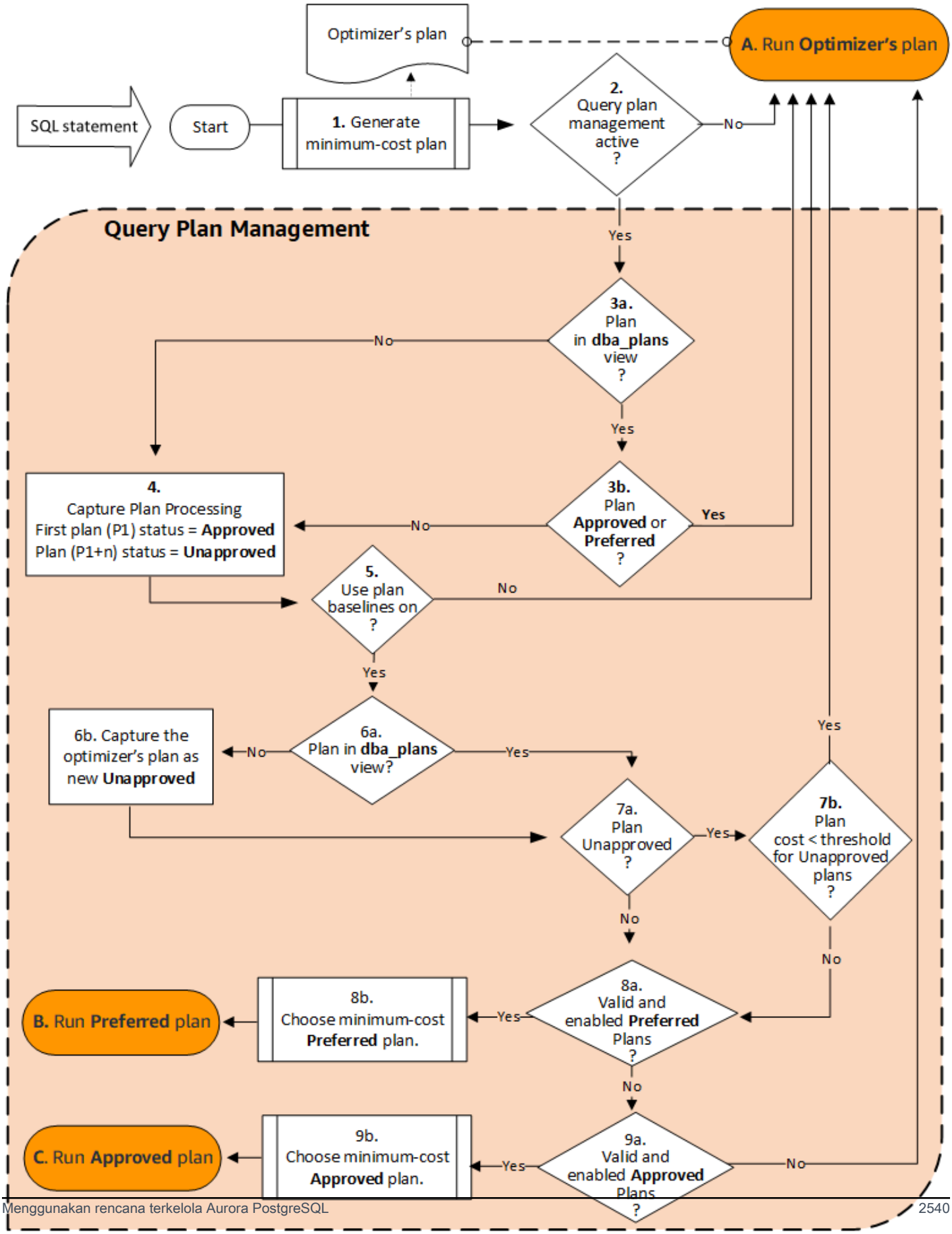
Output ini menunjukkan rencana yang Disetujui dari acuan dasar (baseline) yang akan berjalan. Namun, output ini juga menunjukkan bahwa rencana yang berbiaya lebih rendah ditemukan. Dalam hal ini, Anda perlu mengambil rencana berbiaya minimum yang baru ini dengan mengaktifkan pengambilan rencana otomatis seperti yang dijelaskan dalam Mengambil rencana secara otomatis.

Rencana baru selalu diambil oleh pengoptimisasi sebagai `Unapproved`. Gunakan fungsi `apg_plan_mgmt.evolve_plan_baselines` untuk membandingkan rencana dan mengubahnya menjadi disetujui, ditolak, atau dinonaktifkan. Untuk informasi selengkapnya, lihat Mengevaluasi performa rencana.

Bagaimana cara pengoptimisasi memilih rencana yang akan dijalankan

Biaya rencana eksekusi adalah perkiraan yang dibuat oleh pengoptimisasi untuk membandingkan beberapa rencana yang berbeda. Saat menghitung biaya rencana, pengoptimisasi mempertimbangkan faktor-faktor seperti operasi CPU dan I/O yang diperlukan oleh rencana tersebut. Untuk mempelajari selengkapnya tentang perkiraan biaya perencana kueri PostgreSQL, lihat Query Planning dalam dokumentasi PostgreSQL.

Gambar berikut menunjukkan bagaimana rencana dipilih untuk pernyataan SQL tertentu ketika manajemen rencana kueri aktif, dan ketika tidak.


Optimizer's plan
A. Run Optimizer's plan
SQL statement
Start
1. Generate minimum-cost plan
2. Query plan management active ?
No
Query Plan Management
Yes
3a. Plan in dba_plans view ?
No
Yes
4. Capture Plan Processing
First plan (P1) status = Approved
Plan (P1+n) status = Unapproved
3b. Plan Approved or Preferred ?
No
Yes
5. Use plan baselines on ?
No
Yes
6a. Plan in dba_plans view?
No
6b. Capture the optimizer's plan as new Unapproved
Yes
7a. Plan Unapproved ?
Yes
7b. Plan cost < threshold for Unapproved plans ?
Yes
No
No
8a. Valid and enabled Preferred Plans ?
Yes
8b. Choose minimum-cost Preferred plan.
B. Run Preferred plan
No
9a. Valid and enabled Approved Plans ?
Yes
9b. Choose minimum-cost Approved plan.
C. Run Approved plan
No

Alurnya adalah sebagai berikut:

1. Pengoptimisasi menghasilkan rencana berbiaya minimum untuk pernyataan SQL.
2. Jika manajemen rencana kueri tidak aktif, rencana pengoptimisasi akan segera dijalankan (A. Jalankan rencana Pengoptimisasi). Manajemen rencana kueri tidak aktif ketika parameter `apg_plan_mgmt.capture_plan_baselines` dan `apg_plan_mgmt.use_plan_baselines` berada pada pengaturan default-nya (masing-masing “off” dan “false”).

   Jika tidak, manajemen rencana kueri aktif. Dalam hal ini, pernyataan SQL dan rencana pengoptimisasi untuk pernyataan tersebut akan dinilai lebih lanjut sebelum rencana dipilih.

   > **Tip**
   >
   > Pengguna basis data dengan peran `apg_plan_mgmt` dapat secara proaktif membandingkan rencana, mengubah status rencana, dan memaksa penggunaan rencana tertentu sesuai kebutuhan. Untuk informasi selengkapnya, lihat Mengelola rencana eksekusi Aurora PostgreSQL.

3. Pernyataan SQL mungkin sudah memiliki rencana yang disimpan oleh manajemen rencana kueri di masa lalu. Rencana disimpan di `apg_plan_mgmt.dba_plans`, bersama dengan informasi tentang pernyataan SQL yang digunakan untuk membuatnya. Informasi tentang rencana akan menyertakan statusnya. Status rencana dapat menentukan apakah rencana tersebut digunakan atau tidak, sebagai berikut.
   a. Jika rencana tidak termasuk di antara rencana yang disimpan untuk pernyataan SQL, berarti ini adalah pertama kalinya rencana spesifik ini dihasilkan oleh pengoptimisasi untuk pernyataan SQL yang diberikan. Rencana dikirim ke Pemrosesan Pengambilan Rencana (4).
   b. Jika rencana termasuk di antara rencana yang disimpan dan statusnya Disetujui atau Pilihan, rencana ini akan dijalankan (A. Jalankan rencana Pengoptimisasi).

      Jika rencana termasuk di antara rencana yang disimpan, tetapi tidak berstatus Disetujui atau Pilihan, rencana ini akan dikirim ke Pemrosesan Pengambilan Rencana (4).
4. Ketika rencana diambil pertama kalinya untuk pernyataan SQL tertentu, status rencana ini selalu diatur ke Disetujui (P1). Jika pengoptimisasi selanjutnya menghasilkan rencana yang sama untuk pernyataan SQL yang sama, status rencana tersebut diubah menjadi Tidak Disetujui (P1+n).

   Dengan rencana diambil dan statusnya diperbarui, evaluasi berlanjut pada langkah berikutnya (5).

5. Acuan dasar rencana terdiri dari riwayat pernyataan SQL dan rencananya dalam berbagai status. Manajemen rencana kueri dapat mempertimbangkan acuan dasar ini saat memilih rencana, tergantung pada apakah opsi Gunakan acuan dasar rencana diaktifkan atau tidak, sebagai berikut.
   - Gunakan acuan dasar rencana “tidak aktif” ketika parameter `apg_plan_mgmt.use_plan_baselines` diatur ke nilai default (`false`). Rencana ini tidak akan dibandingkan dengan acuan dasar (baseline) sebelum dijalankan (A. Jalankan rencana Pengoptimisasi).
   - Gunakan acuan dasar rencana “aktif” saat parameter `apg_plan_mgmt.use_plan_baselines` diatur ke `true`. Rencana tersebut dinilai lebih lanjut menggunakan acuan dasar (6).
6. Rencana tersebut dibandingkan dengan rencana lain untuk pernyataan di acuan dasar.
   a. Jika rencana pengoptimisasi termasuk di antara rencana di acuan dasar, statusnya diperiksa (7a).
   b. Jika rencana pengoptimisasi tidak termasuk di antara rencana dalam acuan dasar, rencana ini akan ditambahkan ke rencana untuk pernyataan tersebut sebagai rencana `Unapproved` baru.
7. Status rencana diperiksa hanya untuk menentukan apakah statusnya Tidak Disetujui.
   a. Jika status rencana Tidak Disetujui, perkiraan biaya rencana ini dibandingkan dengan perkiraan biaya yang ditentukan untuk ambang batas rencana eksekusi yang tidak disetujui.
      - Jika perkiraan biaya rencana di bawah ambang batas, pengoptimisasi akan menggunakannya meskipun rencana ini Tidak Disetujui (A. Jalankan rencana Pengoptimisasi). Umumnya, pengoptimisasi tidak akan menjalankan rencana yang Tidak Disetujui. Namun, ketika parameter `apg_plan_mgmt.unapproved_plan_execution_threshold` menentukan nilai ambang batas biaya, pengoptimisasi akan membandingkan biaya rencana yang Tidak Disetujui dengan ambang batas ini. Jika perkiraan biaya lebih rendah dari ambang batas, pengoptimisasi akan menjalankan rencana. Untuk informasi selengkapnya, lihat apg_plan_mgmt.unapproved_plan_execution_threshold.
      - Jika perkiraan biaya rencana tidak di bawah ambang batas, atribut lainnya dari rencana ini akan diperiksa (8a).
   b. Jika status rencana adalah apa pun selain Tidak Disetujui, atribut lainnya akan diperiksa (8a).
8. Pengoptimisasi tidak akan menggunakan rencana yang dinonaktifkan. Rencana yang dinonaktifkan adalah rencana yang atribut `enable`-nya diatur ke 'f' (false). Pengoptimisasi juga tidak akan menggunakan rencana yang memiliki status Ditolak.

Pengoptimisasi tidak dapat menggunakan rencana apa pun yang tidak valid. Rencana dapat menjadi tidak valid dari waktu ke waktu ketika objek yang diandalkannya, misalnya indeks dan partisi tabel, dihapus.

a. Jika pernyataan memiliki rencana Pilihan yang diaktifkan dan valid, pengoptimisasi akan memilih rencana berbiaya minimum di antara rencana Pilihan yang disimpan untuk pernyataan SQL ini. Pengoptimisasi kemudian menjalankan rencana Pilihan berbiaya minimum.

b. Jika pernyataan tidak memiliki rencana Pilihan yang diaktifkan dan valid, pernyataan ini akan dinilai pada langkah berikutnya (9).

9. Jika pernyataan memiliki rencana Disetujui yang diaktifkan dan valid, pengoptimisasi akan memilih rencana berbiaya minimum dari berbagai rencana Disetujui yang disimpan untuk pernyataan SQL ini. Pengoptimisasi kemudian menjalankan rencana berbiaya minimum yang Disetujui.

   Jika pernyataan tidak memiliki rencana Disetujui yang valid dan diaktifkan, pengoptimisasi akan menggunakan rencana berbiaya minimum (A. Jalankan rencana Pengoptimisasi).

Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans

Pengguna dan administrator basis data yang telah diberi peran `apg_plan_mgmt` dapat melihat dan mengelola rencana yang disimpan dalam `apg_plan_mgmt.dba_plans`. Administrator klaster DB Aurora PostgreSQL (seseorang dengan izin `rds_superuser`) harus secara eksplisit memberikan peran ini kepada pengguna basis data yang perlu menggunakan manajemen rencana kueri.

Tampilan `apg_plan_mgmt` berisi riwayat rencana untuk semua pernyataan SQL terkelola untuk setiap basis data pada instans penulis dari klaster DB PostgreSQL Aurora. Tampilan ini memungkinkan Anda memeriksa rencana, statusnya, kapan terakhir digunakan, dan semua detail relevan lainnya.

Seperti yang dibahas dalam Normalisasi dan hash SQL, setiap rencana terkelola diidentifikasi oleh nilai hash SQL dan nilai hash rencana. Dengan pengidentifikasi ini, Anda dapat menggunakan alat seperti Wawasan Performa Amazon RDS untuk melacak performa rencana individu. Untuk informasi selengkapnya tentang Wawasan Performa, lihat Menggunakan Wawasan Performa Amazon RDS.

Menampilkan daftar rencana yang dikelola

Untuk menampilkan daftar rencana terkelola, gunakan SELECT pada tampilan `apg_plan_mgmt.dba_plans`. Contoh berikut menampilkan beberapa kolom dalam tampilan `dba_plans` seperti `status`, yang mengidentifikasi rencana yang disetujui dan tidak disetujui.

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;

 sql_hash   | plan_hash |   status   | enabled | stmt_name
------------+-----------+------------+---------+------------
 1984047223 | 512153379 | Approved   | t       | rangequery
 1984047223 | 512284451 | Unapproved | t       | rangequery
 (2 rows)
```

Agar mudah dibaca, kueri dan output yang ditampilkan hanya mencantumkan sebagian kecil kolom dari tampilan `dba_plans`. Untuk informasi selengkapnya, lihat Referensi untuk tampilan apg_plan_mgmt.dba_plans.

Mengelola rencana eksekusi Aurora PostgreSQL

Manajemen rencana kueri menyediakan teknik dan fungsi untuk menambahkan, memelihara, dan meningkatkan rencana eksekusi.

Mengevaluasi performa rencana

Setelah pengoptimal mengambil rencana sebagai tidak disetujui, gunakan fungsi `apg_plan_mgmt.evolve_plan_baselines` untuk membandingkan rencana berdasarkan performanya yang sebenarnya. Bergantung pada hasil eksperimen performa Anda, Anda dapat mengubah status rencana dari tidak disetujui ke disetujui atau ditolak. Anda dapat memutuskan untuk menggunakan fungsi `apg_plan_mgmt.evolve_plan_baselines` untuk menonaktifkan rencana secara sementara jika tidak memenuhi persyaratan Anda.

Menyetujui rencana yang lebih baik

Contoh berikut menunjukkan cara mengubah status rencana terkelola menjadi disetujui menggunakan fungsi `apg_plan_mgmt.evolve_plan_baselines`.

```
SELECT apg_plan_mgmt.evolve_plan_baselines (
   sql_hash,
   plan_hash,
   min_speedup_factor := 1.0,
   action := 'approve'
)
FROM apg_plan_mgmt.dba_plans WHERE status = 'Unapproved';
```

```
NOTICE:     rangequery (1,10000)
NOTICE:     Baseline   [ Planning time 0.761 ms, Execution time 13.261 ms]
NOTICE:     Baseline+1 [ Planning time 0.204 ms, Execution time 8.956 ms]
NOTICE:     Total time benefit: 4.862 ms, Execution time benefit: 4.305 ms
NOTICE:     Unapproved -> Approved
evolve_plan_baselines
----------------------
0
(1 row)
```

Output menampilkan laporan performa untuk pernyataan `rangequery` dengan ikatan parameter 1 dan 10.000. Rencana baru yang belum disetujui (`Baseline+1`) lebih baik daripada rencana terbaik yang telah disetujui sebelumnya (`Baseline`). Untuk mengonfirmasi bahwa rencana yang baru sekarang berstatus `Approved`, periksa tampilan `apg_plan_mgmt.dba_plans`.

```
SELECT sql_hash, plan_hash, status, enabled, stmt_name
FROM apg_plan_mgmt.dba_plans;
```

```
sql_hash  | plan_hash |  status  | enabled | stmt_name
------------+-----------+----------+---------+------------
1984047223 | 512153379 | Approved | t       | rangequery
1984047223 | 512284451 | Approved | t       | rangequery
(2 rows)
```

Rencana terkelola kini mencakup dua rencana yang disetujui yang merupakan acuan dasar rencana pernyataan. Anda juga dapat memanggil fungsi `apg_plan_mgmt.set_plan_status` untuk mengatur langsung bidang status rencana ke `'Approved'`, `'Rejected'`, `'Unapproved'`, atau `'Preferred'`.

Menolak atau menonaktifkan rencana yang lebih lambat

Untuk menolak atau menonaktifkan rencana, teruskan `'reject'` atau `'disable'` sebagai parameter tindakan ke fungsi `apg_plan_mgmt.evolve_plan_baselines`. Contoh ini menonaktifkan setiap rencana `Unapproved` yang diambil yang lebih lambat setidaknya 10 persen dari rencana `Approved` terbaik untuk pernyataan.

```
SELECT apg_plan_mgmt.evolve_plan_baselines(
sql_hash,  -- The managed statement ID
plan_hash, -- The plan ID
1.1,       -- number of times faster the plan must be
```

```
'disable'  -- The action to take. This sets the enabled field to false.
)
FROM apg_plan_mgmt.dba_plans
WHERE status = 'Unapproved' AND   -- plan is Unapproved
origin = 'Automatic';       -- plan was auto-captured
```

Anda juga dapat secara langsung mengatur rencana ke ditolak atau dinonaktifkan. Untuk secara langsung mengatur bidang yang diaktifkan untuk rencana ke `true` atau `false`, panggil fungsi `apg_plan_mgmt.set_plan_enabled`. Untuk secara langsung mengatur bidang status rencana ke `'Approved'`, `'Rejected'`, `'Unapproved'`, atau `'Preferred'`, panggil fungsi `apg_plan_mgmt.set_plan_status`.

Memvalidasi rencana

Gunakan fungsi `apg_plan_mgmt.validate_plans` untuk menghapus atau menonaktifkan rencana yang tidak valid.

Rencana dapat menjadi tidak valid atau stale ketika objek yang diandalkan rencana tersebut dihapus, misalnya indeks atau tabel. Namun, rencana mungkin hanya menjadi tidak valid untuk sementara jika objek yang dihapus dibuat kembali. Jika rencana yang tidak valid dapat menjadi valid di lain waktu, mungkin Anda sebaiknya memilih untuk menonaktifkan rencana yang tidak valid atau tidak melakukan apa-apa daripada menghapusnya.

Untuk menemukan dan menghapus semua rencana yang tidak valid dan belum digunakan dalam kurun waktu seminggu, gunakan fungsi `apg_plan_mgmt.validate_plans` sebagai berikut.

```
SELECT apg_plan_mgmt.validate_plans(sql_hash, plan_hash, 'delete')
FROM apg_plan_mgmt.dba_plans
WHERE last_used < (current_date - interval '7 days');
```

Untuk mengaktifkan atau menonaktifkan rencana secara langsung, gunakan fungsi `apg_plan_mgmt.set_plan_enabled`.

Memperbaiki rencana menggunakan pg_hint_plan

Pengoptimal kueri dirancang dengan baik untuk menemukan rencana yang optimal untuk semua pernyataan, dan pada sebagian besar kasus, pengoptimal menemukan rencana yang baik. Namun, terkadang Anda mungkin tahu bahwa ada rencana yang jauh lebih baik dari yang dihasilkan oleh pengoptimal. Dua cara yang disarankan agar pengoptimal menghasilkan rencana yang diinginkan

adalah menggunakan ekstensi pg_hint_plan atau mengatur variabel Grand Unified Configuration (GUC) di PostgreSQL:

- Ekstensi pg_hint_plan – Menentukan "petunjuk" untuk memodifikasi cara perencana bekerja menggunakan ekstensi pg_hint_plan PostgreSQL. Untuk menginstal dan mempelajari selengkapnya tentang cara menggunakan ekstensi pg_hint_plan, lihat dokumentasi pg_hint_plan.
- Variabel GUC – Menulis ulang satu atau beberapa parameter model biaya atau parameter pengoptimal lainnya, seperti from_collapse_limit atau GEQO_threshold.

Saat Anda menggunakan salah satu teknik ini untuk memaksa pengoptimal kueri agar menggunakan suatu rencana, Anda juga dapat menggunakan manajemen rencana kueri untuk mengambil dan memberlakukan penggunaan rencana baru.

Anda dapat menggunakan ekstensi pg_hint_plan untuk mengubah urutan join, metode join, atau jalur akses untuk pernyataan SQL. Anda menggunakan komentar SQL dengan sintaksis pg_hint_plan khusus untuk memodifikasi cara pengoptimal membuat rencana. Misalnya, anggaplah pernyataan SQL yang bermasalah memiliki join dua arah.

```
SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

Kemudian, anggaplah bahwa pengoptimal memilih urutan join (t1, t2), tetapi Anda tahu bahwa urutan join (t2, t1) lebih cepat. Petunjuk berikut memaksa pengoptimal untuk menggunakan urutan join yang lebih cepat, (t2, t1). Sertakan EXPLAIN sehingga pengoptimal menghasilkan rencana untuk pernyataan SQL, tetapi tanpa menjalankan pernyataan ini. (Output tidak ditampilkan.)

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

Langkah-langkah berikut menunjukkan cara menggunakan pg_hint_plan.

Untuk mengubah rencana yang dibuat oleh pengoptimal dan mengambil rencana menggunakan pg_hint_plan

1. Aktifkan mode pengambilan manual.

```
SET apg_plan_mgmt.capture_plan_baselines = manual;
```

2. Tentukan petunjuk untuk pernyataan SQL yang ingin diamati.

```
/*+ Leading ((t2 t1)) */ EXPLAIN SELECT *
FROM t1, t2
WHERE t1.id = t2.id;
```

Setelah proses ini berjalan, pengoptimal mengambil rencana dalam tampilan `apg_plan_mgmt.dba_plans`. Rencana yang diambil tidak mencakup sintaksis komentar `pg_hint_plan` khusus karena manajemen rencana kueri menormalisasi pernyataan dengan menghapus komentar awal.

3. Lihat rencana terkelola menggunakan tampilan `apg_plan_mgmt.dba_plans`.

```
SELECT sql_hash, plan_hash, status, sql_text, plan_outline
FROM apg_plan_mgmt.dba_plans;
```

4. Atur status rencana menjadi `Preferred`. Tindakan ini memastikan bahwa pengoptimal memilih untuk menjalankannya dan tidak memilih dari rencana yang disetujui, saat rencana berbiaya minimum belum berstatus `Approved` atau `Preferred`.

```
SELECT apg_plan_mgmt.set_plan_status(sql-hash, plan-hash, 'preferred' );
```

5. Nonaktifkan pengambilan rencana manual dan berlakukan penggunaan rencana terkelola.

```
SET apg_plan_mgmt.capture_plan_baselines = false;
SET apg_plan_mgmt.use_plan_baselines = true;
```

Sekarang, ketika pernyataan SQL asli berjalan, pengoptimal memilih rencana `Approved` atau `Preferred`. Jika rencana berbiaya minimum tidak berstatus `Approved` atau `Preferred`, pengoptimal akan memilih rencana `Preferred`.

Menghapus rencana

Rencana dihapus secara otomatis jika belum digunakan selama lebih dari sebulan, khususnya, 32 hari. Ini adalah pengaturan default untuk parameter `apg_plan_mgmt.plan_retention_period`. Anda dapat mengubah periode retensi rencana ke periode waktu yang lebih lama, atau ke

periode waktu yang lebih pendek mulai dari nilai 1. Menentukan jumlah hari sejak rencana terakhir digunakan yang dihitung dengan mengurangi tanggal saat ini dengan tanggal `last_used`. Tanggal `last_used` adalah tanggal terbaru saat pengoptimal memilih rencana sebagai rencana berbiaya minimum atau saat rencana tersebut dijalankan. Tanggal disimpan untuk rencana dalam tampilan `apg_plan_mgmt.dba_plans`.

Kami menyarankan Anda menghapus rencana yang belum digunakan dalam waktu lama atau yang tidak berguna. Setiap rencana memiliki tanggal `last_used` yang diperbarui oleh pengoptimal setiap kali pengoptimal menjalankan rencana atau memilih rencana sebagai rencana berbiaya minimum untuk suatu pernyataan. Periksa tanggal `last_used` terakhir untuk mengidentifikasi rencana yang dapat Anda hapus dengan aman.

Kueri berikut menampilkan tabel tiga kolom dengan jumlah total rencana, rencana yang gagal dihapus, dan rencana yang berhasil dihapus. Kueri tersebut memiliki kueri bersarang yang merupakan contoh cara menggunakan fungsi `apg_plan_mgmt.delete_plan` untuk menghapus semua rencana yang belum dipilih sebagai rencana berbiaya minimum dalam 31 hari terakhir dan statusnya bukan `Rejected`

```
SELECT (SELECT COUNT(*) from apg_plan_mgmt.dba_plans) total_plans,
       COUNT(*) FILTER (WHERE result = -1) failed_to_delete,
       COUNT(*) FILTER (WHERE result = 0) successfully_deleted
       FROM (
            SELECT apg_plan_mgmt.delete_plan(sql_hash, plan_hash) as result
            FROM apg_plan_mgmt.dba_plans
            WHERE last_used < (current_date - interval '31 days')
            AND status <> 'Rejected'
            ) as dba_plans ;
```

```
 total_plans | failed_to_delete | successfully_deleted
-------------+------------------+----------------------
           3 |                0 |                    2
```

Untuk informasi selengkapnya, lihat apg_plan_mgmt.delete_plan.

Untuk menghapus rencana yang tidak valid dan yang Anda perkirakan akan tetap tidak valid, gunakan fungsi `apg_plan_mgmt.validate_plans`. Fungsi ini memungkinkan Anda menghapus atau menonaktifkan rencana yang tidak valid. Untuk informasi selengkapnya, lihat Memvalidasi rencana.

> **Important**
>
> Jika Anda tidak menghapus rencana yang tidak berguna, Anda mungkin akan kehabisan memori bersama yang dialokasikan untuk manajemen rencana kueri. Untuk mengontrol berapa banyak memori yang tersedia untuk rencana terkelola, gunakan parameter `apg_plan_mgmt.max_plans`. Atur parameter ini di grup parameter DB kustom Anda lalu boot ulang instans DB Anda agar perubahan pada instans diterapkan. Untuk informasi selengkapnya, lihat parameter apg_plan_mgmt.max_plans.

Mengekspor dan mengimpor rencana

Anda dapat mengekspor rencana terkelola Anda dan mengimpornya ke dalam instans DB lain.

Untuk mengekspor rencana terkelola

Pengguna dapat menyalin subset tabel `apg_plan_mgmt.plans` ke tabel lain, lalu menyimpannya menggunakan perintah pg_dump. Berikut adalah contohnya.

```
CREATE TABLE plans_copy AS SELECT *
FROM apg_plan_mgmt.plans [ WHERE predicates ] ;
```

```
% pg_dump --table apg_plan_mgmt.plans_copy -Ft mysourcedatabase > plans_copy.tar
```

```
DROP TABLE apg_plan_mgmt.plans_copy;
```

Untuk mengimpor rencana terkelola

1. Salin file .tar dari rencana terkelola yang diekspor ke sistem tempat rencana tersebut akan dipulihkan.
2. Gunakan perintah `pg_restore` untuk menyalin file .tar ke dalam tabel baru.

   ```
   % pg_restore --dbname mytargetdatabase -Ft plans_copy.tar
   ```
3. Gabungkan tabel `plans_copy` dengan tabel `apg_plan_mgmt.plans` seperti yang ditunjukkan dalam contoh berikut.

> **Note**
>
> Dalam beberapa kasus, Anda mungkin membuang dari satu versi ekstensi `apg_plan_mgmt` dan memulihkan ke versi yang berbeda. Dalam kasus ini, kolom di tabel rencana mungkin berbeda. Jika ya, tentukan nama kolom tersebut dengan eksplisit dan bukan menggunakan SELECT *.

```
INSERT INTO apg_plan_mgmt.plans SELECT * FROM plans_copy
 ON CONFLICT ON CONSTRAINT plans_pkey
 DO UPDATE SET
 status = EXCLUDED.status,
 enabled = EXCLUDED.enabled,
 -- Save the most recent last_used date
 --
 last_used = CASE WHEN EXCLUDED.last_used > plans.last_used
 THEN EXCLUDED.last_used ELSE plans.last_used END,
 -- Save statistics gathered by evolve_plan_baselines, if it ran:
 --
 estimated_startup_cost = EXCLUDED.estimated_startup_cost,
 estimated_total_cost = EXCLUDED.estimated_total_cost,
 planning_time_ms = EXCLUDED.planning_time_ms,
 execution_time_ms = EXCLUDED.execution_time_ms,
 total_time_benefit_ms = EXCLUDED.total_time_benefit_ms,
 execution_time_benefit_ms = EXCLUDED.execution_time_benefit_ms;
```

4. Muat ulang rencana terkelola ke dalam memori bersama dan hapus tabel rencana sementara.

```
SELECT apg_plan_mgmt.reload(); -- refresh shared memory
DROP TABLE plans_copy;
```

Referensi untuk manajemen rencana kueri Aurora PostgreSQL

Di bagian berikut ini, Anda dapat menemukan informasi referensi untuk beberapa fitur dan fungsionalitas manajemen rencana kueri Aurora PostgreSQL.

Topik

- Referensi parameter untuk manajemen rencana kueri Aurora PostgreSQL

- Referensi fungsi untuk manajemen rencana kueri Aurora PostgreSQL
- Referensi untuk tampilan apg_plan_mgmt.dba_plans

Referensi parameter untuk manajemen rencana kueri Aurora PostgreSQL

Anda dapat mengatur preferensi Anda untuk ekstensi `apg_plan_mgmt` dengan menggunakan parameter yang tercantum di bagian ini. Hal ini tersedia dalam parameter klaster DB kustom dan grup parameter DB yang terkait dengan klaster DB Aurora PostgreSQL Anda. Parameter ini mengontrol perilaku fitur manajemen rencana kueri dan bagaimana pengaruhnya terhadap pengoptimisasi. Untuk informasi tentang pengaturan manajemen rencana kueri, lihat Mengaktifkan manajemen rencana kueri Aurora PostgreSQL. Perubahan pada parameter berikut tidak berpengaruh jika ekstensi `apg_plan_mgmt` tidak diatur seperti yang dijelaskan dalam bagian tersebut. Untuk informasi tentang memodifikasi parameter, lihat Mengubah parameter dalam grup parameter klaster DB dan Bekerja dengan grup parameter DB dalam instance DB.

Parameter

- apg_plan_mgmt.capture_plan_baselines
- apg_plan_mgmt.plan_capture_threshold
- apg_plan_mgmt.explain_hashes
- apg_plan_mgmt.log_plan_enforcement_result
- apg_plan_mgmt.max_databases
- apg_plan_mgmt.max_plans
- apg_plan_mgmt.plan_hash_version
- apg_plan_mgmt.plan_retention_period
- apg_plan_mgmt.unapproved_plan_execution_threshold
- apg_plan_mgmt.use_plan_baselines
- auto_explain.hashes

apg_plan_mgmt.capture_plan_baselines

Mengambil rencana eksekusi kueri yang dihasilkan oleh pengoptimisasi untuk setiap pernyataan SQL dan menyimpannya dalam tampilan `dba_plans`. Secara default, jumlah maksimum rencana yang dapat disimpan adalah 10.000 seperti yang ditentukan oleh parameter `apg_plan_mgmt.max_plans`. Untuk informasi referensi, lihat apg_plan_mgmt.max_plans.

Anda dapat mengatur parameter ini dalam grup parameter klaster DB kustom atau dalam grup parameter DB kustom. Mengubah nilai parameter ini tidak memerlukan boot ulang.

Default	Nilai yang diizinkan	Deskripsi
off	automatic	Mengaktifkan pengambilan rencana untuk semua basis data pada instans DB. Mengumpulkan rencana untuk setiap pernyataan SQL yang berjalan dua kali atau lebih. Gunakan pengaturan ini untuk beban kerja yang besar atau berkembang untuk memberikan stabilitas rencana.
	manual	Mengaktifkan pengambilan rencana untuk pernyataan berikutny a saja, sampai Anda menonaktifkannya lagi. Pengaturan ini memungkinkan Anda mengambil rencana eksekusi kueri untuk pernyataan SQL kritis tertentu saja atau untuk kueri bermasalah yang diketahui.
	off	Menonaktifkan pengambilan rencana.

Untuk informasi selengkapnya, lihat Mengambil rencana eksekusi Aurora PostgreSQL.

apg_plan_mgmt.plan_capture_threshold

Menentukan ambang batas sehingga jika total biaya rencana eksekusi kueri di bawah ambang batas, rencana tidak akan diambil dalam tampilan `apg_plan_mgmt.dba_plans`.

Mengubah nilai parameter ini tidak memerlukan boot ulang.

Default	Nilai yang diizinkan	Deskripsi
0	0 - 1.79769e+308	Menetapkan ambang batas total biaya eksekusi rencana kueri `apg_plan_mgmt` untuk mengambil rencana.

Untuk informasi selengkapnya, lihat Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans.

apg_plan_mgmt.explain_hashes

Menentukan apakah `EXPLAIN [ANALYZE]` menunjukkan `sql_hash` dan `plan_hash` pada akhir output-nya. Mengubah nilai parameter ini tidak memerlukan boot ulang.

Default	Nilai yang diizinkan	Deskripsi
0	0 (aktif)	EXPLORE tidak menampilkan sql_hash dan plan_hash tanpa opsi hashes true.
	1 (nonaktif)	EXPLORE menampilkan sql_hash dan plan_hash tanpa opsi hashes true.

apg_plan_mgmt.log_plan_enforcement_result

Menentukan apakah hasil harus dicatat untuk melihat apakah rencana yang dikelola QPM digunakan dengan benar. Ketika rencana generik yang disimpan digunakan, tidak akan ada catatan yang ditulis dalam file log. Mengubah nilai parameter ini tidak memerlukan boot ulang.

Default	Nilai yang diizinkan	Deskripsi
none	none	Tidak menunjukkan hasil pemberlakuan rencana apa pun dalam file log.
	on_error	Hanya menampilkan hasil pemberlakuan rencana dalam file log ketika QPM gagal menggunakan rencana terkelola.
	all	Menampilkan semua hasil pemberlakuan rencana dalam file log termasuk keberhasilan dan kegagalan.

apg_plan_mgmt.max_databases

Menentukan jumlah maksimum basis data pada instans Penulis klaster DB Aurora PostgreSQL Anda yang dapat menggunakan manajemen rencana kueri. Secara default, hingga 10 basis data dapat menggunakan manajemen rencana kueri. Jika Anda memiliki lebih dari 10 basis data pada instans, Anda dapat mengubah nilai pengaturan ini. Untuk mengetahui jumlah basis data pada instans tertentu, hubungkan ke instans menggunakan `psql`. Kemudian, gunakan metacommand psql, `\l`, untuk menampilkan daftar basis data.

Mengubah nilai parameter ini akan mengharuskan Anda mem-boot ulang instans agar pengaturan diterapkan.

Default	Nilai yang diizinkan	Deskripsi
10	10-2147483647	Jumlah maksimum basis data yang dapat menggunakan manajemen rencana kueri pada instans.

Anda dapat mengatur parameter ini dalam grup parameter klaster DB kustom atau dalam grup parameter DB kustom.

apg_plan_mgmt.max_plans

Menetapkan jumlah maksimum pernyataan SQL yang dapat dipertahankan oleh manajer rencana kueri dalam tampilan `apg_plan_mgmt.dba_plans`. Sebaiknya atur parameter ini ke `10000` atau lebih tinggi untuk semua versi Aurora PostgreSQL.

Anda dapat mengatur parameter ini dalam grup parameter klaster DB kustom atau dalam grup parameter DB kustom. Mengubah nilai parameter ini akan mengharuskan Anda mem-boot ulang instans agar pengaturan diterapkan.

Default	Nilai yang diizinkan	Deskripsi
10000	10-2147483647	Jumlah maksimum rencana yang dapat disimpan dalam tampilan `apg_plan_mgmt.dba_plans` . Default untuk Aurora PostgreSQL versi 10 dan yang lebih lama adalah 1000.

Untuk informasi selengkapnya, lihat Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans.

apg_plan_mgmt.plan_hash_version

Menentukan kasus penggunaan yang dapat dicakup oleh perhitungan plan_hash sesuai rancangannya. Versi `apg_plan_mgmt.plan_hash_version` yang lebih tinggi mencakup semua fungsionalitas versi yang lebih rendah. Misalnya, versi 3 mencakup kasus penggunaan yang didukung oleh versi 2.

Perubahan pada nilai parameter ini harus diikuti dengan panggilan ke `apg_plan_mgmt.validate_plans('update_plan_hash')`. Parameter ini memperbarui nilai plan_hash di setiap basis data dengan apg_plan_mgmt diinstal dan entri dalam tabel rencana. Untuk informasi selengkapnya, lihat Memvalidasi rencana

Default	Nilai yang diizinkan	Deskripsi
1	1	Perhitungan plan_hash default.
	2	Perhitungan plan_hash yang dimodifikasi untuk dukungan multi-skema.
	3	Perhitungan plan_hash yang dimodifikasi untuk dukungan multi-skema dan dukungan tabel yang dipartisi.
	4	Perhitungan plan_hash yang dimodifikasi untuk operator paralel dan untuk mendukung simpul terwujud.

apg_plan_mgmt.plan_retention_period

Menentukan jumlah hari untuk mempertahankan rencana dalam tampilan `apg_plan_mgmt.dba_plans`, dan setelah itu, rencana tersebut akan dihapus secara otomatis. Secara default, rencana dihapus ketika 32 hari telah berlalu sejak rencana tersebut terakhir digunakan (kolom `last_used` dalam tampilan `apg_plan_mgmt.dba_plans`). Anda dapat mengubah pengaturan ini ke angka apa pun, 1 dan lebih tinggi.

Mengubah nilai parameter ini akan mengharuskan Anda mem-boot ulang instans agar pengaturan diterapkan.

Default	Nilai yang diizinkan	Deskripsi
32	1-2147483647	Jumlah hari maksimum sejak rencana terakhir digunakan sebelum dihapus.

Untuk informasi selengkapnya, lihat Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans.

apg_plan_mgmt.unapproved_plan_execution_threshold

Menentukan ambang batas biaya yang jika tidak tercapai, rencana yang Tidak Disetujui dapat digunakan oleh pengoptimisasi. Secara default ambang batas ini adalah 0, sehingga pengoptimisasi tidak akan menjalankan rencana yang Tidak Disetujui. Jika parameter ini diatur ke ambang batas biaya yang cukup rendah seperti 100, tidak akan ada overhead pemberlakuan rencana pada rencana sederhana (trivial). Anda juga dapat mengatur parameter ini ke nilai yang sangat besar seperti 10000000 menggunakan gaya manajemen rencana reaktif. Hal ini memungkinkan pengoptimisasi menggunakan semua rencana yang dipilih tanpa overhead pemberlakuan rencana. Namun, ketika rencana buruk ditemukan, Anda dapat menandainya secara manual sebagai “ditolak” sehingga tidak akan digunakan di lain waktu.

Nilai parameter ini merepresentasikan perkiraan biaya untuk menjalankan rencana yang diberikan. Jika rencana yang Tidak Disetujui di bawah perkiraan biaya tersebut, pengoptimisasi akan menggunakannya untuk pernyataan SQL. Anda dapat melihat rencana yang diambil dan statusnya (Disetujui, Tidak Disetujui) di tampilan `dba_plans`. Untuk mempelajari selengkapnya, lihat Memeriksa rencana kueri Aurora PostgreSQL dalam tampilan dba_plans.

Mengubah nilai parameter ini tidak memerlukan boot ulang.

Default	Nilai yang diizinkan	Deskripsi
0	0-2147483647	Perkiraan biaya rencana yang jika tidak tercapai, akan membuat rencana yang Tidak Disetujui digunakan.

Untuk informasi selengkapnya, lihat Menggunakan rencana terkelola Aurora PostgreSQL.

apg_plan_mgmt.use_plan_baselines

Menentukan bahwa pengoptimisasi harus menggunakan salah satu rencana Disetujui yang diambil dan disimpan dalam tampilan `apg_plan_mgmt.dba_plans`. Secara default, parameter ini tidak aktif (false), sehingga menyebabkan pengoptimisasi menggunakan rencana berbiaya minimum yang dihasilkannya tanpa penilaian lebih lanjut. Mengaktifkan parameter ini (mengaturnya ke true) akan memaksa pengoptimisasi untuk memilih rencana eksekusi kueri untuk pernyataan dari acuan dasar (baseline) rencananya. Untuk informasi selengkapnya, lihat Menggunakan rencana terkelola Aurora PostgreSQL. Untuk menemukan gambar yang menguraikan proses ini, lihat Bagaimana cara pengoptimisasi memilih rencana yang akan dijalankan.

Anda dapat mengatur parameter ini dalam grup parameter klaster DB kustom atau dalam grup parameter DB kustom. Mengubah nilai parameter ini tidak memerlukan boot ulang.

Default	Nilai yang diizinkan	Deskripsi
false	true	Menggunakan rencana yang Disetujui, Pilihan, atau Tidak Disetujui dari `apg_plan_mgmt.dba_plans` . Jika tidak ada yang memenuhi semua kriteria evaluasi untuk pengoptimisasi, maka pengoptimisasi dapat menggunakan rencana berbiaya minimum yang dihasilkannya sendiri. Untuk informasi selengkapnya, lihat Bagaimana cara pengoptimisasi memilih rencana yang akan dijalankan.
	false	Menggunakan rencana berbiaya minimum yang dihasilkan oleh pengoptimisasi.

Anda dapat mengevaluasi waktu respons dari berbagai rencana yang diambil dan mengubah status rencana, sesuai kebutuhan. Untuk informasi selengkapnya, lihat Mengelola rencana eksekusi Aurora PostgreSQL.

auto_explain.hashes

Menentukan apakah output auto_explain menampilkan sql_hash dan plan_hash. Mengubah nilai parameter ini tidak memerlukan boot ulang.

Default	Nilai yang diizinkan	Deskripsi
0 (nonaktif)	0 (nonaktif)	Hasil `auto_explain` tidak menunjukkan `sql_hash` dan `plan_hash` .
	1 (aktif)	Hasil `auto_explain` menunjukkan `sql_hash` dan `plan_hash` .

Referensi fungsi untuk manajemen rencana kueri Aurora PostgreSQL

Ekstensi `apg_plan_mgmt` menyediakan fungsi-fungsi berikut.

Fungsi

- apg_plan_mgmt.copy_outline
- apg_plan_mgmt.delete_plan
- apg_plan_mgmt.evolve_plan_baselines
- apg_plan_mgmt.get_explain_plan
- apg_plan_mgmt.plan_last_used
- apg_plan_mgmt.reload
- apg_plan_mgmt.set_plan_enabled
- apg_plan_mgmt.set_plan_status
- apg_plan_mgmt.update_plans_last_used
- apg_plan_mgmt.validate_plans

apg_plan_mgmt.copy_outline

Menyalin hash rencana dan outline rencana SQL tertentu ke hash rencana dan outline rencana SQL target, sehingga akan menimpa hash rencana dan outline rencana target. Fungsi ini tersedia dalam `apg_plan_mgmt` versi 2.3 dan rilis yang lebih tinggi.

Sintaksis

```
apg_plan_mgmt.copy_outline(
    source_sql_hash,
    source_plan_hash,
    target_sql_hash,
    target_plan_hash,
    force_update_target_plan_hash
)
```

Nilai yang ditampilkan

Menampilkan 0 ketika penyalinan berhasil. Menampilkan pengecualian untuk input yang tidak valid.

Parameter

Parameter	Deskripsi
`source_sql_hash`	ID `sql_hash` yang terkait dengan `plan_hash` yang akan disalin ke kueri target.

Parameter	Deskripsi
`source_plan_hash`	ID `plan_hash` yang akan disalin ke kueri target.
`target_sql_hash`	ID `sql_hash` kueri yang akan diperbarui dengan hash rencana dan outline rencana sumber.
`target_plan_hash`	ID `plan_hash` kueri yang akan diperbaru i dengan hash rencana dan outline rencana sumber.
`force_update_target_plan_hash`	(Opsional) ID `target_plan_hash` kueri diperbarui meskipun rencana sumber tidak dapat direproduksi untuk `target_sql_hash` . Ketika diatur ke true, fungsi ini dapat digunakan untuk menyalin rencana ke seluruh skema yang memiliki nama dan kolom relasi yang konsisten.

Catatan penggunaan

Fungsi ini memungkinkan Anda untuk menyalin hash rencana dan outline rencana yang menggunakan petunjuk ke pernyataan lain yang serupa, sehingga Anda tidak perlu menggunakan pernyataan petunjuk inline pada setiap peristiwa dalam pernyataan target. Fungsi ini memungkinkan Anda untuk menyalin hash rencana dan outline rencana yang menggunakan petunjuk ke pernyataan lain yang serupa, dan sehingga Anda tidak perlu menggunakan pernyataan petunjuk inline pada setiap peristiwa dalam pernyataan target.

apg_plan_mgmt.delete_plan

Menghapus rencana terkelola.

Sintaksis

```
apg_plan_mgmt.delete_plan(
    sql_hash,
    plan_hash
)
```

Nilai yang ditampilkan

Menampilkan 0 jika penghapusan berhasil atau -1 jika penghapusan gagal.

Parameter

Parameter	Deskripsi
`sql_hash`	ID `sql_hash` pernyataan SQL terkelola milik rencana.
`plan_hash`	ID `plan_hash` rencana terkelola.

apg_plan_mgmt.evolve_plan_baselines

Memverifikasi apakah rencana yang sudah disetujui lebih cepat atau apakah rencana yang diidentifikasi oleh pengoptimal kueri sebagai rencana berbiaya minimum yang lebih cepat.

Sintaksis

```
apg_plan_mgmt.evolve_plan_baselines(
    sql_hash,
    plan_hash,
    min_speedup_factor,
    action
)
```

Nilai yang ditampilkan

Jumlah rencana yang tidak lebih cepat dari rencana terbaik yang disetujui.

Parameter

Parameter	Deskripsi
`sql_hash`	ID `sql_hash` pernyataan SQL terkelola milik rencana.
`plan_hash`	ID `plan_hash` rencana terkelola. Gunakan NULL untuk mengindikasikan semua rencana yang memiliki nilai ID `sql_hash` yang sama.

Parameter	Deskripsi
`min_speedup_factor`	Faktor percepatan minimum adalah seberapa kali lebih cepat suatu rencana daripada rencana lain yang sudah disetujui agar rencana tersebut dapat disetujui. Alternatifnya, faktor ini dapat berupa berapa kali lebih lambat agar rencana harus ditolak atau dinonaktifkan. Ini adalah nilai float positif.
`action`	Tindakan yang harus dilakukan oleh fungsi tersebut. Nilai yang valid mencakup hal berikut. Huruf besar/kecil tidak berpengaruh. • `'disable'` – Menonaktifkan setiap rencana yang tidak memenuhi faktor percepatan minimum. • `'approve'` – Mengaktifkan setiap rencana yang memenuhi faktor percepatan minimum dan menetapkan statusnya menjadi `approved`. • `'reject'` – Untuk setiap rencana yang tidak memenuhi faktor percepatan minimum, menetapkan statusnya menjadi `rejected`. • NULL – Fungsi hanya menampilkan jumlah rencana yang tidak memiliki keuntungan performa karena tidak memenuhi faktor percepatan minimum.

Catatan penggunaan

Tetapkan rencana ke status disetujui, ditolak, atau dinonaktifkan berdasarkan apakah waktu perencanaan plus eksekusi lebih cepat daripada rencana yang disetujui berdasarkan faktor yang dapat Anda tetapkan. Parameter tindakan dapat diatur ke `'approve'` atau `'reject'` untuk secara otomatis menyetujui atau menolak rencana yang memenuhi kriteria performa. Alternatifnya, parameter dapat diatur ke '' (string kosong) untuk melakukan eksperimen performa dan membuat laporan, tetapi tidak mengambil tindakan.

Anda dapat mencegah fungsi `apg_plan_mgmt.evolve_plan_baselines` dijalankan kembali secara sia-sia untuk rencana tempat fungsi ini baru-baru ini dijalankan. Untuk melakukannya, batasi rencana hanya untuk rencana yang baru-baru ini dibuat yang belum disetujui. Alternatifnya, Anda dapat mencegah fungsi `apg_plan_mgmt.evolve_plan_baselines` dijalankan pada rencana yang disetujui yang memiliki stempel waktu `last_verified` baru-baru ini.

Lakukan eksperimen performa untuk membandingkan waktu perencanaan dan eksekusi untuk setiap rencana secara relatif terhadap rencana lain dalam acuan dasar (baseline). Dalam beberapa kasus, hanya ada satu rencana untuk satu pernyataan dan rencana ini disetujui. Dalam kasus seperti itu, bandingkan waktu perencanaan dan eksekusi rencana dengan waktu perencanaan dan eksekusi tanpa menggunakan rencana.

Keuntungan (atau kerugian) tambahan dari setiap rencana dicatat di tampilan `apg_plan_mgmt.dba_plans` dalam kolom `total_time_benefit_ms`. Jika nilainya positif, ada keuntungan performa yang dapat diukur untuk memasukkan rencana ini dalam acuan dasar.

Selain mengumpulkan waktu perencanaan dan eksekusi dari setiap rencana kandidat, kolom `last_verified` pada tampilan `apg_plan_mgmt.dba_plans` diperbarui dengan `current_timestamp`. Stempel waktu `last_verified` dapat digunakan agar fungsi ini tidak lagi dijalankan pada rencana yang performanya baru-baru ini diverifikasi.

apg_plan_mgmt.get_explain_plan

Menghasilkan teks dari pernyataan `EXPLAIN` untuk pernyataan SQL yang ditentukan.

Sintaksis

```
apg_plan_mgmt.get_explain_plan(
    sql_hash,
    plan_hash,
    [explainOptionList]
)
```

Nilai yang ditampilkan

Menampilkan statistik runtime untuk pernyataan SQL yang ditentukan. Gunakan tanpa `explainOptionList` untuk menampilkan rencana `EXPLAIN` sederhana.

Parameter

Parameter	Deskripsi
`sql_hash`	ID `sql_hash` pernyataan SQL terkelola milik rencana.
`plan_hash`	ID `plan_hash` rencana terkelola.

Parameter	Deskripsi
explainOptionList	Daftar opsi penjelasan yang dipisahkan dengan koma. Nilai yang valid termasuk 'analyze' , 'verbose' , 'buffers' , 'hashes', dan 'format json'. Jika explainOption List adalah NULL atau string kosong (''), fungsi ini menghasilkan pernyataan EXPLAIN, tanpa statistik.

Catatan penggunaan

Untuk explainOptionList, Anda dapat menggunakan salah satu dari opsi yang sama yang akan Anda gunakan dengan pernyataan EXPLAIN. Pengoptimal Aurora PostgreSQL menggabungkan daftar opsi yang Anda berikan ke pernyataan EXPLAIN.

apg_plan_mgmt.plan_last_used

Menampilkan tanggal last_used untuk rencana yang ditetapkan dari memori bersama.

> **Note**
>
> Nilai dalam memori bersama selalu terbaru pada instans DB primer dalam klaster DB. Nilai hanya secara berkala di-flush ke kolom last_used dalam tampilan apg_plan_mgmt.dba_plans.

Sintaksis

```
apg_plan_mgmt.plan_last_used(
    sql_hash,
    plan_hash
)
```

Nilai yang ditampilkan

Menampilkan tanggal last_used.

Parameter

Parameter	Deskripsi
`sql_hash`	ID `sql_hash` pernyataan SQL terkelola milik rencana.
`plan_hash`	ID `plan_hash` rencana terkelola.

apg_plan_mgmt.reload

Memuat ulang rencana ke dalam memori bersama dari tampilan `apg_plan_mgmt.dba_plans`.

Sintaksis

```
apg_plan_mgmt.reload()
```

Nilai yang ditampilkan

Tidak ada.

Parameter

Tidak ada.

Catatan penggunaan

Memanggil `reload` untuk situasi-situasi berikut:

- Gunakan untuk me-refresh memori bersama replika hanya baca, dan bukan menunggu adanya rencana baru untuk disebarkan ke replika.
- Gunakan setelah mengimpor rencana terkelola.

apg_plan_mgmt.set_plan_enabled

Mengaktifkan atau menonaktifkan rencana terkelola.

Sintaksis

```
apg_plan_mgmt.set_plan_enabled(
```

```
    sql_hash,
    plan_hash,
    [true | false]
)
```

Nilai yang ditampilkan

Menampilkan 0 jika pengaturan berhasil atau -1 jika pengaturan gagal.

Parameter

Parameter	Deskripsi
`sql_hash`	ID `sql_hash` pernyataan SQL terkelola milik rencana.
`plan_hash`	ID `plan_hash` rencana terkelola.
`enabled`	Nilai Boolean benar atau salah: • Nilai `true` mengaktifkan rencana. • Nilai `false` menonaktifkan rencana.

apg_plan_mgmt.set_plan_status

Mengatur status rencana terkelola ke `Approved`, `Unapproved`, `Rejected`, atau `Preferred`.

Sintaksis

```
apg_plan_mgmt.set_plan_status(
    sql_hash,
    plan_hash,
    status
)
```

Nilai yang ditampilkan

Menampilkan 0 jika pengaturan berhasil atau -1 jika pengaturan gagal.

Parameter

Parameter	Deskripsi
`sql_hash`	ID `sql_hash` pernyataan SQL terkelola milik rencana.
`plan_hash`	ID `plan_hash` rencana terkelola.
`status`	String dengan salah satu nilai berikut: • `'Approved'` • `'Unapproved'` • `'Rejected'` • `'Preferred'` Huruf besar/kecil yang Anda gunakan tidak berpengaruh, tetapi nilai status menggunakan huruf besar di awal dalam tampilan `apg_plan_mgmt.dba_plans` . Untuk informasi selengkapnya tentang nilai-nilai tersebut, lihat `status` dalam Referensi untuk tampilan apg_plan_mgmt.dba_plans.

apg_plan_mgmt.update_plans_last_used

Segera memperbarui tabel rencana dengan tanggal `last_used` yang disimpan dalam memori bersama.

Sintaksis

```
apg_plan_mgmt.update_plans_last_used()
```

Nilai yang ditampilkan

Tidak ada.

Parameter

Tidak ada.

Catatan penggunaan

Panggil `update_plans_last_used` untuk memastikan kueri terhadap kolom `dba_plans.last_used` menggunakan informasi terbaru. Jika tanggal `last_used` tidak segera diperbarui, proses latar belakang akan memperbarui tabel rencana dengan tanggal `last_used` sekali setiap jam (secara default).

Misalnya, jika pernyataan dengan `sql_hash` tertentu mulai berjalan lambat, Anda dapat menentukan rencana mana yang dieksekusi untuk pernyataan tersebut sejak regresi performa dimulai. Untuk melakukannya, pertama-tama lakukan flushing data dalam memori bersama ke disk sehingga tanggal `last_used` menjadi terkini, lalu jalankan kueri untuk semua rencana `sql_hash` dari pernyataan yang memiliki regresi performa. Dalam kueri, pastikan tanggal `last_used` lebih besar dari atau sama dengan tanggal saat regresi performa dimulai. Kueri tersebut mengidentifikasi rencana atau kumpulan rencana yang mungkin bertanggung jawab atas regresi performa. Anda dapat menggunakan `apg_plan_mgmt.get_explain_plan` dengan `explainOptionList` yang diatur ke `verbose, hashes`. Anda juga dapat menggunakan `apg_plan_mgmt.evolve_plan_baselines` untuk menganalisis rencana dan rencana alternatif apa pun yang mungkin beperforma lebih baik.

Fungsi `update_plans_last_used` hanya berpengaruh pada instans DB primer klaster DB.

apg_plan_mgmt.validate_plans

Memvalidasi bahwa pengoptimal masih dapat membuat ulang rencana. Pengoptimal memvalidasi rencana `Approved`, `Unapproved`, dan `Preferred`, baik ketika rencana tersebut diaktifkan maupun dinonaktifkan. Rencana `Rejected` tidak divalidasi. Secara opsional, Anda dapat menggunakan fungsi `apg_plan_mgmt.validate_plans` untuk menghapus atau menonaktifkan rencana yang tidak valid.

Sintaksis

```
apg_plan_mgmt.validate_plans(
    sql_hash,
    plan_hash,
    action)

apg_plan_mgmt.validate_plans(
    action)
```

Nilai yang ditampilkan

Jumlah rencana tidak valid.

Parameter

Parameter	Deskripsi
`sql_hash`	ID `sql_hash` pernyataan SQL terkelola milik rencana.
`plan_hash`	ID `plan_hash` rencana terkelola. Gunakan NULL untuk merujuk pada semua rencana dengan nilai ID `sql_hash` yang sama.
`action`	Tindakan yang harus dilakukan fungsi untuk rencana yang tidak valid. Nilai string yang valid mencakup hal berikut. Huruf besar/kecil tidak berpengaruh. • `'disable'` – Setiap rencana yang tidak valid dinonaktifkan. • `'delete'` – Setiap rencana yang tidak valid dihapus. • `'update_plan_hash'` – Memperbarui ID `plan_hash` untuk rencana yang tidak dapat direproduksi dengan tepat. Hal ini juga memungkinkan Anda memperbaiki rencana dengan menulis ulang SQL. Anda kemudian dapat mendaftarkan rencana yang baik sebagai rencana `Approved` untuk SQL asli. • NULL – Fungsi hanya menampilkan jumlah rencana yang tidak valid. Tidak ada tindakan lain yang dilakukan. • '' – String kosong menghasilkan pesan yang menunjukkan jumlah rencana yang valid dan tidak valid. Nilai lainnya diperlakukan seperti string kosong.

Catatan penggunaan

Gunakan form `validate_plans(action)` untuk memvalidasi semua rencana terkelola untuk semua pernyataan terkelola dalam tampilan `apg_plan_mgmt.dba_plans` secara keseluruhan.

Gunakan form `validate_plans(sql_hash, plan_hash, action)` untuk memvalidasi rencana terkelola yang ditentukan dengan `plan_hash`, untuk pernyataan terkelola yang ditentukan dengan `sql_hash`.

Gunakan form `validate_plans(sql_hash, NULL, action)` untuk memvalidasi semua rencana terkelola untuk laporan terkelola yang ditentukan dengan `sql_hash`.

Referensi untuk tampilan apg_plan_mgmt.dba_plans

Kolom informasi rencana dalam tampilan `apg_plan_mgmt.dba_plans` mencakup hal-hal berikut ini.

Kolom dba_plans	Deskripsi
`cardinality_error`	Pengukuran kesalahan antara kardinalitas yang diperkirakan dan kardinalitas yang sebenarnya. Kardinalitas adalah jumlah baris tabel yang akan diproses oleh rencana. Jika kesalahan pada kardinalitas berukuran besar, hal tersebut akan menambah kemungkinan bahwa rencana tersebut tidak optimal. Kolom ini diisi oleh fungsi apg_plan_mgmt.evolve_plan_baselines.
`compatibility_level`	Tingkat fitur pengoptimisasi Aurora PostgreSQL.
`created_by`	Pengguna terautentikasi (`session_user`) yang membuat rencana.
`enabled`	Indikator apakah rencana diaktifkan atau dinonaktifkan. Semua rencana diaktifkan secara default. Anda dapat menonaktifkan rencana untuk mencegah pengoptimisasi menggunakannya. Untuk mengubah nilai ini, gunakan fungsi apg_plan_mgmt.set_plan_enabled.
`environment_variables`	Parameter dan nilai Grand Unified Configuration (GUC) PostgreSQL yang telah diganti oleh pengoptimisasi pada saat rencana diambil.
`estimated_startup_cost`	Perkiraan biaya penyiapan pengoptimisasi sebelum pengoptimisasi menghasilkan baris tabel.
`estimated_total_cost`	Perkiraan biaya pengoptimisasi untuk mengirimkan baris tabel akhir.
`execution_time_benefit_ms`	Keuntungan waktu eksekusi dalam milidetik untuk mengaktifkan rencana. Kolom ini diisi oleh fungsi apg_plan_mgmt.evolve_plan_baselines.

Kolom dba_plans	Deskripsi
`execution_time_ms`	Perkiraan waktu dalam milidetik bahwa rencana akan berjalan. Kolom ini diisi oleh fungsi apg_plan_mgmt.evolve_plan_baselines.
`has_side_effects`	Nilai yang menunjukkan bahwa pernyataan SQL adalah pernyataan bahasa manipulasi data (DML) atau pernyataan SELECT yang berisi fungsi VOLATILE.
`last_used`	Nilai ini diperbarui ke tanggal saat ini setiap kali rencana dijalankan atau jika rencana adalah rencana berbiaya minimum dari pengoptimisasi kueri. Nilai ini disimpan di memori bersama dan dialirkan secara berkala ke disk. Untuk mendapatkan nilai terbaru, baca tanggal dari memori bersama dengan memanggil fungsi `apg_plan_mgmt.plan_last_used(sql_hash, plan_hash)` dan bukan membaca nilai `last_used` . Untuk informasi tambahan, lihat parameter apg_plan_mgmt.plan_retention_period.
`last_validated`	Tanggal dan waktu terbaru saat terverifikasi bahwa rencana dapat dibuat ulang dengan fungsi apg_plan_mgmt.validate_plans atau fungsi apg_plan_mgmt.evolve_plan_baselines.
`last_verified`	Tanggal dan waktu terbaru saat rencana terverifikasi sebagai rencana beperforma terbaik untuk parameter yang ditentukan oleh fungsi apg_plan_mgmt.evolve_plan_baselines.
`origin`	Cara pengambilan rencana dengan parameter apg_plan_mgmt.capture_plan_baselines. Nilai yang valid mencakup hal berikut: `M` – Rencana diambil dengan pengambilan rencana manual. `A` – Rencana diambil dengan pengambilan rencana manual.
`param_list`	Nilai parameter yang diteruskan ke pernyataan jika pernyataan ini merupakan pernyataan yang disiapkan.

Kolom dba_plans	Deskripsi
plan_created	Tanggal dan waktu rencana yang dibuat.
plan_hash	Pengidentifikasi rencana. Kombinasi dari plan_hash dan sql_hash mengidentifikasi rencana tertentu secara unik.
plan_outline	Representasi rencana yang digunakan untuk membuat ulang rencana eksekusi sebenarnya dan yang bersifat independen dari basis data. Operator di hierarki menunjukkan operator yang muncul di output EXPLAIN.
planning_time_ms	Waktu sebenarnya untuk menjalankan perencana, dalam milidetik. Kolom ini diisi oleh fungsi apg_plan_mgmt.evolve_plan_baselines.
queryId	Hash pernyataan, sebagaimana dihitung oleh ekstensi pg_stat_statements . Ini bukan pengidentifikasi yang stabil atau yang bersifat independen dari basis data karena bergantung pada pengidentifikasi objek (OID). Nilainya adalah 0 jika compute_query_id adalah off saat mengambil rencana kueri.
sql_hash	Nilai hash teks pernyataan SQL, yang dinormalisasikan dengan menghapus literal.
sql_text	Teks lengkap pernyataan SQL.

Kolom dba_plans	Deskripsi
`status`	Status rencana, yang menentukan cara pengoptimisasi menggunakan rencana. Nilai yang valid mencakup hal berikut: • `Approved` – Rencana yang dapat digunakan yang dapat dipilih pengoptimisasi untuk dijalankan. Pengoptimisasi menjalankan rencana berbiaya paling rendah dari kumpulan rencana yang disetujui (acuan dasar) untuk pernyataa n terkelola. Untuk mengatur ulang rencana ke disetujui, gunakan fungsi apg_plan_mgmt.evolve_plan_baselines. • `Unapproved` – Rencana yang diambil dan belum Anda verifikasi untuk digunakan. Untuk informasi selengkapnya, lihat Mengevaluasi performa rencana. • `Rejected` – Rencana yang tidak akan digunakan oleh pengoptimal. Untuk informasi selengkapnya, lihat Menolak atau menonaktifkan rencana yang lebih lambat. • `Preferred` – Rencana yang telah Anda tentukan sebagai rencana yang lebih disukai untuk digunakan untuk pernyataan terkelola. Jika rencana berbiaya minimum dari pengoptimisasi bukanlah rencana yang disetujui atau yang lebih disukai, Anda dapat mengurangi overhead penerapan rencana. Untuk melakukan nya, buat subkumpulan dari rencana yang disetujui, yaitu `Preferred` . Ketika rencana berbiaya minimum dari pengoptimisasi bukanlah rencana `Approved`, rencana `Preferred` akan dipilih sebelum rencana `Approved`. Untuk mengatur ulang rencana ke `Preferred` , gunakan fungsi apg_plan_mgmt.set_plan_status.
`stmt_name`	Nama pernyataan SQL di dalam pernyataan PREPARE. Nilai ini adalah string kosong untuk pernyataan yang disiapkan tanpa nama. Nilai ini adalah NULL untuk pernyataan yang tidak disiapkan.

Kolom dba_plans	Deskripsi
`total_time_benefit_ms`	Keuntungan total waktu dalam milidetik untuk mengaktifkan rencana ini. Nilai ini mempertimbangkan waktu perencanaan dan waktu eksekusi. Jika nilai ini negatif, ada kerugian untuk mengaktifkan rencana ini. Kolom ini diisi oleh fungsi apg_plan_mgmt.evolve_plan_baselines.

Fitur lanjutan dalam Manajemen Rencana Kueri

Di bagian berikut ini, Anda dapat menemukan informasi tentang fitur lanjutan Manajemen Rencana Kueri (QPM) Aurora PostgreSQL:

Topik

- Mengambil rencana eksekusi Aurora PostgreSQL di Replika
- Mendukung partisi tabel

Mengambil rencana eksekusi Aurora PostgreSQL di Replika

QPM (Query Plan Management) memungkinkan Anda mengambil rencana kueri yang dihasilkan oleh Replika Aurora dan menyimpannya di instans DB primer dari klaster Aurora DB. Anda dapat mengumpulkan rencana kueri dari semua Replika Aurora, dan memelihara serangkaian rencana optimal dalam tabel persisten terpusat pada instans primer. Anda kemudian dapat menerapkan rencana ini pada Replika lain jika diperlukan. Ini membantu Anda menjaga stabilitas rencana eksekusi dan meningkatkan performa kueri di seluruh klaster DB dan versi mesin.

Topik

- Prasyarat
- Mengelola pengambilan rencana untuk Replika Aurora
- Pemecahan Masalah

Prasyarat

Aktifkan **`capture_plan_baselines parameter`** di Replika Aurora - Atur parameter `capture_plan_baselines` ke otomatis atau manual untuk mengambil rencana di Replika Aurora. Untuk informasi selengkapnya, lihat apg_plan_mgmt.capture_plan_baselines.

Instal ekstensi postgres_fdw - Anda harus menginstal ekstensi wrapper data asing `postgres_fdw` untuk mengambil rencana di Replika Aurora. Jalankan perintah berikut di setiap basis data untuk menginstal ekstensi ini.

```
postgres=> CREATE EXTENSION IF NOT EXISTS postgres_fdw;
```

Mengelola pengambilan rencana untuk Replika Aurora

Aktifkan pengambilan rencana untuk Replika Aurora

Anda harus memiliki hak akses `rds_superuser` untuk membuat atau menghapus Pengambilan Rencana di Replika Aurora. Untuk informasi selengkapnya tentang peran dan izin pengguna, lihat Memahami peran dan izin PostgreSQL.

Untuk mengambil rencana, panggil fungsi apg_plan_mgmt.create_replica_plan_capture dalam instans DB penulis, seperti yang ditunjukkan berikut ini:

```
postgres=> CALL
 apg_plan_mgmt.create_replica_plan_capture('cluster_endpoint', 'password');
```

- cluster_endpoint - cluster_endpoint (titik akhir penulis) menyediakan dukungan failover untuk Pengambilan Rencana di Replika Aurora.
- password - Kami menyarankan Anda untuk mengikuti panduan di bawah ini saat membuat kata sandi untuk meningkatkan keamanan:
  - Kata sandi harus berisi setidaknya 8 karakter.
  - Kata sandi harus berisi setidaknya satu huruf besar, satu huruf kecil, dan satu angka.
  - Kata sandi harus memiliki setidaknya satu karakter khusus (?, !, #, <, >, *, dan sebagainya).

> **Note**
>
> Jika Anda mengubah titik akhir, kata sandi, atau nomor port klaster, Anda harus menjalankan `apg_plan_mgmt.create_replica_plan_capture()` lagi dengan titik akhir dan kata

sandi klaster untuk menginisialisasi ulang pengambilan rencana. Jika tidak, pengambilan rencana dari Replika Aurora akan gagal.

Nonaktifkan pengambilan rencana untuk Replika Aurora

Anda dapat menonaktifkan parameter `capture_plan_baselines` di Replika Aurora dengan menetapkan nilainya ke `off` dalam grup Parameter.

Hapus pengambilan rencana untuk Replika Aurora

Anda dapat sepenuhnya menghapus Pengambilan Rencana di Replika Aurora tetapi pastikan untuk menonaktifkannya terlebih dahulu. Untuk menghapus pengambilan rencana, panggil `apg_plan_mgmt.remove_replica_plan_capture` seperti yang ditunjukkan berikut:

```
postgres=> CALL apg_plan_mgmt.remove_replica_plan_capture();
```

Anda harus memanggil apg_plan_mgmt.create_replica_plan_capture() lagi untuk mengaktifkan pengambilan rencana di Replika Aurora dengan titik akhir dan kata sandi klaster.

Pemecahan Masalah

Di bagian berikut ini, Anda dapat menemukan ide dan solusi pemecahan masalah jika rencana tidak diambil di Replika Aurora seperti yang diharapkan.

- Pengaturan parameter - Periksa apakah parameter `capture_plan_baselines` diatur ke nilai yang tepat untuk mengaktifkan pengambilan rencana.
- Ekstensi **`postgres_fdw`** diinstal - Gunakan kueri berikut untuk memeriksa apakah `postgres_fdw` diinstal.

  ```
  postgres=> SELECT * FROM pg_extension WHERE extname = 'postgres_fdw'
  ```

- create_replica_plan_capture() dipanggil - Gunakan perintah berikut untuk memeriksa apakah pemetaan pengguna sudah ada. Jika tidak, panggil `create_replica_plan_capture()` untuk menginisialisasi fitur ini.

  ```
  postgres=> SELECT * FROM pg_foreign_server WHERE srvname =
   'apg_plan_mgmt_writer_foreign_server';
  ```

- Titik akhir dan nomor port klaster - Periksa apakah titik akhir dan nomor port klaster sudah sesuai. Tidak akan ada pesan kesalahan yang ditampilkan jika nilai-nilai ini salah.

  Gunakan perintah berikut untuk memverifikasi titik akhir yang digunakan dalam create() dan untuk memeriksa di basis data mana titik akhir tersebut berada:

  ```
  postgres=> SELECT srvoptions FROM pg_foreign_server WHERE srvname =
   'apg_plan_mgmt_writer_foreign_server';
  ```

- reload() - Anda harus memanggil apg_plan_mgmt.reload() setelah memanggil apg_plan_mgmt.delete_plan() di Replika Aurora untuk membuat fungsi penghapusan efektif. Hal ini akan memastikan bahwa perubahan telah berhasil diterapkan.
- Kata sandi - Anda harus memasukkan kata sandi di create_replica_plan_capture() sesuai pedoman yang disebutkan. Jika tidak, Anda akan menerima pesan kesalahan. Untuk informasi selengkapnya, lihat Mengelola pengambilan rencana untuk Replika Aurora. Gunakan kata sandi lain yang sesuai dengan persyaratan.
- Koneksi Lintas Wilayah - Pengambilan rencana di Replika Aurora juga didukung di basis data global Aurora, yang memungkinkan instans penulis dan Replika Aurora berada di wilayah yang berbeda. Instans penulis dan Replika lintas Wilayah harus dapat berkomunikasi menggunakan Peering VPC. Untuk informasi selengkapnya, lihat Peering VPC. Jika failover lintas Wilayah terjadi, Anda harus mengonfigurasi ulang titik akhir ke titik akhir klaster DB primer yang baru.

Mendukung partisi tabel

Manajemen Rencana Kueri (QPM) Aurora PostgreSQL mendukung partisi tabel dalam versi berikut:

- 15.3 dan versi 15 yang lebih tinggi
- 14.8 dan versi 14 yang lebih tinggi
- 13.11 dan versi 13 yang lebih tinggi

Untuk informasi selengkapnya, lihat Pemartisian Tabel.

Topik

- Mengatur partisi tabel
- Mengambil rencana untuk partisi tabel
- Memberlakukan rencana partisi tabel

- Konvensi Penamaan

Mengatur partisi tabel

Untuk mengatur partisi tabel di QPM Aurora PostgreSQL, lakukan hal berikut:

1. Atur `apg_plan_mgmt.plan_hash_version` ke 3 atau lebih di grup parameter klaster DB.
2. Temukan basis data yang menggunakan Manajemen Rencana Kueri dan memiliki entri dalam tampilan `apg_plan_mgmt.dba_plans`.
3. Panggil `apg_plan_mgmt.validate_plans('update_plan_hash')` untuk memperbarui nilai `plan_hash` dalam tabel rencana.
4. Ulangi langkah 2-3 untuk semua basis data dengan Manajemen Rencana Kueri diaktifkan yang memiliki entri dalam tampilan `apg_plan_mgmt.dba_plans`.

Untuk informasi selengkapnya tentang parameter ini, lihat Referensi parameter untuk manajemen rencana kueri Aurora PostgreSQL.

Mengambil rencana untuk partisi tabel

Di QPM, berbagai rencana dibedakan berdasarkan nilai `plan_hash`-nya. Untuk memahami perubahan `plan_hash`, Anda harus terlebih dahulu memahami jenis rencana yang serupa.

Kombinasi metode akses, nama indeks tanpa digit, dan nama partisi tanpa digit, yang terakumulasi pada tingkat simpul Append harus konstan agar rencana dianggap sama. Partisi spesifik yang diakses dalam rencana tidaklah signifikan. Dalam contoh berikut, tabel `tbl_a` dibuat dengan 4 partisi.

```
postgres=>create table tbl_a(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table tbl_a1 partition of tbl_a for values from (0) to (1000);
CREATE TABLE
postgres=>create table tbl_a2 partition of tbl_a for values from (1001) to (2000);
CREATE TABLE
postgres=>create table tbl_a3 partition of tbl_a for values from (2001) to (3000);
CREATE TABLE
postgres=>create table tbl_a4 partition of tbl_a for values from (3001) to (4000);
CREATE TABLE
postgres=>create index t_i on tbl_a using btree (i);
CREATE INDEX
postgres=>create index t_j on tbl_a using btree (j);
```

```
CREATE INDEX
postgres=>create index t_k on tbl_a using btree (k);
CREATE INDEX
```

Rencana berikut dianggap sama karena metode pemindaian tunggal digunakan untuk memindai `tbl_a` terlepas dari jumlah partisi yang dicari kueri.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 999 and j < 9910 and k > 50;

                        QUERY PLAN
-----------------------------------------------------------------------
Seq Scan on tbl_a1 tbl_a
    Filter: ((i >= 990) AND (i <= 999) AND (j < 9910) AND (k > 50))
SQL Hash: 1553185667, Plan Hash: -694232056
(3 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                        QUERY PLAN
-----------------------------------------------------------------------
Append
    ->  Seq Scan on tbl_a1 tbl_a_1
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    ->  Seq Scan on tbl_a2 tbl_a_2
            Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
    SQL Hash: 1553185667, Plan Hash: -694232056
    (6 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
```

```
 SQL Hash: 1553185667, Plan Hash: -694232056
(8 rows)
```

Tiga rencana berikut juga dianggap sama karena pada tingkat induk, metode akses, nama indeks tanpa digit, dan nama partisi tanpa digit adalah `SeqScan tbl_a`, `IndexScan (i_idx) tbl_a`.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_i_idx on tbl_a2 tbl_a_2
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(7 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
```

```
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a3 tbl_a_3
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(11 rows)
```

Terlepas dari urutan dan jumlah kemunculan yang berbeda dalam partisi turunan, metode akses, nama indeks tanpa digit, dan nama partisi tanpa digit akan tetap konstan pada tingkat induk untuk masing-masing rencana di atas.

Namun, rencana akan dianggap berbeda jika salah satu dari persyaratan berikut terpenuhi:

- Setiap metode akses tambahan digunakan dalam rencana.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a3_i_idx
               Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 1134525070
(11 rows)
```

- Salah satu metode akses dalam rencana tidak digunakan lagi.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;
```

```
                                QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -694232056
(6 rows)
```

- Indeks yang terkait dengan metode indeks diubah.

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between
 990 and 1100 and j < 9910 and k > 50;

                             QUERY PLAN
---------------------------------------------------------------------------
 Append
   ->  Seq Scan on tbl_a1 tbl_a_1
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a2_j_idx on tbl_a2 tbl_a_2
         Index Cond: (j < 9910)
         Filter: ((i >= 990) AND (i <= 1100) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993343726
(7 rows)
```

Memberlakukan rencana partisi tabel

Rencana yang disetujui untuk tabel yang dipartisi diberlakukan dengan kesesuaian posisi. Rencana tidak bersifat spesifik untuk partisi, dan dapat diberlakukan pada partisi selain rencana yang direferensikan dalam kueri asli. Rencana juga memiliki kemampuan untuk diberlakukan untuk kueri yang mengakses jumlah partisi yang berbeda dari garis besar asli yang disetujui.

Misalnya, jika garis besar yang disetujui adalah untuk rencana berikut:

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
---------------------------------------------------------------------------
 Append
```

```
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
 SQL Hash: 1553185667, Plan Hash: -993736942
(10 rows)
```

Dengan demikian, rencana ini juga dapat diterapkan pada kueri SQL yang mereferensikan 2, 4, atau lebih partisi. Rencana yang mungkin muncul dari skenario ini untuk akses partisi 2 dan 4 adalah:

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(8 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
```

```
   ->  Seq Scan on tbl_a4 tbl_a_4
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(12 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 3100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 3100) AND (j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a3_i_idx on tbl_a3 tbl_a_3
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Index Scan using tbl_a4_i_idx on tbl_a4 tbl_a_4
         Index Cond: ((i >= 990) AND (i <= 3100))
         Filter: ((j < 9910) AND (k > 50))
 Note: An Approved plan was used instead of the minimum cost plan.
 SQL Hash: 1553185667, Plan Hash: -993736942, Minimum Cost Plan Hash: -1873216041
(14 rows)
```

Pertimbangkan rencana lain yang disetujui dengan metode akses berbeda untuk setiap partisi:

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 2100 and j < 9910 and k > 50;

                                QUERY PLAN
-------------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 2100) AND (j < 9910) AND (k > 50))
   ->  Bitmap Heap Scan on tbl_a3 tbl_a_3
         Recheck Cond: ((i >= 990) AND (i <= 2100))
         Filter: ((j < 9910) AND (k > 50))
```

```
         ->  Bitmap Index Scan on tbl_a3_i_idx
               Index Cond: ((i >= 990) AND (i <= 2100))
 SQL Hash: 1553185667, Plan Hash: 2032136998
(12 rows)
```

Dalam hal ini, rencana apa pun yang dibaca dari dua partisi akan gagal diberlakukan. Kecuali jika semua kombinasi (metode akses, nama indeks) dari rencana yang disetujui dapat digunakan, rencana tidak dapat diberlakukan. Misalnya, rencana berikut memiliki hash rencana yang berbeda dan rencana yang disetujui tidak dapat diberlakukan dalam kasus ini:

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;

                              QUERY PLAN
-----------------------------------------------------------------------
 Append
   ->  Bitmap Heap Scan on tbl_a1 tbl_a_1
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a1_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
   ->  Bitmap Heap Scan on tbl_a2 tbl_a_2
         Recheck Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
         ->  Bitmap Index Scan on tbl_a2_i_idx
               Index Cond: ((i >= 990) AND (i <= 1900))
  Note: This is not an Approved plan.  No usable Approved plan was found.
  SQL Hash: 1553185667, Plan Hash: -568647260
(13 rows)
```

```
postgres=>explain (hashes true, costs false) select j, k from tbl_a where i between 990
 and 1900 and j < 9910 and k > 50;

                              QUERY PLAN
-----------------------------------------------------------------------
 Append
   ->  Index Scan using tbl_a1_i_idx on tbl_a1 tbl_a_1
         Index Cond: ((i >= 990) AND (i <= 1900))
         Filter: ((j < 9910) AND (k > 50))
   ->  Seq Scan on tbl_a2 tbl_a_2
         Filter: ((i >= 990) AND (i <= 1900) AND (j < 9910) AND (k > 50))
 Note: This is not an Approved plan.  No usable Approved plan was found.
```

```
 SQL Hash: 1553185667, Plan Hash: -496793743
(8 rows)
```

Konvensi Penamaan

Untuk pemberlakuan rencana partisi tabel di QPM, tabel induk harus sesuai dengan aturan penamaan berikut:

- Nama tabel induk harus berbeda menurut huruf atau karakter khusus, dan bukan hanya angka. Misalnya, tA, tB, dan tC adalah nama yang dapat diterima untuk tabel induk terpisah, sedangkan t1, t2, dan t3 tidak.
- Partisi dari induk yang sama harus berbeda satu sama lain menurut angka saja. Misalnya, nama partisi tA yang dapat diterima bisa berupa tA1, tA2, atau t1A, t2A, atau bahkan dengan beberapa angka.

Perbedaan lain (huruf, karakter khusus) tidak akan menjamin pemberlakuan rencana. Tabel yang diwarisi harus mengikuti konvensi penamaan yang sama dengan tabel yang dipartisi.

Indeks mengikuti konvensi penamaan yang sama seperti tabel induk.

- Nama indeks pada tabel induk harus berbeda menurut huruf atau karakter khusus, dan bukan hanya angka. Misalnya, jika tabel induk tA memiliki indeks i_idx, maka tabel induk tB seharusnya tidak memiliki indeks seperti `i_idx2` atau `i2_idx`. Namun, nama-nama seperti `i_idx_2` (garis bawah pembeda) atau `icol_idx` (huruf pembeda `col`) akan sesuai dengan konvensi penamaan.

Jika konvensi penamaan di atas tidak dipatuhi, pemberlakuan rencana yang disetujui dapat gagal. Contoh berikut mengilustrasikan pemberlakuan yang gagal:

```
postgres=>create table t1(i int, j int, k int, l int, m int) partition by range(i);
CREATE TABLE
postgres=>create table t1a partition of t1 for values from (0) to (1000);
CREATE TABLE
postgres=>create table t1b partition of t1 for values from (1001) to (2000);
CREATE TABLE
postgres=>SET apg_plan_mgmt.capture_plan_baselines TO 'manual';
SET
postgres=>explain (hashes true, costs false) select count(*) from t1 where i > 0;

                            QUERY PLAN
```

```
------------------------------------------------------------------------------
 Aggregate
   ->  Append
         ->  Seq Scan on t1a t1_1
               Filter: (i > 0)
         ->  Seq Scan on t1b t1_2
               Filter: (i > 0)
 SQL Hash: -1720232281, Plan Hash: -1010664377
(7 rows)
```

```
postgres=>SET apg_plan_mgmt.use_plan_baselines TO 'on';
SET
            postgres=>explain (hashes true, costs false) select count(*) from t1 where
 i > 1000;

                            QUERY PLAN
------------------------------------------------------------------------------
 Aggregate
   ->  Seq Scan on t1b t1
         Filter: (i > 1000)
 Note: This is not an Approved plan. No usable Approved plan was found.
 SQL Hash: -1720232281, Plan Hash: 335531806
(5 rows)
```

Meskipun dua rencana di atas seharusnya dianggap sama, rencana tersebut berbeda karena nama tabel turunannya tidak sama setelah angkanya dihapus, sehingga tidak mengikuti aturan konvensi penamaan.

Menangani ekstensi dan pembungkus data asing

Untuk memperluas fungsionalitas ke klaster Aurora PostgreSQL-Compatible Edition, Anda dapat menginstal dan menggunakan berbagai ekstensi PostgreSQL. Misalnya, jika kasus penggunaan Anda memerlukan entri data intensif di seluruh tabel yang sangat besar, Anda dapat menginstal ekstensi `pg_partman` untuk mempartisi data dan dengan demikian menyebarkan beban kerja.

> **Note**
>
> Mulai versi Aurora PostgreSQL 14.5, Aurora PostgreSQL mendukung Ekstensi Bahasa Tepercaya untuk PostgreSQL. Fitur ini diimplementasikan sebagai ekstensi `pg_tle`, yang dapat Anda tambahkan ke Aurora PostgreSQL Anda. Dengan ekstensi ini, pengembang dapat membuat ekstensi PostgreSQL mereka sendiri di lingkungan yang aman yang menyederhanakan persyaratan penyiapan dan konfigurasi, serta sebagian besar pengujian awal untuk ekstensi baru. Untuk informasi selengkapnya, lihat Bekerja dengan Ekstensi Bahasa Tepercaya untuk PostgreSQL.

Dalam beberapa kasus, daripada menginstal ekstensi, Anda dapat menambahkan modul tertentu ke daftar `shared_preload_libraries` dalam grup parameter klaster DB khusus klaster DB Aurora PostgreSQL. Biasanya, grup parameter klaster DB default hanya memuat `pg_stat_statements`, tetapi beberapa modul lain tersedia untuk ditambahkan ke daftar. Misalnya, Anda dapat menambahkan kemampuan penjadwalan dengan menambahkan modul `pg_cron`, seperti yang dijelaskan dalam Menjadwalkan pemeliharaan dengan ekstensi pg_cron PostgreSQL. Sebagai contoh lain, Anda dapat men-log rencana eksekusi kueri dengan memuat modul `auto_explain`. Untuk mempelajari selengkapnya, lihat Mencatat log rencana eksekusi kueri di pusat pengetahuan AWS.

Ekstensi yang memberikan akses ke data eksternal lebih khusus dikenal sebagai pembungkus data asing (FDW). Sebagai contoh, ekstensi `oracle_fdw` memungkinkan klaster DB Aurora PostgreSQL bekerja dengan basis data Oracle.

Anda juga dapat menentukan dengan tepat ekstensi yang dapat diinstal pada instans DB Aurora PostgreSQL, dengan mencantumkannya dalam parameter `rds.allowed_extensions`. Untuk informasi selengkapnya, lihat Membatasi instalasi ekstensi PostgreSQL.

Berikut ini, Anda dapat menemukan informasi tentang pengaturan dan penggunaan beberapa ekstensi, modul, dan FDW yang tersedia untuk Aurora PostgreSQL. Untuk menyederhanakan,

ini semua akan disebut sebagai "ekstensi". Anda dapat menemukan daftar ekstensi yang dapat digunakan dengan versi Aurora PostgreSQL yang tersedia saat ini, lihat Versi ekstensi untuk Amazon Aurora PostgreSQL dalam Catatan Rilis untuk Aurora PostgreSQL.

- Mengelola objek besar dengan modul lo
- Mengelola data spasial dengan ekstensi PostGIS
- Mengelola partisi PostgreSQL dengan ekstensi pg_partman
- Menjadwalkan pemeliharaan dengan ekstensi pg_cron PostgreSQL
- Menggunakan pgAudit untuk membuat log aktivitas basis data
- Menggunakan pglogical untuk menyinkronkan data di seluruh instans
- Bekerja dengan basis data Oracle menggunakan ekstensi oracle_fdw
- Bekerja dengan basis data SQL Server menggunakan ekstensi tds_fdw

Menggunakan dukungan ekstensi yang didelegasikan Amazon Aurora untuk PostgreSQL

Menggunakan dukungan ekstensi yang didelegasikan Amazon Aurora untuk PostgreSQL, Anda dapat mendelegasikan manajemen ekstensi kepada pengguna yang tidak perlu menjadi file. `rds_superuser` Dengan dukungan ekstensi yang didelegasikan ini, peran baru yang `rds_extension` disebut dibuat dan Anda harus menetapkan ini kepada pengguna untuk mengelola ekstensi lainnya. Peran ini dapat membuat, memperbarui, dan menjatuhkan ekstensi.

Anda dapat menentukan ekstensi yang dapat diinstal pada instance Aurora PostgreSQL DB Anda, dengan mencantumkannya di parameter. `rds.allowed_extensions` Untuk informasi selengkapnya, lihat Menggunakan ekstensi PostgreSQL dengan Amazon RDS for PostgreSQL.

Anda dapat membatasi daftar ekstensi yang tersedia yang dapat dikelola oleh pengguna dengan `rds_extension` peran menggunakan `rds.allowed_delegated_extensions` parameter.

Dukungan ekstensi yang didelegasikan tersedia dalam versi berikut:

- Semua versi yang lebih tinggi
- 15.5 dan versi 15 yang lebih tinggi
- 14.10 dan versi 14 yang lebih tinggi
- 13.13 dan 13 versi yang lebih tinggi
- 12.17 dan versi 12 yang lebih tinggi

Topik

- Mengaktifkan dukungan ekstensi delegasi ke pengguna
- Konfigurasi yang digunakan dalam dukungan ekstensi yang didelegasikan Aurora untuk PostgreSQL
- Mematikan dukungan untuk ekstensi yang didelegasikan
- Manfaat menggunakan dukungan ekstensi yang didelegasikan Amazon Aurora
- Batasan dukungan ekstensi yang didelegasikan Aurora untuk PostgreSQL
- Izin diperlukan untuk ekstensi tertentu
- Pertimbangan Keamanan
- Jatuhkan kaskade ekstensi dinonaktifkan
- Contoh ekstensi yang dapat ditambahkan menggunakan dukungan ekstensi yang didelegasikan

Mengaktifkan dukungan ekstensi delegasi ke pengguna

Anda harus melakukan hal berikut untuk mengaktifkan dukungan ekstensi delegasi kepada pengguna:

1. Berikan **`rds_extension`** peran kepada pengguna - Connect ke database sebagai `rds_superuser` dan jalankan perintah berikut:

   ```
   Postgres => grant rds_extension to user_name;
   ```

2. Mengatur daftar ekstensi yang tersedia bagi pengguna yang didelegasikan untuk dikelola — `rds.allowed_delegated_extensions` Memungkinkan Anda menentukan subset ekstensi yang tersedia menggunakan parameter `rds.allowed_extensions` cluster DB. Anda dapat melakukan ini di salah satu level berikut:

   - Di cluster atau grup parameter instance, melalui AWS Management Console atau API. Untuk informasi selengkapnya, lihat Bekerja dengan grup parameter.
   - Gunakan perintah berikut di tingkat database:

     ```
     alter database database_name set rds.allowed_delegated_extensions =
      'extension_name_1,
                        extension_name_2,...extension_name_n';
     ```

   - Gunakan perintah berikut di tingkat pengguna:

```
alter user user_name set rds.allowed_delegated_extensions = 'extension_name_1,
                    extension_name_2,...extension_name_n';
```

> **Note**
>
> Anda tidak perlu me-restart database setelah mengubah parameter `rds.allowed_delegated_extensions` dinamis.

3. Izinkan akses ke pengguna yang didelegasikan ke objek yang dibuat selama proses pembuatan ekstensi - Ekstensi tertentu membuat objek yang memerlukan izin tambahan untuk diberikan sebelum pengguna dengan `rds_extension` peran dapat mengaksesnya. `rds_superuser`Harus memberikan akses pengguna yang didelegasikan ke objek tersebut. Salah satu opsi adalah menggunakan pemicu peristiwa untuk secara otomatis memberikan izin kepada pengguna yang didelegasikan. Untuk informasi lebih lanjut, lihat contoh pemicu peristiwa diMematikan dukungan untuk ekstensi yang didelegasikan.

Konfigurasi yang digunakan dalam dukungan ekstensi yang didelegasikan Aurora untuk PostgreSQL

Nama Konfigsi	Deskripsi	nilai default	Catatan	Siapa yang dapat memodifikasi atau memberikan izin
`rds.aed_dted_esions`	Parameter ini membatasi ekstensi yang dapat dikelola peran rds_extension dalam database. Itu harus merupakan bagian dari rds.allowed_extensions.	string kosong	• Secara default, parameter ini adalah string kosong, yang berarti bahwa tidak ada ekstensi yang telah didelegasikan kepada pengguna dengan`rds_extension` .	rds_superuser

Nama Konfigsi	Deskripsi	nilai default	Catatan	Siapa yang dapat memodifikasi atau memberikan izin
			• Setiap ekstensi yang didukung dapat ditambahkan jika pengguna memiliki izin untuk melakukannya. Untuk melakukan ini, atur `rds.allowed_delegated_extensions` parameter ke string nama ekstensi yang dipisahkan koma. Dengan menambahkan daftar ekstensi ke parameter ini, Anda secara eksplisit mengidentifikasi ekstensi yang dapat dipasang oleh pengguna dengan `rds_extension` peran tersebut. • Ketika diatur ke`*`, itu berarti bahwa semua ekstensi	

Nama Konfigsi	Deskripsi	nilai default	Catatan	Siapa yang dapat memodifikasi atau memberikan izin
			yang terdaftar di `rds_allowed_extensions` didelegasikan kepada pengguna dengan `rds_extension` peran. Untuk mempelajari lebih lanjut tentang pengaturan parameter ini, lihatMengaktifkan dukungan ekstensi delegasi kepengguna.	

Nama Konfigurasi	Deskripsi	nilai default	Catatan	Siapa yang dapat memodifikasi atau memberikan izin
`rds.allowed_extensions`	Parameter ini memungkinkan pelanggan membatasi ekstensi yang dapat diinstal di instance Aurora PostgreSQL DB. Untuk informasi selengkapnya, lihat Membatasi pemasangan ekstensi PostgreSQL	"*"	Secara default, parameter ini diatur ke “*”, yang berarti bahwa semua ekstensi yang didukung pada RDS untuk PostgreSQL dan Aurora PostgreSQL diizinkan untuk dibuat oleh pengguna dengan hak istimewa yang diperlukan. Kosong berarti tidak ada ekstensi yang dapat diinstal di instance Aurora PostgreSQL DB.	administrator

Nama Konfigurasi	Deskripsi	nilai default	Catatan	Siapa yang dapat memodifikasi atau memberikan izin
`rds-delegated_nsionow_drascac`	Parameter ini mengontrol kemampuan pengguna `rds_extension` untuk menjatuhkan ekstensi menggunakan opsi kaskade.	off	Secara default, `rds-delegated_extensionallow_drop_cascade` diatur ke `off`. Ini berarti bahwa pengguna dengan tidak `rds_extension` diizinkan untuk menjatuhkan ekstensi menggunakan opsi kaskade. Untuk memberikan kemampuan itu, `rds.delegated_extensionallow_drop_cascade` parameter harus diatur keon.	rds_superuser

Mematikan dukungan untuk ekstensi yang didelegasikan

Mematikan sebagian

Pengguna yang didelegasikan tidak dapat membuat ekstensi baru tetapi masih dapat memperbarui ekstensi yang ada.

- Atur ulang `rds.allowed_delegated_extensions` ke nilai default di grup parameter cluster DB.
- Gunakan perintah berikut di tingkat database:

```
alter database database_name reset rds.allowed_delegated_extensions;
```

- Gunakan perintah berikut di tingkat pengguna:

```
alter user user_name reset rds.allowed_delegated_extensions;
```

Mematikan sepenuhnya

Mencabut `rds_extension` peran dari pengguna akan mengembalikan pengguna ke izin standar. Pengguna tidak dapat lagi membuat, memperbarui, atau menjatuhkan ekstensi.

```
postgres => revoke rds_extension from user_name;
```

Contoh pemicu peristiwa

Jika Anda ingin mengizinkan pengguna yang didelegasikan `rds_extension` untuk menggunakan ekstensi yang memerlukan izin pengaturan pada objek mereka yang dibuat oleh pembuatan ekstensi, Anda dapat menyesuaikan contoh pemicu peristiwa di bawah ini dan hanya menambahkan ekstensi yang Anda inginkan agar pengguna yang didelegasikan memiliki akses ke fungsionalitas penuh. Pemicu peristiwa ini dapat dibuat pada template1 (template default), oleh karena itu semua database yang dibuat dari template1 akan memiliki pemicu peristiwa itu. Ketika pengguna yang didelegasikan menginstal ekstensi, pemicu ini akan secara otomatis memberikan kepemilikan pada objek yang dibuat oleh ekstensi.

```
CREATE OR REPLACE FUNCTION create_ext()

  RETURNS event_trigger AS $$

DECLARE

  schemaname TEXT;
  databaseowner TEXT;

  r RECORD;
```

```
BEGIN

  IF tg_tag = 'CREATE EXTENSION' and current_user != 'rds_superuser' THEN
    RAISE NOTICE 'SECURITY INVOKER';
    RAISE NOTICE 'user: %', current_user;
    FOR r IN SELECT * FROM pg_event_trigger_ddl_commands()
    LOOP
        CONTINUE WHEN r.command_tag != 'CREATE EXTENSION' OR r.object_type !=
 'extension';

        schemaname = (
            SELECT n.nspname
            FROM pg_catalog.pg_extension AS e
            INNER JOIN pg_catalog.pg_namespace AS n
            ON e.extnamespace = n.oid
            WHERE e.oid = r.objid
        );

        databaseowner = (
            SELECT pg_catalog.pg_get_userbyid(d.datdba)
            FROM pg_catalog.pg_database d
            WHERE d.datname = current_database()
        );
        RAISE NOTICE 'Record for event trigger %, objid: %,tag: %, current_user: %,
 schema: %, database_owenr: %', r.object_identity, r.objid, tg_tag, current_user,
 schemaname, databaseowner;
        IF r.object_identity = 'address_standardizer_data_us' THEN
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_gaz TO
 %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_lex TO
 %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE %I.us_rules
 TO %I WITH GRANT OPTION;', schemaname, databaseowner);
        ELSIF r.object_identity = 'dict_int' THEN
            EXECUTE format('ALTER TEXT SEARCH DICTIONARY %I.intdict OWNER TO %I;',
 schemaname, databaseowner);
        ELSIF r.object_identity = 'pg_partman' THEN
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE
 %I.part_config TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE
 %I.part_config_sub TO %I WITH GRANT OPTION;', schemaname, databaseowner);
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE
 %I.custom_time_partitions TO %I WITH GRANT OPTION;', schemaname, databaseowner);
```

```
        ELSIF r.object_identity = 'postgis_topology' THEN
            EXECUTE format('GRANT SELECT, UPDATE, INSERT, DELETE ON ALL TABLES IN
 SCHEMA topology TO %I WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA topology TO
 %I WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA topology TO %I
 WITH GRANT OPTION;', databaseowner);
            EXECUTE format('GRANT USAGE ON SCHEMA topology TO %I WITH GRANT OPTION;',
 databaseowner);
        END IF;
    END LOOP;
  END IF;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE EVENT TRIGGER log_create_ext ON ddl_command_end EXECUTE PROCEDURE create_ext();
```

Manfaat menggunakan dukungan ekstensi yang didelegasikan Amazon Aurora

Dengan menggunakan dukungan ekstensi yang didelegasikan Amazon Aurora untuk PostgreSQL, Anda mendelegasikan manajemen ekstensi dengan aman kepada pengguna yang tidak memiliki peran tersebut. `rds_superuser` Fitur ini memberikan manfaat sebagai berikut:

- Anda dapat dengan mudah mendelegasikan manajemen ekstensi kepada pengguna pilihan Anda.
- Ini tidak membutuhkan `rds_superuser` peran.
- Menyediakan kemampuan untuk mendukung kumpulan ekstensi yang berbeda untuk database yang berbeda di cluster DB yang sama.

Batasan dukungan ekstensi yang didelegasikan Aurora untuk PostgreSQL

- Objek yang dibuat selama proses pembuatan ekstensi mungkin memerlukan hak istimewa tambahan agar ekstensi berfungsi dengan baik.

Izin diperlukan untuk ekstensi tertentu

Untuk membuat, menggunakan, atau memperbarui ekstensi berikut, pengguna yang didelegasikan harus memiliki hak istimewa yang diperlukan pada fungsi, tabel, dan skema berikut.

Ekste yang mem an kepe an atau izin	Fungsi	Tabel	Skema	Kamus Pencarian Teks	Komentar
addre tanda er_da s		us_gaz, us_lex, us_lex, i.us_rule s			
amch	bt_index_check, bt_index_ parent_check				
dict_i				intdict	
pg_p n		custom_ti me_partisi, part_config, part_config_sub			
pg_st tatem					
Post(	st_tileamplop	spatial_ref_sys			
postg aster					
postg opolo		topologi, lapisan	topologi		pengguna yang didelegas ikan Harus menjadi pemilik database

Ekste yang mem an kepe an atau izin	Fungsi	Tabel	Skema	Kamus Pencarian Teks	Komentar
log_f	create_fo reign_tab le_for_log_file				
rds_t	role_pass word_encr yption_type				
postg iger_ oder		geocode_s ettings_default, geocode_s ettings	harimau		
pg_fr acem	pg_freespace				
pg_vi lity	pg_visibility				

Pertimbangan Keamanan

Perlu diingat bahwa pengguna dengan `rds_extension` peran akan dapat mengelola ekstensi di semua database tempat mereka memiliki hak istimewa terhubung. Jika tujuannya adalah agar pengguna yang didelegasikan mengelola ekstensi pada satu database, praktik yang baik adalah mencabut semua hak istimewa dari publik di setiap database, kemudian secara eksplisit memberikan hak istimewa koneksi untuk database spesifik tersebut kepada pengguna delegasi.

Ada beberapa ekstensi yang dapat memungkinkan pengguna untuk mengakses informasi dari beberapa database. Pastikan pengguna yang Anda berikan `rds_extension` memiliki kemampuan

lintas basis data sebelum menambahkan ekstensi ini`rds.allowed_delegated_extensions`. Misalnya, `postgres_fdw` dan `dblink` menyediakan fungsionalitas untuk kueri di seluruh database pada instance yang sama atau instance jarak jauh. `log_fdw`membaca file log mesin postgres, yang untuk semua database dalam instance, berpotensi berisi kueri lambat atau pesan kesalahan dari beberapa database. `pg_cron`memungkinkan menjalankan pekerjaan latar belakang terjadwal pada instans DB dan dapat mengonfigurasi pekerjaan untuk dijalankan di database yang berbeda.

Jatuhkan kaskade ekstensi dinonaktifkan

Kemampuan untuk menjatuhkan ekstensi dengan opsi kaskade oleh pengguna dengan `rds_extension` peran dikendalikan oleh `rds.delegated_extension_allow_drop_cascade` parameter. Secara default, `rds-delegated_extension_allow_drop_cascade` diatur ke `off`. Ini berarti bahwa pengguna dengan `rds_extension` peran tidak diizinkan untuk menjatuhkan ekstensi menggunakan opsi kaskade seperti yang ditunjukkan pada kueri di bawah ini.

```
DROP EXTENSION CASCADE;
```

Karena ini akan secara otomatis menjatuhkan objek yang bergantung pada ekstensi, dan pada gilirannya semua objek yang bergantung pada objek tersebut. Mencoba menggunakan opsi kaskade akan menghasilkan kesalahan.

Untuk memberikan kemampuan itu, `rds.delegated_extension_allow_drop_cascade` parameter harus diatur keon.

Mengubah parameter `rds.delegated_extension_allow_drop_cascade` dinamis tidak memerlukan restart database. Anda dapat melakukan ini di salah satu level berikut:

- Di cluster atau grup parameter instance, melalui AWS Management Console atau API.
- Menggunakan perintah berikut di tingkat database:

  ```
  alter database database_name set rds.delegated_extension_allow_drop_cascade = 'on';
  ```

- Menggunakan perintah berikut di tingkat pengguna:

  ```
  alter role tenant_user set rds.delegated_extension_allow_drop_cascade = 'on';
  ```

Contoh ekstensi yang dapat ditambahkan menggunakan dukungan ekstensi yang didelegasikan

- `rds_tools`

```
extension_test_db=> create extension rds_tools;
CREATE EXTENSION
extension_test_db=> SELECT * from rds_tools.role_password_encryption_type() where
 rolname = 'pg_read_server_files';
ERROR: permission denied for function role_password_encryption_type
```

- `amcheck`

```
extension_test_db=> CREATE TABLE amcheck_test (id int);
CREATE TABLE
extension_test_db=> INSERT INTO amcheck_test VALUES (generate_series(1,100000));
INSERT 0 100000
extension_test_db=> CREATE INDEX amcheck_test_btree_idx ON amcheck_test USING btree
 (id);
CREATE INDEX
extension_test_db=> create extension amcheck;
CREATE EXTENSION
extension_test_db=> SELECT bt_index_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_check
extension_test_db=> SELECT bt_index_parent_check('amcheck_test_btree_idx'::regclass);
ERROR: permission denied for function bt_index_parent_check
```

- `pg_freespacemap`

```
extension_test_db=> create extension pg_freespacemap;
CREATE EXTENSION
extension_test_db=> SELECT * FROM pg_freespace('pg_authid');
ERROR: permission denied for function pg_freespace
extension_test_db=> SELECT * FROM pg_freespace('pg_authid',0);
ERROR: permission denied for function pg_freespace
```

- `pg_visibility`

```
extension_test_db=> create extension pg_visibility;
CREATE EXTENSION
extension_test_db=> select * from pg_visibility('pg_database'::regclass);
ERROR: permission denied for function pg_visibility
```

- `postgres_fdw`

```
extension_test_db=> create extension postgres_fdw;
CREATE EXTENSION
extension_test_db=> create server myserver foreign data wrapper postgres_fdw options
 (host 'foo', dbname 'foodb', port '5432');
ERROR: permission denied for foreign-data wrapper postgres_fdw
```

Mengelola objek besar dengan modul lo

Modul (ekstensi) lo ditujukan bagi pengguna dan pengembang basis data yang bekerja dengan basis data PostgreSQL melalui driver JDBC atau ODBC. Baik JDBC maupun ODBC mengharapkan basis data menangani penghapusan objek besar ketika referensi ke objek tersebut berubah. Namun, cara kerja PostgreSQL bukan seperti itu. PostgreSQL tidak berasumsi bahwa suatu objek harus dihapus ketika referensinya berubah. Hasilnya adalah bahwa objek tetap berda di disk, tidak direferensikan. Ekstensi lo mencakup fungsi yang Anda gunakan untuk memicu perubahan referensi untuk menghapus objek jika diperlukan.

> **Tip**
>
> Untuk menentukan apakah basis data Anda dapat memperoleh manfaat dari ekstensi lo, gunakan utilitas vacuumlo untuk memeriksa objek besar tak berinduk. Untuk mendapatkan hitungan objek tak berinduk tanpa mengambil tindakan apa pun, jalankan utilitas dengan opsi `-n` (no-op). Untuk mempelajari caranya, lihat vacuumlo utility berikut.

Modul lo tersedia untuk Aurora PostgreSQL 13.7, 12.11, 11.16, 10.21, dan versi minor yang lebih tinggi.

Untuk menginstal modul (ekstensi), Anda memerlukan hak istimewa `rds_superuser`. Menginstal ekstensi lo akan menambahkan berikut ke basis data Anda:

- `lo` – Ini adalah jenis data objek besar (lo) yang dapat digunakan untuk objek besar biner (BLOB) dan objek besar lainnya. Tipe data `lo` adalah domain dari jenis data `oid`. Dengan kata lain, ini adalah ID objek dengan kendala opsional. Untuk informasi selengkapnya, lihat ID objek dalam dokumentasi PostgreSQL. Sederhananya, Anda dapat menggunakan jenis data `lo` untuk membedakan kolom basis data yang menyimpan referensi objek besar dengan ID objek lain (OID).

- `lo_manage` – Ini adalah fungsi yang dapat digunakan dalam pemicu pada kolom tabel yang berisi referensi objek besar. Setiap kali Anda menghapus atau mengubah nilai yang mereferensikan objek besar, pemicu akan memutuskan tautan objek (`lo_unlink`) dari referensinya. Gunakan pemicu pada kolom hanya jika kolom tersebut adalah referensi basis data tunggal ke objek besar.

Untuk informasi selengkapnya tentang modul objek besar, lihat lo dalam dokumentasi PostgreSQL.

Menginstal ekstensi lo

Sebelum menginstal ekstensi lo, pastikan bahwa Anda memiliki hak istimewa `rds_superuser`.

Untuk menginstal ekstensi lo.

1. Gunakan `psql` untuk terhubung ke instans DB primer dari klaster DB Aurora PostgreSQL Anda.

   ```
   psql --host=your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com --
   port=5432 --username=postgres --password
   ```

   Jika diminta, masukkan kata sandi Anda. Klien `psql` menghubungkan dan menampilkan basis data koneksi administratif default, `postgres=>`, sebagai perintah.
2. Instal ekstensi seperti berikut.

   ```
   postgres=> CREATE EXTENSION lo;
   CREATE EXTENSION
   ```

Anda kini dapat menggunakan jenis data `lo` untuk menentukan kolom dalam tabel Anda. Misalnya, Anda dapat membuat tabel (`images`) yang berisi data citra raster. Anda dapat menggunakan jenis data `lo` untuk kolom `raster`, seperti yang ditunjukkan pada contoh berikut, yang membuat tabel.

```
postgres=> CREATE TABLE images (image_name text, raster lo);
```

Menggunakan fungsi pemicu lo_manage untuk menghapus objek

Anda dapat menggunakan fungsi `lo_manage` dalam pemicu pada `lo` atau kolom objek besar lainnya untuk membersihkan (dan mencegah objek tak berinduk) ketika `lo` diperbarui atau dihapus.

Untuk menyiapkan pemicu pada kolom yang mereferensikan objek besar

- Lakukan salah satu langkah berikut:
  - Buat pemicu BEFORE UPDATE OR DELETE pada setiap kolom agar berisi referensi unik ke objek besar, menggunakan nama kolom untuk argumen.

```
postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OR DELETE ON images
    FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
```

  - Terapkan pemicu hanya ketika kolom sedang diperbarui.

```
postgres=> CREATE TRIGGER t_raster BEFORE UPDATE OF images
    FOR EACH ROW EXECUTE FUNCTION lo_manage(raster);
```

Fungsi `lo_manage` pemicu hanya berfungsi saat memasukkan atau menghapus data kolom, bergantung pada bagaimana Anda mendefinisikan pemicu. Ini tidak berpengaruh ketika Anda melakukan operasi `DROP` atau `TRUNCATE` pada basis data. Berarti Anda harus menghapus kolom objek dari tabel apa pun sebelum hilang, untuk mencegah pembuatan objek tak berinduk.

Misalnya, anggaplah Anda ingin menghilangkan basis data yang berisi tabel `images`. Anda menghapus kolom sebagai berikut.

```
postgres=> DELETE FROM images COLUMN raster
```

Dengan asumsi bahwa fungsi `lo_manage` didefinisikan pada kolom itu untuk menangani penghapusan, Anda sekarang dapat dengan aman menghilangkan tabel.

Menggunakan utilitas vacuumlo

Utilitas vacuumlo mengidentifikasi dan dapat menghapus objek besar tak berinduk dari basis data. Utilitas ini telah tersedia sejak PostgreSQL 9.1.24. Jika pengguna basis data Anda secara rutin bekerja dengan objek besar, sebaiknya Anda menjalankan vacuumlo sesekali untuk membersihkan objek besar tak berinduk.

Sebelum menginstal ekstensi lo, Anda dapat menggunakan vacuumlo untuk menilai apakah klaster DB Aurora PostgreSQL Anda dapat memperoleh manfaat. Untuk melakukannya, jalankan `vacuumlo` dengan opsi `-n` (no-op) untuk menunjukkan apa yang akan dihapus, seperti yang ditunjukkan berikut:

```
$ vacuumlo -v -n -h your-cluster-instance-1.666666666666.aws-region.rds.amazonaws.com -
p 5433 -U postgres docs-lab-spatial-db
Password:*****
Connected to database "docs-lab-spatial-db"
Test run: no large objects will be removed!
Would remove 0 large objects from database "docs-lab-spatial-db".
```

Seperti yang ditunjukkan output, objek besar tak berinduk tidak menjadi masalah untuk basis data khusus ini.

Untuk informasi selengkapnya, lihat vacuumlo dalam dokumentasi PostgreSQL.

Mengelola data spasial dengan ekstensi PostGIS

PostGIS adalah ekstensi dari PostgreSQL untuk menyimpan dan mengelola informasi spasial. Untuk mempelajari selengkapnya tentang PostGIS, lihat PostGIS.net.

Mulai dari versi 10.5, PostgreSQL mendukung pustaka libprotobuf 1.3.0 yang digunakan oleh PostGIS agar berfungsi dengan data petak vektor map box.

Menyiapkan ekstensi PostGIS membutuhkan hak akses `rds_superuser`. Sebaiknya Anda membuat (peran) pengguna untuk mengelola ekstensi PostGIS dan data spasial. Ekstensi PostGIS dan komponen terkaitnya menambahkan ribuan fungsi ke PostgreSQL. Anda juga dapat membuat ekstensi PostGIS dalam skema tersendiri jika sesuai untuk kasus penggunaan Anda. Contoh berikut menunjukkan cara menginstal ekstensi dalam basis data tersendiri, tetapi ini tidak wajib dilakukan.

Topik

- Langkah 1: Membuat (peran) pengguna untuk mengelola ekstensi PostGIS
- Langkah 2: Memuat ekstensi PostGIS
- Langkah 3: Mentransfer kepemilikan ekstensi
- Langkah 4: Mentransfer kepemilikan objek PostGIS
- Langkah 5: Menguji ekstensi
- Langkah 6: Meningkatkan ekstensi PostGIS
- Versi ekstensi PostGIS
- Meningkatkan PostGIS 2 ke PostGIS 3

Langkah 1: Membuat (peran) pengguna untuk mengelola ekstensi PostGIS

Pertama, hubungkan ke instans DB RDS for PostgreSQL sebagai pengguna yang memiliki hak akses `rds_superuser`. Jika tidak mengubah nama default saat menyiapkan instans, Anda akan terhubung sebagai `postgres`.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --username=postgres
 --password
```

Buat peran (pengguna) terpisah untuk mengelola ekstensi PostGIS.

```
postgres=>  CREATE ROLE gis_admin LOGIN PASSWORD 'change_me';
CREATE ROLE
```

Berikan hak akses `rds_superuser` kepada peran ini agar dapat menginstal ekstensi.

```
postgres=> GRANT rds_superuser TO gis_admin;
GRANT
```

Buat basis data agar dapat digunakan untuk artefak PostGIS. Langkah ini bersifat opsional. Atau, Anda dapat membuat skema di basis data pengguna untuk ekstensi PostGIS, tetapi ini juga tidak wajib dilakukan.

```
postgres=> CREATE DATABASE lab_gis;
CREATE DATABASE
```

Berikan semua hak akses `gis_admin` pada basis data `lab_gis`.

```
postgres=> GRANT ALL PRIVILEGES ON DATABASE lab_gis TO gis_admin;
GRANT
```

Keluar dari sesi, lalu sambungkan kembali ke instans DB RDS for PostgreSQL Anda sebagai `gis_admin`.

```
postgres=> psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=gis_admin --password --dbname=lab_gis
Password for user gis_admin:...
lab_gis=>
```

Lanjutkan menyiapkan ekstensi seperti yang dijelaskan pada langkah berikutnya.

Langkah 2: Memuat ekstensi PostGIS

Ekstensi PostGIS mencakup beberapa ekstensi terkait yang berfungsi bersama untuk menyediakan fungsionalitas geospasial. Tergantung pada kasus penggunaannya, Anda mungkin tidak memerlukan semua ekstensi yang dibuat pada langkah ini.

Gunakan pernyataan `CREATE EXTENSION` untuk memuat ekstensi PostGIS.

```
CREATE EXTENSION postgis;
CREATE EXTENSION
CREATE EXTENSION postgis_raster;
CREATE EXTENSION
CREATE EXTENSION fuzzystrmatch;
CREATE EXTENSION
CREATE EXTENSION postgis_tiger_geocoder;
CREATE EXTENSION
CREATE EXTENSION postgis_topology;
CREATE EXTENSION
CREATE EXTENSION address_standardizer_data_us;
CREATE EXTENSION
```

Anda dapat memverifikasi hasilnya dengan menjalankan kueri SQL yang ditunjukkan dalam contoh berikut, yang mencantumkan ekstensi dan pemiliknya.

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;
List of schemas
     Name     |   Owner
--------------+-----------
 public       | postgres
 tiger        | rdsadmin
 tiger_data   | rdsadmin
 topology     | rdsadmin
(4 rows)
```

Langkah 3: Mentransfer kepemilikan ekstensi

Gunakan pernyataan ALTER SCHEMA untuk mentransfer kepemilikan skema ke peran `gis_admin`.

```
ALTER SCHEMA tiger OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA tiger_data OWNER TO gis_admin;
ALTER SCHEMA
ALTER SCHEMA topology OWNER TO gis_admin;
ALTER SCHEMA
```

Anda dapat mengonfirmasi perubahan kepemilikan dengan menjalankan kueri SQL berikut. Bisa juga dengan menggunakan metacommand \dn dari baris perintah psql.

```
SELECT n.nspname AS "Name",
  pg_catalog.pg_get_userbyid(n.nspowner) AS "Owner"
  FROM pg_catalog.pg_namespace n
  WHERE n.nspname !~ '^pg_' AND n.nspname <> 'information_schema'
  ORDER BY 1;

       List of schemas
     Name     |     Owner
--------------+---------------
 public       | postgres
 tiger        | gis_admin
 tiger_data   | gis_admin
 topology     | gis_admin
(4 rows)
```

Langkah 4: Mentransfer kepemilikan objek PostGIS

Gunakan fungsi berikut untuk mentransfer kepemilikan objek PostGIS ke peran `gis_admin`. Jalankan pernyataan berikut dari perintah psql untuk membuat fungsinya.

```
CREATE FUNCTION exec(text) returns text language plpgsql volatile AS $f$ BEGIN EXECUTE
 $1; RETURN $1; END; $f$;
CREATE FUNCTION
```

Selanjutnya, jalankan kueri berikut untuk menjalankan fungsi `exec` yang nantinya akan menjalankan pernyataan dan mengubah izin.

```
SELECT exec('ALTER TABLE ' || quote_ident(s.nspname) || '.' || quote_ident(s.relname)
 || ' OWNER TO gis_admin;')
  FROM (
    SELECT nspname, relname
```

```
    FROM pg_class c JOIN pg_namespace n ON (c.relnamespace = n.oid)
    WHERE nspname in ('tiger','topology') AND
    relkind IN ('r','S','v') ORDER BY relkind = 'S')
s;
```

Langkah 5: Menguji ekstensi

Agar Anda tidak harus menentukan nama skema, tambahkan skema `tiger` ke jalur pencarian menggunakan perintah berikut.

```
SET search_path=public,tiger;
SET
```

Uji skema `tiger` menggunakan pernyataan SELECT berikut.

```
SELECT address, streetname, streettypeabbrev, zip
 FROM normalize_address('1 Devonshire Place, Boston, MA 02109') AS na;
address | streetname | streettypeabbrev |  zip
---------+------------+------------------+-------
       1 | Devonshire | Pl               | 02109
(1 row)
```

Untuk mempelajari selengkapnya tentang ekstensi ini, lihat Tiger Geocode dalam dokumentasi PostGIS.

Uji akses ke skema `topology` menggunakan pernyataan `SELECT` berikut. Tindakan ini akan memanggil fungsi `createtopology` untuk mendaftarkan objek topologi baru (my_new_topo) dengan pengidentifikasi referensi spasial (26986) dan toleransi default (0,5) yang ditentukan. Untuk mempelajari selengkapnya, lihat CreateTopology dalam dokumentasi PostGIS.

```
SELECT topology.createtopology('my_new_topo',26986,0.5);
 createtopology
----------------
              1
(1 row)
```

Langkah 6: Meningkatkan ekstensi PostGIS

Setiap rilis baru PostgreSQL mendukung satu atau beberapa versi ekstensi PostGIS yang kompatibel dengan rilis tersebut. Meningkatkan mesin PostgreSQL ke versi baru tidak secara otomatis

meningkatkan ekstensi PostGIS. Sebelum meningkatkan mesin PostgreSQL, Anda biasanya meningkatkan PostGIS ke versi terbaru yang tersedia untuk versi PostgreSQL saat ini. Lihat perinciannya di Versi ekstensi PostGIS.

Setelah peningkatan mesin PostgreSQL, selanjutnya tingkatkan ekstensi PostGIS lagi, ke versi yang didukung untuk versi mesin PostgreSQL yang baru ditingkatkan. Untuk informasi selengkapnya tentang cara meningkatkan mesin PostgreSQL, lihat Menguji peningkatan klaster DB produksi ke versi mayor baru.

Anda dapat memeriksa pembaruan versi ekstensi PostGIS yang tersedia di klaster DB Aurora PostgreSQL kapan saja. Untuk melakukannya, jalankan perintah berikut. Fungsi ini tersedia untuk PostGIS 2.5.0 dan versi yang lebih baru.

```
SELECT postGIS_extensions_upgrade();
```

Jika aplikasi Anda tidak mendukung versi PostGIS terbaru, Anda dapat menginstal versi PostGIS lama yang tersedia di versi utama Anda sebagai berikut.

```
CREATE EXTENSION postgis VERSION "2.5.5";
```

Jika ingin meningkatkan ke versi PostGIS tertentu dari versi lama, Anda juga dapat menggunakan perintah berikut.

```
ALTER EXTENSION postgis UPDATE TO "2.5.5";
```

Tergantung pada versi sebelum peningkatan, Anda mungkin perlu menggunakan fungsi ini lagi. Hasil dari menjalankan fungsi pertama akan menentukan perlu atau tidaknya fungsi peningkatan lain. Misalnya, ini adalah kasus peningkatan dari PostGIS 2 ke PostGIS 3. Untuk informasi selengkapnya, lihat Meningkatkan PostGIS 2 ke PostGIS 3.

Jika meningkatkan ekstensi ini untuk mempersiapkan peningkatan versi utama mesin PostgreSQL, Anda dapat melanjutkan tugas awal lainnya. Untuk informasi selengkapnya, lihat Menguji peningkatan klaster DB produksi ke versi mayor baru.

Versi ekstensi PostGIS

Sebaiknya, Anda menginstal versi semua ekstensi seperti PostGIS sebagaimana tercantum di Versi ekstensi untuk Aurora PostgreSQL-Compatible Edition dalam Catatan Rilis Aurora PostgreSQL. Untuk mendapatkan daftar versi yang tersedia dalam rilis Anda, gunakan perintah berikut.

```
SELECT * FROM pg_available_extension_versions WHERE name='postgis';
```

Anda dapat menemukan informasi versi di bagian berikut dalam Catatan Rilis Aurora PostgreSQL:

- Versi ekstensi untuk Aurora PostgreSQL 14
- Versi ekstensi untuk Aurora PostgreSQL-Compatible Edition 13
- Versi ekstensi untuk Aurora PostgreSQL-Compatible Edition 12
- Versi ekstensi untuk Aurora PostgreSQL-Compatible Edition 11
- Versi ekstensi untuk Aurora PostgreSQL-Compatible Edition 10
- Versi ekstensi untuk Aurora PostgreSQL-Compatible Edition 9.6

Meningkatkan PostGIS 2 ke PostGIS 3

Mulai dari versi 3.0, fungsi raster PostGIS kini menjadi ekstensi terpisah, `postgis_raster`. Ekstensi ini memiliki jalur instalasi dan peningkatan tersendiri. Ekstensi ini menghapus berbagai fungsi, tipe data, dan artefak lain yang diperlukan untuk pemrosesan gambar raster dari ekstensi `postgis` inti. Artinya, jika kasus penggunaan Anda tidak memerlukan pemrosesan raster, Anda tidak perlu menginstal ekstensi `postgis_raster`.

Dalam contoh peningkatan berikut, perintah peningkatan pertama mengekstrak fungsionalitas raster ke dalam ekstensi `postgis_raster`. Selanjutnya, perintah peningkatan kedua diperlukan untuk meningkatkan `postgres_raster` ke versi baru.

Untuk meningkatkan dari PostGIS 2 ke PostGIS 3

1. Identifikasi versi default PostGIS yang tersedia untuk versi PostgreSQL di klaster DB Aurora PostgreSQL. Untuk melakukannya, jalankan kueri berikut.

```
SELECT * FROM pg_available_extensions
    WHERE default_version > installed_version;
  name   | default_version | installed_version |                          comment
---------+-----------------+-------------------
+-------------------------------------------------------------
 postgis | 3.1.4           | 2.3.7             | PostGIS geometry and geography
 spatial types and functions
(1 row)
```

2. Identifikasi versi PostGIS yang diinstal di setiap basis data pada instans penulis klaster DB Aurora PostgreSQL. Dengan kata lain, buat kueri setiap basis data pengguna sebagai berikut.

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
        AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
  Name   | Version | Schema |                            Description
---------+---------+--------
+------------------------------------------------------------------------
 postgis | 2.3.7   | public | PostGIS geometry, geography, and raster spatial types
 and functions
(1 row)
```

Ketidakcocokan antara versi default (PostGIS 3.1.4) dan versi yang diinstal (PostGIS 2.3.7) ini mengindikasikan bahwa Anda perlu meningkatkan ekstensi PostGIS.

```
ALTER EXTENSION postgis UPDATE;
ALTER EXTENSION
WARNING: unpackaging raster
WARNING: PostGIS Raster functionality has been unpackaged
```

3. Jalankan kueri berikut untuk memverifikasi bahwa fungsi raster kini telah berada dalam paketnya tersendiri.

```
SELECT
    probin,
    count(*)
FROM
    pg_proc
WHERE
    probin LIKE '%postgis%'
```

```
GROUP BY
    probin;
          probin          | count
--------------------------+-------
 $libdir/rtpostgis-2.3    | 107
 $libdir/postgis-3        | 487
(2 rows)
```

Output-nya menunjukkan bahwa masih ada perbedaan antarversi. Fungsi PostGIS adalah versi 3 (postgis-3), sedangkan fungsi raster (rtpostgis) adalah versi 2 (rtpostgis-2.3). Untuk menyelesaikan peningkatan, Anda dapat menjalankan perintah peningkatan lagi sebagai berikut.

```
postgres=> SELECT postgis_extensions_upgrade();
```

Anda dapat dengan aman mengabaikan pesan peringatan. Jalankan lagi kueri berikut untuk memverifikasi bahwa peningkatan telah selesai. Peningkatan selesai saat PostGIS dan semua ekstensi terkait tidak ditandai sebagai perlu peningkatan.

```
SELECT postgis_full_version();
```

4. Gunakan kueri berikut untuk melihat proses peningkatan yang telah selesai dan ekstensi yang dikemas secara terpisah, lalu verifikasi bahwa versinya telah sesuai.

```
SELECT
    e.extname AS "Name",
    e.extversion AS "Version",
    n.nspname AS "Schema",
    c.description AS "Description"
FROM
    pg_catalog.pg_extension e
    LEFT JOIN pg_catalog.pg_namespace n ON n.oid = e.extnamespace
    LEFT JOIN pg_catalog.pg_description c ON c.objoid = e.oid
        AND c.classoid = 'pg_catalog.pg_extension'::pg_catalog.regclass
WHERE
    e.extname LIKE '%postgis%'
ORDER BY
    1;
      Name      | Version | Schema |                            Description
----------------+---------+--------
+------------------------------------------------------------------------
```

```
 postgis        | 3.1.5   | public | PostGIS geometry, geography, and raster
 spatial types and functions
 postgis_raster | 3.1.5   | public | PostGIS raster types and functions
(2 rows)
```

Output menunjukkan bahwa ekstensi PostGIS 2 telah ditingkatkan ke PostGIS 3, serta ekstensi `postgis` dan ekstensi `postgis_raster` yang sekarang terpisah adalah versi 3.1.5.

Setelah peningkatan ini selesai, Anda dapat menghapus ekstensi seperti berikut jika tidak berencana menggunakan fungsionalitas raster.

```
DROP EXTENSION postgis_raster;
```

Mengelola partisi PostgreSQL dengan ekstensi pg_partman

Partisi tabel PostgreSQL menyediakan kerangka kerja untuk penanganan input data dan laporan performa tinggi. Gunakan partisi untuk basis data yang memerlukan input yang sangat cepat berupa data dalam jumlah besar. Partisi juga menyediakan kueri tabel besar yang lebih cepat. Partisi membantu menjaga data tanpa memengaruhi instans basis data karena membutuhkan lebih sedikit sumber daya I/O.

Dengan partisi, Anda dapat membagi data menjadi potongan-potongan berukuran kustom untuk diproses. Misalnya, Anda dapat mempartisi data deret waktu selama berbagai rentang seperti per jam, harian, mingguan, bulanan, triwulanan, tahunan, kustom, atau kombinasinya. Untuk contoh data deret waktu, jika Anda mempartisi tabel per jam, setiap partisi berisi data satu jam. Jika Anda mempartisi tabel deret waktu per hari, partisi akan menyimpan data senilai satu hari, dan seterusnya. Kunci partisi mengontrol ukuran partisi.

Saat Anda menggunakan perintah SQL `INSERT` atau `UPDATE` pada tabel yang dipartisi, mesin basis data merutekan data ke partisi yang sesuai. Partisi tabel PostgreSQL yang menyimpan data adalah tabel turunan dari tabel utama.

Selama pembacaan kueri basis data, pengoptimal PostgreSQL memeriksa klausul `WHERE` pada kueri dan, jika memungkinkan, mengarahkan pemindaian basis data hanya untuk partisi yang relevan.

Dimulai dengan versi 10, PostgreSQL menggunakan partisi deklaratif untuk mengimplementasikan partisi tabel. Ini juga dikenal sebagai partisi PostgreSQL native. Sebelum PostgreSQL versi 10, Anda menggunakan pemicu untuk mengimplementasikan partisi.

Partisi tabel PostgreSQL dilengkapi dengan fitur berikut:

- Pembuatan partisi baru setiap saat.
- Rentang partisi bervariasi.
- Partisi yang dapat dilepas dan dapat dipasang kembali menggunakan pernyataan bahasa definisi data (DDL).

  Sebagai contoh, partisi yang dapat dilepas berguna untuk menghapus data historis dari partisi utama, tetapi menyimpan data historis untuk analisis.
- Partisi baru mewarisi properti tabel basis data induk, termasuk yang berikut ini:
  - Indeks
  - Kunci primer, yang harus berisi kolom kunci partisi
  - Kunci asing
  - Kendala pemeriksaan
  - Referensi
- Membuat indeks untuk tabel lengkap atau setiap partisi tertentu.

Anda tidak dapat mengubah skema untuk partisi individual. Namun, Anda dapat mengubah tabel induk (seperti menambahkan kolom baru), yang bertambah ke partisi.

Topik

- Ikhtisar ekstensi PostgreSQL pg_partman
- Mengaktifkan ekstensi pg_partman
- Mengonfigurasi partisi menggunakan fungsi create_parent
- Mengonfigurasi pemeliharaan partisi menggunakan fungsi run_maintenance_proc

Ikhtisar ekstensi PostgreSQL pg_partman

Anda dapat menggunakan ekstensi PostgreSQL `pg_partman` untuk mengotomatisasi pembuatan dan pemeliharaan partisi tabel. Untuk informasi umum selengkapnya, lihat Manajer Partisi PG dalam dokumentasi `pg_partman`.

> **Note**
>
> Ekstensi `pg_partman` didukung pada RDS for PostgreSQL versi 12.6 dan yang lebih tinggi.

Alih-alih membuat setiap partisi secara manual, Anda mengonfigurasi `pg_partman` dengan pengaturan berikut:

- Tabel yang akan dipartisi
- Jenis partisi
- Kunci partisi
- Granularitas partisi
- Opsi pra-pembuatan dan manajemen partisi

Setelah membuat tabel yang dipartisi PostgreSQL, Anda mendaftarkannya dengan `pg_partman` dengan memanggil fungsi `create_parent`. Melakukan hal ini akan membuat partisi yang diperlukan berdasarkan parameter yang Anda teruskan ke fungsi.

Ekstensi `pg_partman` juga menyediakan fungsi `run_maintenance_proc`, yang dapat Anda panggil sesuai jadwal untuk secara otomatis mengelola partisi. Untuk memastikan bahwa partisi yang tepat dibuat sesuai kebutuhan, jadwalkan fungsi ini untuk berjalan secara berkala (seperti per jam). Anda juga dapat memastikan bahwa partisi secara otomatis dibatalkan.

Mengaktifkan ekstensi pg_partman

Jika Anda memiliki beberapa basis data di dalam instans DB PostgreSQL yang partisinya ingin Anda kelola, aktifkan ekstensi `pg_partman` secara terpisah untuk setiap basis data. Untuk mengaktifkan ekstensi `pg_partman` untuk basis data tertentu, buat skema pemeliharaan partisi, kemudian buat ekstensi `pg_partman` seperti berikut.

```
CREATE SCHEMA partman;
CREATE EXTENSION pg_partman WITH SCHEMA partman;
```

> **Note**
>
> Untuk membuat ekstensi `pg_partman`, pastikan Anda memiliki hak istimewa `rds_superuser`.

Jika Anda menerima kesalahan seperti berikut, berikan hak istimewa `rds_superuser` untuk akun tersebut atau gunakan akun pengguna super Anda.

```
ERROR: permission denied to create extension "pg_partman"
HINT: Must be superuser to create this extension.
```

Untuk memberikan hak istimewa `rds_superuser`, hubungkan dengan akun pengguna super Anda dan jalankan perintah berikut.

```
GRANT rds_superuser TO user-or-role;
```

Untuk contoh yang menunjukkan penggunaan ekstensi pg_partman, kita gunakan contoh tabel dan partisi basis data berikut. Basis data ini menggunakan tabel yang dipartisi berdasarkan stempel waktu. Skema `data_mart` berisi tabel bernama `events` dengan kolom bernama `created_at`. Pengaturan berikut disertakan dalam tabel `events`:

- Kunci primer `event_id` dan `created_at`, yang harus memiliki kolom yang digunakan untuk memandu partisi.
- Kendala pemeriksaan `ck_valid_operation` guna menerapkan nilai untuk kolom tabel `operation`.
- Dua kunci asing, di mana satu kunci (`fk_orga_membership`) menunjuk ke tabel eksternal `organization` dan kunci lainnya (`fk_parent_event_id`) adalah kunci asing referensi mandiri.
- Dua indeks, di mana satu induk (`idx_org_id`) untuk kunci asing dan indeks lainnya (`idx_event_type`) untuk jenis peristiwa.

Pernyataan DDL berikut membuat objek ini, yang secara otomatis disertakan pada setiap partisi.

```
CREATE SCHEMA data_mart;
CREATE TABLE data_mart.organization ( org_id BIGSERIAL,
        org_name TEXT,
        CONSTRAINT pk_organization PRIMARY KEY (org_id)
    );

CREATE TABLE data_mart.events(
        event_id        BIGSERIAL,
        operation       CHAR(1),
        value           FLOAT(24),
        parent_event_id BIGINT,
```

```
        event_type      VARCHAR(25),
        org_id          BIGSERIAL,
        created_at      timestamp,
        CONSTRAINT pk_data_mart_event PRIMARY KEY (event_id, created_at),
        CONSTRAINT ck_valid_operation CHECK (operation = 'C' OR operation = 'D'),
        CONSTRAINT fk_orga_membership
            FOREIGN KEY(org_id)
            REFERENCES data_mart.organization (org_id),
        CONSTRAINT fk_parent_event_id
            FOREIGN KEY(parent_event_id, created_at)
            REFERENCES data_mart.events (event_id,created_at)
    ) PARTITION BY RANGE (created_at);

CREATE INDEX idx_org_id     ON  data_mart.events(org_id);
CREATE INDEX idx_event_type ON  data_mart.events(event_type);
```

Mengonfigurasi partisi menggunakan fungsi create_parent

Setelah Anda mengaktifkan ekstensi `pg_partman`, gunakan fungsi `create_parent` untuk mengonfigurasi partisi dalam skema pemeliharaan partisi. Contoh berikut menggunakan contoh tabel `events` yang dibuat di Mengaktifkan ekstensi pg_partman. Panggil fungsi `create_parent` seperti berikut.

```
SELECT partman.create_parent( p_parent_table => 'data_mart.events',
 p_control => 'created_at',
 p_type => 'native',
 p_interval=> 'daily',
 p_premake => 30);
```

Parameternya adalah sebagai berikut:

- `p_parent_table` – Tabel induk yang dipartisi. Tabel ini harus sudah ada dan sepenuhnya memenuhi syarat, termasuk skemanya.
- `p_control` – Kolom yang menjadi dasar pembuatan partisi. Jenis data harus bilangan bulat atau berbasis waktu.
- `p_type` – Jenisnya adalah `'native'` atau `'partman'`. Anda biasanya menggunakan jenis `native` untuk perbaikan performa dan fleksibilitas. Jenis `partman` bergantung pada warisan.
- `p_interval` – Interval waktu atau rentang bilangan bulat untuk setiap partisi. Contoh nilainya termasuk `daily`, per jam, dan sebagainya.

- p_premake – Jumlah partisi yang akan dibuat terlebih dahulu untuk mendukung sisipan baru.

Untuk keterangan lengkap tentang fungsi create_parent, lihat Fungsi Pembuatan dalam dokumentasi pg_partman.

Mengonfigurasi pemeliharaan partisi menggunakan fungsi run_maintenance_proc

Anda dapat menjalankan operasi pemeliharaan partisi untuk secara otomatis membuat partisi baru, melepaskan partisi, atau menghapus partisi lama. Pemeliharaan partisi bergantung pada fungsi run_maintenance_proc ekstensi pg_partman dan ekstensi pg_cron, yang memulai penjadwal internal. Penjadwal pg_cron secara otomatis mengeksekusi pernyataan, fungsi, dan prosedur SQL yang ditetapkan dalam basis data Anda.

Contoh berikut menggunakan contoh tabel events yang dibuat di Mengaktifkan ekstensi pg_partman untuk mengatur operasi pemeliharaan partisi agar berjalan secara otomatis. Sebagai prasyarat, tambahkan pg_cron ke parameter shared_preload_libraries dalam grup parameter instans DB.

```
CREATE EXTENSION pg_cron;

UPDATE partman.part_config
SET infinite_time_partitions = true,
    retention = '3 months',
    retention_keep_table=true
WHERE parent_table = 'data_mart.events';
SELECT cron.schedule('@hourly', $$CALL partman.run_maintenance_proc()$$);
```

Berikut ini, Anda dapat menemukan penjelasan langkah demi langkah tentang contoh sebelumnya:

1. Ubah grup parameter yang terkait dengan instans DB Anda dan tambahkan pg_cron ke nilai parameter shared_preload_libraries. Perubahan ini mengharuskan instans DB dimulai ulang agar dapat diterapkan. Untuk informasi selengkapnya, lihat Memodifikasi parameter dalam grup parameter DB.
2. Jalankan perintah CREATE EXTENSION pg_cron; menggunakan akun yang memiliki izin rds_superuser. Langkah ini akan mengaktifkan ekstensi pg_cron. Untuk informasi selengkapnya, lihat Menjadwalkan pemeliharaan dengan ekstensi pg_cron PostgreSQL.
3. Jalankan perintah UPDATE partman.part_config guna menyesuaikan pengaturan pg_partman untuk tabel data_mart.events.

4. Jalankan perintah SET . . . Untuk mengonfigurasi tabel `data_mart.events`, dengan klausul ini:
   a. `infinite_time_partitions = true,` – Mengonfigurasi tabel untuk dapat membuat partisi baru secara otomatis tanpa batas.
   b. `retention = '3 months',` – Mengonfigurasi tabel agar memiliki retensi maksimum tiga bulan.
   c. `retention_keep_table=true` – Mengonfigurasi tabel sehingga ketika periode retensi sudah habis, tabel tidak akan dihapus secara otomatis. Sebaliknya, partisi yang lebih lama dari periode retensi hanya dilepaskan dari tabel induk.
5. Jalankan perintah `SELECT cron.schedule` . . . untuk membuat panggilan fungsi `pg_cron`. Panggilan ini menetapkan frekuensi penjadwal menjalankan prosedur pemeliharaan `pg_partman`, `partman.run_maintenance_proc`. Untuk contoh ini, prosedur berjalan setiap jam.

Untuk keterangan lengkap tentang fungsi `run_maintenance_proc`, lihat Fungsi Pemeliharaan dalam dokumentasi `pg_partman`.

Menjadwalkan pemeliharaan dengan ekstensi pg_cron PostgreSQL

Anda dapat menggunakan ekstensi `pg_cron` PostgreSQL untuk menjadwalkan perintah pemeliharaan dalam basis data PostgreSQL. Untuk informasi selengkapnya tentang ekstensi, lihat Apa itu pg_cron? dalam dokumentasi pg_cron.

Ekstensi `pg_cron` didukung pada mesin Aurora PostgreSQL versi 12.6 dan yang lebih tinggi

Untuk mempelajari selengkapnya tentang penggunaan `pg_cron`, lihat Menjadwalkan pekerjaan dengan pg_cron di RDS for PostgreSQL atau basis data Edisi yang kompatibel dengan Aurora PostgreSQL.

Topik

- Menyiapkan ekstensi pg_cron
- Memberikan izin pengguna basis data untuk menggunakan pg_cron
- Menjadwalkan pekerjaan pg_cron
- Referensi untuk ekstensi pg_cron

Menyiapkan ekstensi pg_cron

Siapkan ekstensi `pg_cron` sebagai berikut:

1. Ubah grup parameter kustom yang terkait dengan instans DB PostgreSQL Anda dengan menambahkan pg_cron ke nilai parameter shared_preload_libraries.

   Mulai ulang instans DB PostgreSQL agar perubahan pada grup parameter dapat diterapkan. Untuk mempelajari selengkapnya tentang bekerja menggunakan grup parameter, lihat Parameter Amazon Aurora PostgreSQL.
2. Setelah instans DB PostgreSQL dimulai ulang, jalankan perintah berikut menggunakan akun yang memiliki izin rds_superuser. Misalnya, jika Anda menggunakan pengaturan default saat membuat klaster DB Aurora PostgreSQL, sambungkan sebagai pengguna postgres dan buat ekstensi.

   ```
   CREATE EXTENSION pg_cron;
   ```

   Penjadwal pg_cron diatur dalam basis data PostgreSQL default bernama postgres. Objek pg_cron dibuat dalam basis data postgres ini dan semua tindakan penjadwalan berjalan dalam basis data ini.
3. Anda dapat menggunakan pengaturan default, atau Anda dapat menjadwalkan pekerjaan untuk berjalan di basis data lain dalam instans DB PostgreSQL Anda. Untuk menjadwalkan pekerjaan untuk basis data lain dalam instans DB PostgreSQL Anda, lihat contoh di Menjadwalkan pekerjaan cron untuk basis data selain basis data default.

Memberikan izin pengguna basis data untuk menggunakan pg_cron

Menginstal ekstensi pg_cron membutuhkan hak istimewa rds_superuser. Namun, izin untuk menggunakan pg_cron dapat diberikan (oleh anggota grup/peran rds_superuser) kepada pengguna basis data lain, sehingga mereka dapat menjadwalkan pekerjaannya sendiri. Sebaiknya Anda memberikan izin untuk skema cron hanya sesuai kebutuhan jika skema tersebut meningkatkan operasi di lingkungan produksi Anda.

Untuk memberikan izin pengguna basis data dalam skema cron, jalankan perintah berikut:

```
postgres=> GRANT USAGE ON SCHEMA cron TO db-user;
```

Perintah ini memberikan izin *db-user* untuk mengakses skema cron untuk menjadwalkan pekerjaan cron untuk objek yang izin aksesnya mereka miliki. Jika pengguna basis data tidak memiliki izin, tugas akan gagal setelah memposting pesan kesalahan ke file postgresql.log, seperti yang ditunjukkan berikut:

```
2020-12-08 16:41:00 UTC::@:[30647]:ERROR: permission denied for table table-name
2020-12-08 16:41:00 UTC::@:[27071]:LOG: background worker "pg_cron" (PID 30647) exited
 with exit code 1
```

Dengan kata lain, pastikan bahwa pengguna database yang diberikan izin pada `cron` skema juga memiliki izin pada objek (tabel, skema, dan sebagainya) yang mereka rencanakan untuk dijadwalkan.

Detail pekerjaan cron dan keberhasilan atau kegagalannya juga ditangkap dalam `cron.job_run_details` tabel. Untuk informasi selengkapnya, lihat Tabel untuk menjadwalkan pekerjaan dan menangkap status .

Menjadwalkan pekerjaan pg_cron

Bagian berikut menunjukkan bagaimana Anda dapat menjadwalkan berbagai tugas manajemen menggunakan pekerjaan `pg_cron`.

> **Note**
>
> Saat Anda membuat pekerjaan `pg_cron`, periksa apakah pengaturan `max_worker_processes` lebih besar dari jumlah `cron.max_running_jobs`. Pekerjaan `pg_cron` akan gagal jika kehabisan proses pekerja latar belakang. Jumlah default pekerjaan `pg_cron` adalah 5. Untuk informasi selengkapnya, lihat Parameter untuk mengelola ekstensi pg_cron.

Topik

- Mengosongkan tabel
- Membersihkan tabel riwayat pg_cron
- Kesalahan pengelogan ke file postgresql.log saja
- Menjadwalkan pekerjaan cron untuk basis data selain basis data default

Mengosongkan tabel

Pengosongan otomatis menangani pemeliharaan vakum untuk kebanyakan kasus. Namun, Anda mungkin ingin menjadwalkan pengosongan tabel tertentu di waktu yang Anda pilih.

Berikut adalah contoh penggunaan fungsi `cron.schedule` untuk menyiapkan pekerjaan untuk menggunakan `VACUUM FREEZE` pada tabel tertentu setiap hari pukul 22.00 (GMT).

```
SELECT cron.schedule('manual vacuum', '0 22 * * *', 'VACUUM FREEZE pgbench_accounts');
 schedule
----------
1
(1 row)
```

Setelah contoh sebelumnya berjalan, Anda dapat memeriksa riwayat di tabel `cron.job_run_details` seperti berikut.

```
postgres=> SELECT * FROM cron.job_run_details;
jobid  | runid | job_pid | database | username | command                        |
 status    | return_message | start_time                    | end_time
-------+-------+---------+----------+----------+--------------------------------
+-----------+----------------+-------------------------------
+--------------------------------
 1     | 1     | 3395    | postgres | adminuser| vacuum freeze pgbench_accounts
 | succeeded | VACUUM         | 2020-12-04 21:10:00.050386+00 | 2020-12-04
 21:10:00.072028+00
(1 row)
```

Berikut ini adalah query dari `cron.job_run_details` tabel untuk melihat pekerjaan gagal.

```
postgres=> SELECT * FROM cron.job_run_details WHERE status = 'failed';
jobid | runid | job_pid | database | username | command                        | status
 | return_message                                | start_time                    |
 end_time
------+-------+---------+----------+----------+--------------------------------+--------
+-----------------------------------------------+-------------------------------
+--------------------------------
 5    | 4     | 30339   | postgres | adminuser| vacuum freeze pgbench_account | failed
 | ERROR: relation "pgbench_account" does not exist | 2020-12-04 21:48:00.015145+00 |
 2020-12-04 21:48:00.029567+00
(1 row)
```

Untuk informasi selengkapnya, lihat Tabel untuk menjadwalkan pekerjaan dan menangkap status .

Membersihkan tabel riwayat pg_cron

Tabel `cron.job_run_details` berisi riwayat pekerjaan cron yang bisa menjadi sangat besar dari waktu ke waktu. Sebaiknya Anda menjadwalkan pekerjaan yang membersihkan tabel ini. Misalnya, menyimpan entri yang bernilai setara seminggu mungkin cukup untuk tujuan pemecahan masalah.

Contoh berikut menggunakan fungsi cron.schedule untuk menjadwalkan pekerjaan yang berjalan setiap hari di tengah malam untuk membersihkan tabel `cron.job_run_details`. Pekerjaan yang disimpan hanya selama tujuh hari terakhir. Gunakan akun `rds_superuser` untuk menjadwalkan pekerjaan seperti berikut.

```
SELECT cron.schedule('0 0 * * *', $$DELETE
    FROM cron.job_run_details
    WHERE end_time < now() - interval '7 days'$$);
```

Untuk informasi selengkapnya, lihat Tabel untuk menjadwalkan pekerjaan dan menangkap status .

Kesalahan pengelogan ke file postgresql.log saja

Untuk mencegah penulisan ke tabel `cron.job_run_details`, ubah grup parameter yang terkait dengan instans DB PostgreSQL dan atur parameter `cron.log_run` ke nonaktif. Ekstensi `pg_cron` tidak lagi menulis ke tabel dan menangkap kesalahan ke file `postgresql.log` saja. Untuk informasi selengkapnya, lihat Memodifikasi parameter dalam grup parameter DB.

Gunakan perintah berikut untuk memeriksa nilai parameter `cron.log_run`.

```
postgres=> SHOW cron.log_run;
```

Untuk informasi selengkapnya, lihat Parameter untuk mengelola ekstensi pg_cron.

Menjadwalkan pekerjaan cron untuk basis data selain basis data default

Metadata untuk `pg_cron` semua disimpan dalam basis data default PostgreSQL bernama `postgres`. Karena pekerja latar belakang digunakan untuk menjalankan pekerjaan pemeliharaan cron, Anda dapat menjadwalkan pekerjaan di salah satu basis data Anda dalam instans DB PostgreSQL:

1. Dalam basis data cron, jadwalkan pekerjaan seperti yang biasa Anda lakukan menggunakan cron.schedule.

   ```
   postgres=> SELECT cron.schedule('database1 manual vacuum', '29 03 * * *', 'vacuum
    freeze test_table');
   ```

2. Sebagai pengguna dengan peran `rds_superuser`, perbarui kolom basis data untuk pekerjaan yang baru saja Anda buat agar berjalan di basis data lain dalam instans DB PostgreSQL Anda.

```
postgres=> UPDATE cron.job SET database = 'database1' WHERE jobid = 106;
```

3. Verifikasi dengan membuat kueri tabel `cron.job`.

```
postgres=> SELECT * FROM cron.job;
jobid | schedule    | command                          | nodename  | nodeport |
 database | username  | active | jobname
------+-------------+----------------------------------+-----------+----------
+----------+-----------+--------+-------------------------
106   | 29 03 * * * | vacuum freeze test_table         | localhost | 8192     |
 database1| adminuser | t      | database1 manual vacuum
  1   | 59 23 * * * | vacuum freeze pgbench_accounts | localhost | 8192     |
 postgres | adminuser | t      | manual vacuum
(2 rows)
```

> **Note**
>
> Dalam beberapa situasi, Anda mungkin menambahkan pekerjaan cron yang ingin Anda jalankan di basis data yang berbeda. Dalam kasus tersebut, pekerjaan mungkin mencoba untuk dijalankan dalam basis data default (`postgres`) sebelum Anda memperbarui kolom basis data yang benar. Jika nama pengguna memiliki izin, berarti pekerjaan berhasil dijalankan di basis data default.

Referensi untuk ekstensi pg_cron

Anda dapat menggunakan parameter, fungsi, dan tabel berikut dengan ekstensi `pg_cron`. Untuk informasi selengkapnya, lihat Apa itu pg_cron? dalam dokumentasi pg_cron.

Topik

- Parameter untuk mengelola ekstensi pg_cron
- Referensi fungsi: cron.schedule
- Referensi fungsi: cron.unschedule
- Tabel untuk menjadwalkan pekerjaan dan menangkap status

Parameter untuk mengelola ekstensi pg_cron

Berikut adalah daftar parameter yang mengontrol perilaku ekstensi `pg_cron`.

Parameter	Deskripsi
cron.database_name	Basis data tempat metadata `pg_cron` disimpan.
`cron.host`	Nama host untuk terhubung ke PostgreSQL. Anda tidak dapat mengubah nilai ini.
`cron.log_run`	Catat setiap pekerjaan yang berjalan di tabel `job_run_details` . Nilainya adalah on atau `off`. Untuk informasi selengkapnya, lihat Tabel untuk menjadwalkan pekerjaan dan menangkap status .
`cron.log_statement`	Catat semua pernyataan cron sebelum menjalankannya. Nilainya adalah on atau `off`.
`cron.max_running_jobs`	Jumlah maksimum pekerjaan yang dapat dijalankan secara bersamaan.
`cron.use_background_workers`	Gunakan pekerja latar belakang, bukan sesi klien. Anda tidak dapat mengubah nilai ini.

Gunakan perintah SQL berikut untuk menampilkan parameter ini dan nilainya.

```
postgres=> SELECT name, setting, short_desc FROM pg_settings WHERE name LIKE 'cron.%'
 ORDER BY name;
```

Referensi fungsi: cron.schedule

Fungsi ini menjadwalkan pekerjaan cron. Pada mulanya, pekerjaan dijadwalkan di basis data `postgres` default. Fungsi tersebut menampilkan nilai `bigint` yang mewakili ID pekerjaan. Untuk menjadwalkan pekerjaan agar berjalan di basis data lain dalam instans DB PostgreSQL Anda, lihat contohnya di Menjadwalkan pekerjaan cron untuk basis data selain basis data default.

Fungsi ini memiliki dua format sintaks.

Sintaksis

```
cron.schedule (job_name,
    schedule,
    command
);

cron.schedule (schedule,
    command
);
```

Parameter

Parameter	Deskripsi
`job_name`	Nama pekerjaan cron.
`schedule`	Teks yang menunjukkan jadwal untuk pekerjaan cron. Formatnya adalah format cron standar.
`command`	Teks perintah yang akan dijalankan.

Contoh-contoh

```
postgres=> SELECT cron.schedule ('test','0 10 * * *', 'VACUUM pgbench_history');
 schedule
----------
      145
(1 row)

postgres=> SELECT cron.schedule ('0 15 * * *', 'VACUUM pgbench_accounts');
 schedule
----------
      146
(1 row)
```

Referensi fungsi: cron.unschedule

Fungsi ini menghapus pekerjaan cron. Anda dapat menentukan `job_name` atau `job_id`. Suatu kebijakan memastikan bahwa Anda adalah pemilik guna menghapus jadwal untuk pekerjaan. Fungsi ini menampilkan Boolean yang menunjukkan keberhasilan atau kegagalan.

Fungsi tersebut memiliki format sintaks berikut.

Sintaksis

```
cron.unschedule (job_id);

cron.unschedule (job_name);
```

Parameter

Parameter	Deskripsi
`job_id`	ID pekerjaan yang ditampilkan dari fungsi `cron.schedule` saat pekerjaan cron dijadwalkan.
`job_name`	Nama pekerjaan cron yang dijadwalkan dengan fungsi `cron.schedule` .

Contoh-contoh

```
postgres=> SELECT cron.unschedule(108);
 unschedule
------------
 t
(1 row)

postgres=> SELECT cron.unschedule('test');
 unschedule
------------
 t
(1 row)
```

Tabel untuk menjadwalkan pekerjaan dan menangkap status

Tabel berikut digunakan untuk menjadwalkan pekerjaan cron dan mencatat bagaimana pekerjaan tersebut diselesaikan.

Tabel	Deskripsi
`cron.job`	Berisi metadata tentang setiap pekerjaan yang dijadwalkan. Sebagian besar interaksi dengan tabel ini akan dilakukan menggunakan fungsi `cron.schedule` dan `cron.unschedule`. **Important** Sebaiknya Anda tidak memberikan pembaruan atau memasukkan hak istimewa secara langsung ke tabel ini. Dengan begitu, pengguna dapat memperbarui kolom `username` agar berjalan sebagai `rds-superuser`.
`cron.job_run_details`	Berisi informasi historis tentang pekerjaan terjadwal terdahulu yang dijalankan. Hal ini berguna untuk menyelidiki status, pesan kembali, dan waktu mulai dan akhir dari pekerjaan yang berjalan. **Note** Untuk mencegah tabel ini berkembang tanpa batas waktu, bersihkan secara teratur. Sebagai contoh, lihat Membersihkan tabel riwayat pg_cron.

Menggunakan pgAudit untuk membuat log aktivitas basis data

Lembaga keuangan, lembaga pemerintah, dan banyak industri perlu menyimpan log audit untuk memenuhi persyaratan peraturan. Dengan menggunakan ekstensi PostgreSQL Audit (pgAudit) dengan klastier DB Aurora PostgreSQL, Anda dapat menangkap catatan terperinci yang biasanya dibutuhkan oleh auditor atau untuk memenuhi persyaratan peraturan. Misalnya, Anda dapat

mengatur ekstensi pgAudit untuk melacak perubahan yang dibuat pada basis data dan tabel tertentu, untuk merekam pengguna yang membuat perubahan, dan banyak detail lainnya.

Ekstensi pgAudit dibangun di atas fungsionalitas infrastruktur pencatatan log PostgreSQL asli dengan memperluas pesan log dengan lebih detail. Dengan kata lain, Anda menggunakan pendekatan yang sama untuk melihat log audit Anda seperti yang Anda lakukan untuk melihat pesan log apa pun. Untuk informasi selengkapnya tentang pencatatan log PostgreSQL, lihat File log basis data Aurora PostgreSQL.

Ekstensi pgAudit menyunting data sensitif seperti kata sandi cleartext dari log. Jika klaster DB Aurora PostgreSQL Anda dikonfigurasi untuk mencatat pernyataan bahasa manipulasi data (DHTML) seperti yang dijelaskan dalam Mengaktifkan pengelogan kueri untuk klaster DB Aurora PostgreSQL, Anda dapat menghindari masalah kata sandi cleartext dengan menggunakan ekstensi PostgreSQL Audit.

Anda dapat mengonfigurasi audit pada basis data instans Anda dengan tingkat kekhususan yang tinggi. Anda dapat mengaudit semua basis data dan semua pengguna. Atau, Anda dapat memilih untuk mengaudit hanya basis data tertentu, pengguna, dan objek lainnya. Anda juga dapat secara eksplisit mengecualikan pengguna dan basis data tertentu agar tidak diaudit. Untuk informasi selengkapnya, lihat Mengecualikan pengguna atau basis data dari pencatatan log audit.

Mengingat jumlah detail yang dapat ditangkap, kami menyarankan jika Anda menggunakan pgAudit, Anda memantau konsumsi penyimpanan Anda.

Ekstensi pgAudit didukung pada semua versi Aurora PostgreSQL yang tersedia. Untuk daftar versi pgAudit yang didukung oleh versi Aurora PostgreSQL, lihat versi ekstensi untuk Amazon Aurora PostgreSQL di Release Notes for Aurora PostgreSQL.

Topik

- Menyiapkan ekstensi pgAdit
- Audit objek basis data
- Mengecualikan pengguna atau basis data dari pencatatan log audit
- Referensi untuk ekstensi pgAudit

Menyiapkan ekstensi pgAdit

Untuk menyiapkan ekstensi pgAudit pada klaster DB Aurora PostgreSQL, Anda terlebih dahulu menambahkan pgAudit ke pustaka bersama pada grup parameter klaster DB khusus untuk klaster

DB Aurora PostgreSQL Anda. Untuk informasi cara membuat grup grup parameter klaster DB, lihat Bekerja dengan grup parameter. Selanjutnya, Anda menginstal ekstensi pgAudit. Terakhir, Anda menentukan basis data dan objek yang ingin Anda audit. Prosedur di bagian ini menunjukkan caranya kepada Anda. Anda dapat menggunakan AWS Management Console atau AWS CLI.

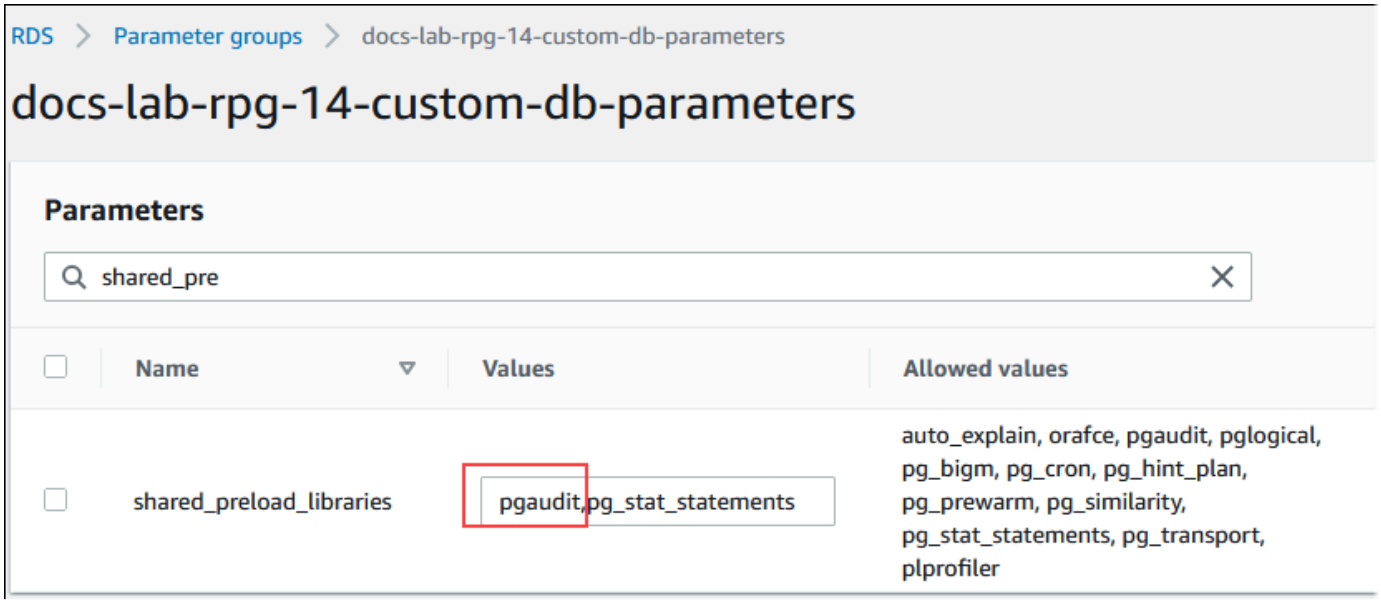
Anda harus memiliki izin sebagai peran `rds_superuser` untuk melakukan semua tugas ini.

Langkah-langkah berikut mengasumsikan bahwa klaster DB Aurora PostgreSQL Anda dikaitkan dengan grup parameter kluster DB khusus.

Konsol

Mengatur ekstensi pgAudit

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Instans penulis klaster DB Aurora PostgreSQL Anda.
3. Buka tab Konfigurasi untuk instans penulis klaster DB Aurora PostgreSQL. Di antara detail Instans, temukan tautan Grup parameter.
4. Pilih tautan untuk membuka parameter kustom yang terkait dengan klaster DB Aurora PostgreSQL.
5. Di kolom pencarian Parameter, ketik `shared_pre` untuk menemukan parameter `shared_preload_libraries`.
6. Pilih Edit parameter untuk mengakses nilai properti.
7. Tambahkan `pgaudit` ke daftar di kolom Nilai. Gunakan koma untuk memisahkan item dalam daftar nilai.



8. Boot ulang instans penulis klaster DB Aurora PostgreSQL Anda sehingga perubahan Anda pada parameter `shared_preload_libraries` berlaku.
9. Ketika instans tersedia, verifikasi bahwa pgAudit yang telah diinisialisasi. Gunakan `psql` untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL, kemudian jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

10. Dengan pgAudit yang diinisialisasi, Anda sekarang dapat membuat ekstensi. Anda perlu membuat ekstensi setelah menginisialisasi pustaka karena ekstensi `pgaudit` menginstal pemicu peristiwa untuk mengaudit pernyataan bahasa definisi data (DDL).

```
CREATE EXTENSION pgaudit;
```

11. Tutup sesi `psql`.

```
labdb=> \q
```

12. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
13. Temukan parameter `pgaudit.log` dalam daftar lalu atur ke nilai yang sesuai untuk kasus penggunaan Anda. Misalnya, menyetel parameter `pgaudit.log` ke `write` seperti yang

ditunjukkan pada gambar berikut menangkap sisipan, pembaruan, penghapusan, dan beberapa jenis lainnya yang berubah pada log.

Anda juga dapat memilih salah satu nilai berikut untuk parameter `pgaudit.log`.

- none – Ini adalah default. Tidak ada perubahan basis data yang dibuat log.
- semua – Semua dibuat log (baca, tulis, fungsi, peran, ddl, lain-lain).
- ddl – Membuat log semua pernyataan bahasa definisi data (DDL) yang tidak disertakan dalam kelas `ROLE`.
- fungsi – Membuat log panggilan fungsi dan blok `DO`.
- lain-lain – Membuat log berbagai perintah, seperti `DISCARD`, `FETCH`, `CHECKPOINT`, `VACUUM`, dan `SET`.
- baca – Membuat log `SELECT` dan `COPY` saat sumbernya adalah relasi (seperti tabel) atau kueri.
- peran – Membuat log pernyataan yang terkait dengan peran dan hak istimewa, seperti `GRANT`, `REVOKE`, `CREATE ROLE`, `ALTER ROLE`, dan `DROP ROLE`.
- write – Membuat log `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, dan `COPY` ketika tujuan adalah relasi (tabel).

14. Pilih Simpan perubahan
15. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
16. Pilih instans penulis klaster DB Aurora PostgreSQL dari daftar Basis Data untuk memilihnya, lalu pilih Boot ulang dari menu Tindakan.

AWS CLI

Menyiapkan pgAudit

Untuk mengatur PGAudit menggunakanAWS CLI, Anda memanggil modify-db-parameter-groupoperasi untuk mengubah parameter log audit di grup parameter kustom Anda, seperti yang ditunjukkan dalam prosedur berikut.

1. Gunakan perintah AWS CLI berikut untuk menambah `pgaudit` ke parameter `shared_preload_libraries`.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pgaudit,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Gunakan perintah AWS CLI berikut untuk melakukan boot ulang instans penulis klaster DB Aurora PostgreSQL sehingga pustaka pgaudit dapat diinisialisasi.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. Ketika instans tersedia, Anda dapat memverifikasi bahwa `pgaudit` telah diinisialisasi. Gunakan `psql` untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL, kemudian jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pgaudit
(1 row)
```

   Dengan pgAudit yang diinisialisasi, Anda sekarang dapat membuat ekstensi.

```
CREATE EXTENSION pgaudit;
```

4. Tutup sesi `psql` sehingga Anda dapat menggunakan AWS CLI.

```
labdb=> \q
```

5. Gunakan perintah AWS CLI berikut untuk menentukan kelas pernyataan yang ingin dibuat log oleh sesi audit pencatatan log. Contoh menetapkan parameter `pgaudit.log` ke `write`, yang menangkap sisipan, pembaruan, dan penghapusan ke log.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.log,ParameterValue=write,ApplyMethod=pending-reboot" \
   --region aws-region
```

Anda juga dapat memilih salah satu nilai berikut untuk parameter `pgaudit.log`.

- none – Ini adalah default. Tidak ada perubahan basis data yang dibuat log.
- semua – Semua dibuat log (baca, tulis, fungsi, peran, ddl, lain-lain).
- ddl – Membuat log semua pernyataan bahasa definisi data (DDL) yang tidak disertakan dalam kelas `ROLE`.
- fungsi – Membuat log panggilan fungsi dan blok `DO`.
- lain-lain – Membuat log berbagai perintah, seperti `DISCARD`, `FETCH`, `CHECKPOINT`, `VACUUM`, dan `SET`.
- baca – Membuat log `SELECT` dan `COPY` saat sumbernya adalah relasi (seperti tabel) atau kueri.
- peran – Membuat log pernyataan yang terkait dengan peran dan hak istimewa, seperti `GRANT`, `REVOKE`, `CREATE ROLE`, `ALTER ROLE`, dan `DROP ROLE`.
- write – Membuat log `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, dan `COPY` ketika tujuan adalah relasi (tabel).

Boot ulang instans penulis Aurora PostgreSQL DB klaster instans PostgreSQL DB menggunakan perintah berikut. AWS CLI

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

Audit objek basis data

Dengan pgAudit yang telah disiapkan di klaster DB Aurora PostgreSQL Anda dan dikonfigurasi untuk kebutuhan Anda, informasi lebih detail akan ditangkap dalam pembuatan log PostgreSQL. Misalnya, sementara konfigurasi pencatatan log PostgreSQL default mengidentifikasi tanggal dan waktu perubahan dibuat dalam tabel basis data, dengan ekstensi pgAudit entri log yang dapat menyertakan skema, pengguna yang membuat perubahan, dan detail lainnya tergantung pada bagaimana parameter ekstensi dikonfigurasi. Anda dapat menyiapkan audit untuk melacak perubahan dengan cara berikut ini.

- Untuk setiap sesi, oleh pengguna. Untuk tingkat sesi, Anda dapat menangkap teks perintah yang sepenuhnya memenuhi syarat.
- Untuk setiap objek, oleh pengguna dan basis data.

Kemampuan audit objek diaktifkan saat Anda membuat peran `rds_pgaudit` pada sistem Anda lalu akan menambahkan peran ini ke parameter `pgaudit.role` dalam grup parameter parameter khusus Anda. Secara default, parameter `pgaudit.role` tidak disetel dan satu-satunya nilai yang diizinkan adalah `rds_pgaudit`. Langkah-langkah berikut mengasumsikan bahwa `pgaudit` telah diinisialisasi dan bahwa Anda telah membuat ekstensi `pgaudit` dengan mengikuti prosedur di Menyiapkan ekstensi pgAdit.

```
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: statement: SELECT feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: AUDIT: SESSION,2,1,READ,SELECT,TABLE,public.support,"SELECT
feedback, s.sentiment,s.confidence
FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
ORDER BY s.confidence DESC;",<none>
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:LOG: QUERY STATISTICS
2022-10-07 23:36:51 UTC:52.95.4.10(14410):postgres@labdb:[1374]:DETAIL: ! system usage stats:
! 0.009494 s user, 0.007442 s system, 0.141985 s elapsed
! [0.022327 s user, 0.007442 s system total]
```

Seperti yang ditunjukkan dalam contoh ini, baris "LOG: AUDIT: SESSION" memberikan informasi tabel dan skemanya, di antara detail lainnya.

Menyiapkan audit objek

1. Gunakan `psql` untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL.

   ```
   psql --host=your-instance-name.aws-region.rds.amazonaws.com --port=5432 --
   username=postgrespostgres --password --dbname=labdb
   ```

2. Buat peran basis data yang dinamai `rds_pgaudit` dengan menggunakan perintah berikut.

```
labdb=> CREATE ROLE rds_pgaudit;
CREATE ROLE
labdb=>
```

3. Tutup sesi `psql`.

```
labdb=> \q
```

Dalam beberapa langkah berikutnya, gunakan AWS CLI untuk memodifikasi parameter log audit di grup parameter khusus Anda.

4. Gunakan perintah AWS CLI berikut untuk mengatur parameter `pgaudit.role` ke `rds_pgaudit`. Secara default, parameter ini kosong, dan `rds_pgaudit` merupakan satu-satunya nilai yang diizinkan.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=pgaudit.role,ParameterValue=rds_pgaudit,ApplyMethod=pending-reboot"
 \
   --region aws-region
```

5. Gunakan perintah AWS CLI berikut untuk melakukan boot ulang instans penulis klaster DB Aurora PostgreSQL sehingga perubahan Anda pada parameter berlaku.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

6. Jalankan perintah berikut untuk mengonfirmasi bahwa `pgaudit.role` diatur ke`rds_pgaudit`.

```
SHOW pgaudit.role;
pgaudit.role
------------------
rds_pgaudit
```

Untuk menguji pencatatan log pgAudit, Anda dapat menjalankan beberapa contoh perintah yang ingin Anda audit. Misalnya, Anda dapat menjalankan perintah berikut.

```
CREATE TABLE t1 (id int);
GRANT SELECT ON t1 TO rds_pgaudit;
SELECT * FROM t1;
id
----
(0 rows)
```

Log basis data harus berisi entri yang serupa dengan entri berikut.

```
...
2017-06-12 19:09:49 UTC:...:rds_test@postgres:[11701]:LOG: AUDIT:
OBJECT,1,1,READ,SELECT,TABLE,public.t1,select * from t1;
...
```

Untuk informasi tentang melihat log, lihat Memantau file log Amazon Aurora.

Untuk mempelajari lebih lanjut tentang ekstensi PGAudit, lihat PGAudit aktif. GitHub

Mengecualikan pengguna atau basis data dari pencatatan log audit

Seperti dibahas dalam File log basis data Aurora PostgreSQL, log PostgreSQL menghabiskan ruang penyimpanan. Menggunakan ekstensi pgAudit dapat menambah volume data yang dikumpulkan di log Anda ke berbagai tingkat, tergantung pada perubahan yang Anda lacak. Anda mungkin tidak perlu mengaudit setiap pengguna atau basis data di klaster DB Aurora PostgreSQL Anda.

Untuk meminimalkan dampak pada penyimpanan Anda dan untuk menghindari pengambilan catatan audit yang tidak perlu, Anda dapat mengecualikan pengguna dan basis data agar tidak diaudit. Anda juga dapat mengubah pencatatan log dalam sesi tertentu. Contoh berikut menunjukkan caranya kepada Anda.

> **Note**
>
> Pengaturan parameter pada tingkat sesi lebih diutamakan daripada pengaturan dalam grup parameter klaster DB khusus untuk instans penulis klaster DB Aurora PostgreSQL. Jika Anda tidak ingin pengguna basis data melewati pengaturan konfigurasi pencatatan audit Anda, pastikan untuk mengubah izin mereka.

Misalkan klaster DB Aurora PostgreSQL Anda dikonfigurasi untuk mengaudit tingkat aktivitas yang sama untuk semua pengguna dan basis data. Anda kemudian memutuskan bahwa Anda tidak ingin mengaudit pengguna `myuser`. Anda dapat mematikan audit `myuser` dengan perintah SQL berikut.

```
ALTER USER myuser SET pgaudit.log TO 'NONE';
```

Kemudian, Anda dapat menggunakan kueri berikut untuk memeriksa kolom `user_specific_settings` untuk `pgaudit.log` agar mengonfirmasi bahwa parameter diatur ke `NONE`.

```
SELECT
    usename AS user_name,
    useconfig AS user_specific_settings
FROM
    pg_user
WHERE
    usename = 'myuser';
```

Anda akan melihat output seperti berikut ini.

```
 user_name | user_specific_settings
-----------+------------------------
 myuser    | {pgaudit.log=NONE}
(1 row)
```

Anda dapat mematikan pencatatan log untuk pengguna tertentu di tengah-tengah sesi mereka dengan basis data menggunakan perintah berikut.

```
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'none';
```

Gunakan kueri berikut untuk memeriksa kolom pengaturan untuk pgaudit.log untuk kombinasi pengguna dan basis data tertentu.

```
SELECT
    usename AS "user_name",
    datname AS "database_name",
    pg_catalog.array_to_string(setconfig, E'\n') AS "settings"
FROM
    pg_catalog.pg_db_role_setting s
    LEFT JOIN pg_catalog.pg_database d ON d.oid = setdatabase
```

```
    LEFT JOIN pg_catalog.pg_user r ON r.usesysid = setrole
WHERE
    usename = 'myuser'
    AND datname = 'mydatabase'
ORDER BY
    1,
    2;
```

Anda akan melihat output yang mirip dengan berikut ini.

```
  user_name | database_name |     settings
-----------+---------------+------------------
 myuser    | mydatabase    | pgaudit.log=none
(1 row)
```

Setelah menonaktifkan audit `myuser`, Anda memutuskan bahwa Anda tidak ingin melacak perubahan ke `mydatabase`. Anda mematikan audit untuk basis data spesifik tersebut menggunakan perintah berikut.

```
ALTER DATABASE mydatabase SET pgaudit.log to 'NONE';
```

Kemudian, gunakan kueri berikut untuk memeriksa kolom database_specific_settings untuk mengonfirmasi bahwa pgaudit.log disetel ke NONE.

```
SELECT
a.datname AS database_name,
b.setconfig AS database_specific_settings
FROM
pg_database a
FULL JOIN pg_db_role_setting b ON a.oid = b.setdatabase
WHERE
a.datname = 'mydatabase';
```

Anda akan melihat output seperti berikut ini.

```
 database_name | database_specific_settings
---------------+----------------------------
 mydatabase    | {pgaudit.log=NONE}
(1 row)
```

Untuk mengembalikan pengaturan ke pengaturan default untuk myuser, gunakan perintah berikut:

```
ALTER USER myuser RESET pgaudit.log;
```

Untuk mengembalikan pengaturan ke pengaturan default untuk basis data, gunakan perintah berikut ini.

```
ALTER DATABASE mydatabase RESET pgaudit.log;
```

Untuk mengatur ulang pengguna dan basis data ke pengaturan default, gunakan perintah berikut.

```
ALTER USER myuser IN DATABASE mydatabase RESET pgaudit.log;
```

Anda juga dapat menangkap peristiwa tertentu ke log dengan menyetel `pgaudit.log` ke salah satu nilai lain yang diizinkan untuk parameter `pgaudit.log`. Untuk informasi selengkapnya, lihat Daftar pengaturan yang diizinkan untuk parameter `pgaudit.log`.

```
ALTER USER myuser SET pgaudit.log TO 'read';
ALTER DATABASE mydatabase SET pgaudit.log TO 'function';
ALTER USER myuser IN DATABASE mydatabase SET pgaudit.log TO 'read,function'
```

Referensi untuk ekstensi pgAudit

Anda dapat menentukan tingkat detail yang Anda inginkan untuk log audit Anda dengan mengubah satu atau beberapa parameter yang tercantum di bagian ini.

Mengatur perilaku pgAudit

Anda dapat mengontrol pencatatan audit dengan mengubah satu atau beberapa parameter yang tercantum dalam tabel berikut.

Parameter	Deskripsi
`pgaudit.log`	Menentukan kelas pernyataan yang akan dicatat oleh sesi audit pencatatan log. Nilai yang diizinkan termasuk ddl, fungsi, misc, baca, peran, tulis, tidak ada, semua. Untuk informasi selengkap nya, lihat Daftar pengaturan yang diizinkan untuk parameter `pgaudit.log` .
`pgaudit.log_catalog`	Saat diaktifkan (disetel ke 1), tambahkan pernyataan ke jejak audit jika semua relasi dalam pernyataan ada di pg_catalog.

Parameter	Deskripsi
`pgaudit.log_level`	Menentukan tingkat log untuk digunakan untuk entri log. Nilai yang diizinkan: debug5, debug4, debug3, debug2, debug1, info, notifikasi, peringatan, log
`pgaudit.log_parameter`	Ketika diaktifkan (diatur ke 1), parameter yang diteruskan dengan pernyataan ditangkap dalam log audit.
`pgaudit.log_relation`	Saat diaktifkan (disetel ke 1), log audit untuk sesi membuat entri log terpisah untuk setiap relasi (TABEL, TAMPILAN, dan sebagainya) yang direferensikan dalam pernyataan SELECT atau DML.
`pgaudit.log_statement_once`	Menentukan apakah pencatatan log akan mencakup teks pernyataan dan parameter dengan entri log pertama untuk kombinasi pernyataan/subpernyataan atau dengan setiap entri.
`pgaudit.role`	Menentukan peran utama yang akan digunakan untuk pencatatan log audit objek. Satu-satunya entri yang diizinkan adalah `rds_pgaudit` .

Daftar pengaturan yang diizinkan untuk parameter **`pgaudit.log`**

Nilai	Deskripsi
tidak ada	Ini adalah default. Tidak ada perubahan basis data yang dibuat log.
semua	Semuanya dibuat log (baca, tulis, fungsi, peran, ddl, misc).
ddl	Membuat log semua pernyataan bahasa definisi data (DDL) yang tidak disertakan dalam kelas `ROLE`.
fungsi	Log berfungsi panggilan dan blok `DO`.
misc	Log berbagai perintah, seperti `DISCARD`, `FETCH`, `CHECKPOINT` , `VACUUM` dan `SET`.

Nilai	Deskripsi
baca	Log `SELECT` dan `COPY` ketika sumbernya adalah relasi (seperti tabel) atau kueri.
peran	Log pernyataan yang terkait dengan peran dan hak istimewa, seperti `GRANT`, `REVOKE`, `CREATE ROLE`, `ALTER ROLE`, dan `DROP ROLE`.
tulis	Log `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, dan `COPY` ketika tujuan adalah relasi (tabel).

Untuk mencatat beberapa jenis peristiwa dengan audit sesi, gunakan daftar yang dipisahkan koma. Untuk membuat log semua jenis acara, atur `pgaudit.log` ke `ALL`. Boot ulang instans DB Anda untuk menerapkan perubahan.

Dengan objek audit, Anda dapat memperbaiki pencatatan log audit untuk bekerja dengan relasi spesifik. Misalnya, Anda dapat menentukan bahwa Anda ingin pencatatan log audit untuk operasi `READ` di satu tabel atau beberapa.

Menggunakan pglogical untuk menyinkronkan data di seluruh instans

Semua versi Aurora PostgreSQL yang tersedia saat ini mendukung ekstensi `pglogical`. Ekstensi pglogical mendahului fitur replikasi logis yang mirip secara fungsional yang diperkenalkan oleh PostgreSQL di versi 10. Untuk informasi selengkapnya, lihat Menggunakan replikasi logis PostgreSQL dengan Aurora.

Ekstensi `pglogical` ini mendukung replikasi logis antara dua atau lebih klaster DB Aurora PostgreSQL. Ini juga mendukung replikasi antara versi PostgreSQL yang berbeda, dan antara basis data yang berjalan pada instans RDS for PostgreSQL DB dan klaster DB Aurora PostgreSQL. Ekstensi `pglogical` menggunakan model berlangganan penerbitan untuk mereplikasi perubahan pada tabel dan objek lain, seperti urutan, dari penerbit ke pelanggan. Itu bergantung pada slot replikasi untuk memastikan bahwa perubahan disinkronkan dari simpul penerbit ke simpul pelanggan, didefinisikan sebagai berikut.

- Simpul penerbit adalah klaster DB Aurora PostgreSQL yang merupakan sumber data yang akan direplikas ke simpul lain. Simpul penerbit mendefinisikan tabel yang akan direplikasi dalam kumpulan publikasi.

- Simpul pelanggan adalah klaster DB Aurora PostgreSQL yang menerima pembaruan WAL dari penerbit. Pelanggan membuat langganan untuk terhubung ke penerbit dan mendapatkan data WAL yang diterjemahkan. Ketika pelanggan membuat langganan, slot replikasi dibuat pada simpul penerbit.

Berikut ini, Anda dapat menemukan informasi tentang cara mengatur ekstensi `pglogical`.

Topik

- Persyaratan dan batasan untuk ekstensi pglogis
- Menyiapkan ekstensi pglogical
- Menyiapkan replikasi logis untuk klaster DB Aurora PostgreSQL
- Membangun kembali replikasi logis setelah peningkatan besar
- Mengelola slot replikasi logis untuk Aurora PostgreSQL
- Referensi parameter untuk ekstensi pglogical

Persyaratan dan batasan untuk ekstensi pglogis

Semua rilis Aurora PostgreSQL yang tersedia saat ini untuk mendukung ekstensi `pglogical`.

Baik simpul penerbit maupun simpul pelanggan harus disiapkan untuk replikasi logis.

Tabel yang ingin Anda replikasi dari pelanggan ke penerbit harus memiliki nama yang sama dan skema yang sama. Tabel ini juga harus berisi kolom yang sama, dan kolom harus menggunakan tipe data yang sama. Tabel penerbit dan pelanggan harus memiliki kunci primer yang sama. Kami menyarankan Anda hanya menggunakan PRIMARY KEY sebagai kendala unik.

Tabel pada simpul pelanggan dapat memiliki kendala yang lebih permisif daripada yang ada di simpul penerbit untuk kendala CHECK dan kendala NOT NULL.

Ekstensi `pglogical` ini menyediakan fitur seperti replikasi dua arah yang tidak didukung oleh fitur replikasi logis yang dibangun ke dalam PostgreSQL (versi 10 dan lebih tinggi). Untuk informasi lebih lanjut, lihat PostgreSQL bi-directional replication using pglogical.

Menyiapkan ekstensi pglogical

Untuk menyiapkan ekstensi `pglogical` pada klaster DB Aurora PostgreSQL, Anda menambahkan `pglogical` ke pustaka bersama pada grup parameter klaster DB khusus untuk klaster DB Aurora

PostgreSQL. Anda juga perlu mengatur nilai parameter `rds.logical_replication` ke `1`, untuk mengaktifkan penguraian kode logis. Akhirnya, Anda membuat ekstensi di basis data. Anda dapat menggunakan AWS Management Console atau AWS CLI untuk tugas-tugas ini.

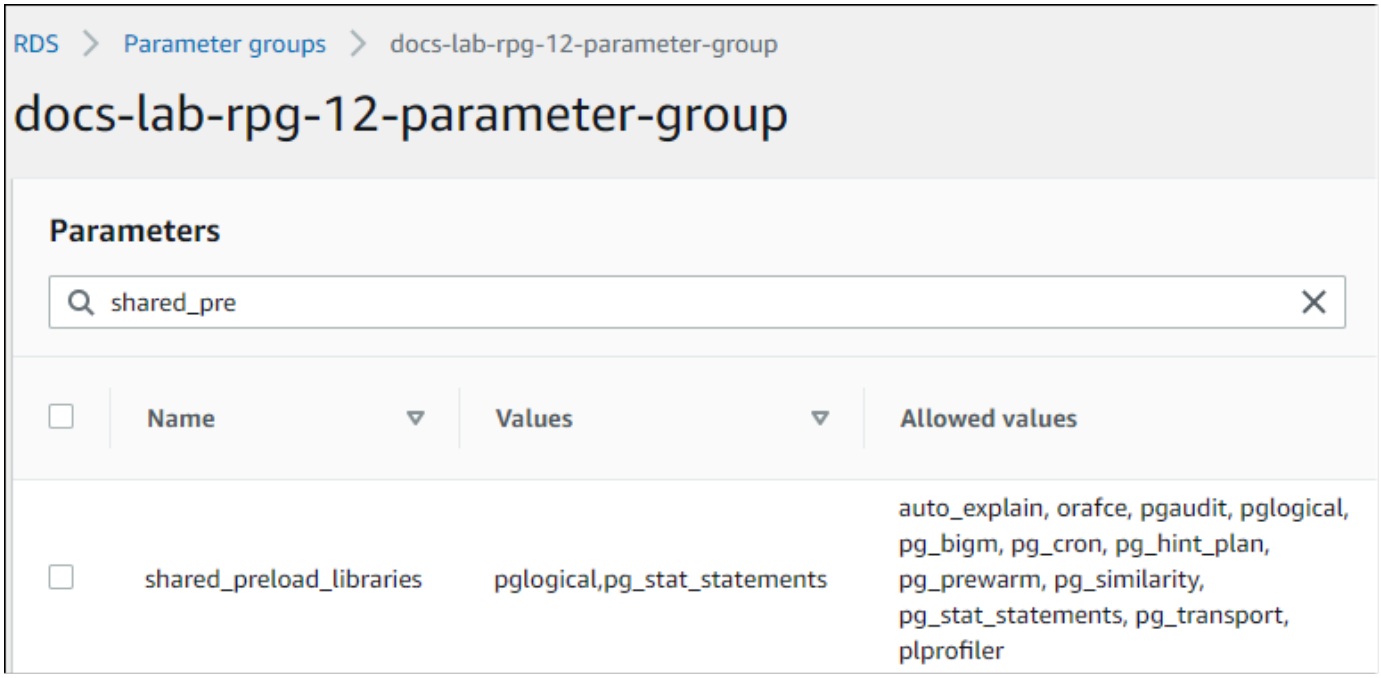
Anda harus memiliki izin sebagai peran `rds_superuser` untuk melakukan tugas-tugas ini.

Langkah-langkah berikut mengasumsikan bahwa klaster DB Aurora PostgreSQL Anda dikaitkan dengan grup parameter kluster DB khusus. Untuk informasi cara membuat grup grup parameter klaster DB, lihat Bekerja dengan grup parameter.

Konsol

Menyiapkan ekstensi pglogical

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Instans penulis klaster DB Aurora PostgreSQL Anda.
3. Buka tab Konfigurasi untuk instans penulis klaster DB Aurora PostgreSQL. Di antara detail Instans, temukan tautan Grup parameter.
4. Pilih tautan untuk membuka parameter kustom yang terkait dengan klaster DB Aurora PostgreSQL.
5. Di kolom pencarian Parameter, ketik `shared_pre` untuk menemukan parameter `shared_preload_libraries`.
6. Pilih Edit parameter untuk mengakses nilai properti.
7. Tambahkan `pglogical` ke daftar di kolom Nilai. Gunakan koma untuk memisahkan item dalam daftar nilai.



8. Temukan parameter `rds.logical_replication` dan atur ke 1, untuk mengaktifkan replikasi logis.
9. Boot ulang instans penulis klaster DB Aurora PostgreSQL sehingga perubahan Anda akan berlaku.
10. Ketika instans tersedia, Anda dapat menggunakan `psql` (atau pgAdmin) untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL Anda.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

11. Untuk memverifikasi bahwa pglogical diinisialisasi, jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pglogical
(1 row)
```

12. Verifikasi pengaturan yang memungkinkan penguraian kode logis, sebagai berikut.

```
SHOW wal_level;
wal_level
-----------
 logical
```

```
(1 row)
```

13. Buat ekstensi, sebagai berikut.

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

14. Pilih Simpan perubahan
15. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
16. Pilih instans penulis klaster DB Aurora PostgreSQL dari daftar Basis Data untuk memilihnya, lalu pilih Boot ulang dari menu Tindakan.

AWS CLI

Menyiapkan ekstensi pglogical

Untuk mengatur pglogical menggunakanAWS CLI, Anda memanggil modify-db-parameter-groupoperasi untuk memodifikasi parameter tertentu dalam grup parameter kustom Anda seperti yang ditunjukkan dalam prosedur berikut.

1. Gunakan perintah AWS CLI berikut untuk menambah `pglogical` ke parameter `shared_preload_libraries`.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pglogical,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Gunakan perintah AWS CLI berikut untuk mengatur `rds.logical_replication` ke 1 untuk mengaktifkan kemampuan penguraian kode logis untuk instans penulis dari klaster DB Aurora PostgreSQL.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=rds.logical_replication,ParameterValue=1,ApplyMethod=pending-
reboot" \
   --region aws-region
```

3. Gunakan AWS CLI perintah berikut untuk melakukan boot ulang instans penulis klaster DB Aurora PostgreSQL sehingga pustaka pglogical dapat diinisialisasi.

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

4. Ketika instans tersedia, gunakan `psql` untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL Anda.

```
psql --host=111122223333.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

5. Buat ekstensi, sebagai berikut.

```
CREATE EXTENSION pglogical;
EXTENSION CREATED
```

6. Boot ulang instans penulis Aurora PostgreSQL DB klaster instans PostgreSQL DB menggunakan perintah berikut. AWS CLI

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

Menyiapkan replikasi logis untuk klaster DB Aurora PostgreSQL

Prosedur berikut menunjukkan cara memulai replikasi logis antara dua klaster DB Aurora PostgreSQL. Langkah-langkah mengasumsikan bahwa sumber (penerbit) dan target (pelanggan) memiliki ekstensi `pglogical` yang disiapkan seperti yang dijelaskan dalam Menyiapkan ekstensi pglogical.

Untuk membuat simpul penerbit dan menentukan tabel untuk direplikasi

Langkah-langkah ini mengasumsikan bahwa klaster DB Aurora PostgreSQL memiliki instans penulis dengan basis data yang memiliki satu atau beberapa tabel yang ingin direplikasi ke simpul lain. Anda perlu membuat ulang struktur tabel dari penerbit pada pelanggan, jadi pertama-tama, dapatkan struktur tabel jika perlu. Anda dapat melakukannya dengan menggunakan `\d tablename`

metacommand `psql` dan kemudian membuat tabel yang sama pada instans pelanggan. Prosedur berikut membuat tabel contoh pada penerbit (sumber) untuk tujuan demonstrasi.

1. Gunakan `psql` untuk terhubung ke instans yang memiliki tabel yang ingin Anda gunakan sebagai sumber untuk pelanggan.

   ```
   psql --host=source-instance.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password --dbname=labdb
   ```

   Jika Anda tidak memiliki tabel yang ingin Anda tiru, Anda dapat membuat tabel contoh sebagai berikut.

   a. Buat contoh tabel menggunakan pernyataan SQL berikut.

      ```
      CREATE TABLE docs_lab_table (a int PRIMARY KEY);
      ```

   b. Mengisi tabel dengan data yang dihasilkan dengan menggunakan pernyataan SQL berikut.

      ```
      INSERT INTO docs_lab_table VALUES (generate_series(1,5000));
      INSERT 0 5000
      ```

   c. Verifikasi bahwa data ada dalam tabel dengan menggunakan pernyataan SQL berikut.

      ```
      SELECT count(*) FROM docs_lab_table;
      ```

2. Identifikasi klaster DB Aurora PostgreSQL ini sebagai simpul penerbit, sebagai berikut.

   ```
   SELECT pglogical.create_node(
       node_name := 'docs_lab_provider',
       dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
    dbname=labdb');
    create_node
   -------------
      3410995529
   (1 row)
   ```

3. Tambahkan tabel yang ingin Anda replikasi ke set replikasi default. Untuk informasi set replikasi selengkapnya, lihat Replication sets dalam dokumentasi pglogical.

   ```
   SELECT pglogical.replication_set_add_table('default', 'docs_lab_table', 'true',
    NULL, NULL);
   ```

```
 replication_set_add_table
  ---------------------------
  t
  (1 row)
```

Penyiapan simpul penerbit selesai. Anda sekarang dapat menyiapkan simpul pelanggan untuk menerima pembaruan dari penerbit.

Untuk menyiapkan simpul pelanggan dan membuat langganan untuk menerima pembaruan

Langkah-langkah ini mengasumsikan bahwa klaster DB Aurora PostgreSQL telah disiapkan dengan ekstensi `pglogical`. Untuk informasi selengkapnya, lihat Menyiapkan ekstensi pglogical.

1. Gunakan `psql` untuk terhubung ke instans yang ingin Anda terima pembaruan dari penerbit.

```
psql --host=target-instance.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. Pada pelanggan klaster DB Aurora PostgreSQL, , buat tabel yang sama yang ada pada penerbit. Untuk contoh ini, tabelnya adalah `docs_lab_table`. Anda dapat membuat tabel sebagai berikut.

```
CREATE TABLE docs_lab_table (a int PRIMARY KEY);
```

3. Verifikasi bahwa tabel ini kosong.

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  0
(1 row)
```

4. Identifikasi klaster DB Aurora PostgreSQL ini sebagai simpul pelanggan, sebagai berikut.

```
SELECT pglogical.create_node(
    node_name := 'docs_lab_target',
    dsn := 'host=target-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=********');
 create_node
-------------
   2182738256
```

```
(1 row)
```

5. Buat langganan.

```
SELECT pglogical.create_subscription(
   subscription_name := 'docs_lab_subscription',
   provider_dsn := 'host=source-instance.aws-region.rds.amazonaws.com port=5432
 sslmode=require dbname=labdb user=postgres password=*******',
   replication_sets := ARRAY['default'],
   synchronize_data := true,
   forward_origins := '{}' );
 create_subscription
---------------------
1038357190
(1 row)
```

Ketika Anda menyelesaikan langkah ini, data dari tabel pada penerbit dibuat dalam tabel pada pelanggan. Anda dapat memverifikasi bahwa ini telah terjadi dengan menggunakan query SQL berikut.

```
SELECT count(*) FROM docs_lab_table;
 count
-------
  5000
(1 row)
```

Dari titik ini ke depan, perubahan yang dilakukan pada tabel pada penerbit direplikasi ke tabel pada pelanggan.

Membangun kembali replikasi logis setelah peningkatan besar

Sebelum Anda dapat melakukan peningkatan versi utama dari klaster DB Aurora PostgreSQL yang disiapkan sebagai simpul penerbit untuk replikasi logis, Anda harus menghapus semua slot replikasi, bahkan yang tidak aktif. Kami menyarankan Anda untuk mengalihkan sementara transaksi basis data dari simpul penerbit, menghapus sementara slot replikasi, meningkatkan klaster Aurora PostgreSQL, dan kemudian membangun kembali dan memulai ulang replikasi.

Slot replikasi hanya dihosting di simpul penerbit. Simpul pelanggan Aurora PostgreSQL dalam skenario replikasi logis tidak memiliki slot untuk dihapus sementara. Proses meningkatkan versi utama Aurora PostgreSQL mendukung peningkatan pelanggan ke versi utama baru PostgreSQL

independen dari simpul penerbit. Namun, proses peningkatan memang mengganggu proses replikasi dan mengganggu sinkronisasi data WAL antara simpul penerbit dan simpul pelanggan. Anda perlu membangun kembali replikasi logis antara penerbit dan pelanggan setelah meningkatkan penerbit, pelanggan, atau keduanya. Prosedur berikut menunjukkan kepada Anda cara menentukan bahwa replikasi telah terganggu dan cara mengatasi masalah tersebut.

Menentukan bahwa replikasi logis telah terganggu

Anda dapat menentukan bahwa proses replikasi telah terganggu dengan menanyakan simpul penerbit atau simpul pelanggan, sebagai berikut.

Untuk memeriksa simpul penerbit

- Gunakan `psql` untuk terhubung ke simpul penerbit, dan kemudian kueri fungsi `pg_replication_slots`. Perhatikan nilai di kolom aktif. Biasanya, ini akan kembali `t` (benar), menunjukkan bahwa replikasi aktif. Jika kueri kembali `f` (salah), ini merupakan indikasi bahwa replikasi ke pelanggan telah berhenti.

```
SELECT slot_name,plugin,slot_type,active FROM pg_replication_slots;
                    slot_name              |      plugin      | slot_type | active
-------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_labcb4fa94_docs_lab3de412c | pglogical_output | logical   | f
(1 row)
```

Untuk memeriksa simpul pelanggan

Pada simpul pelanggan, Anda dapat memeriksa status replikasi dengan tiga cara berbeda.

- Lihatlah log PostgreSQL pada simpul pelanggan untuk menemukan pesan kegagalan. Log dapat mengidentifikasi kegagalan dengan pesan yang menyertakan kode keluar 1, seperti yang ditunjukkan berikut.

```
2022-07-06 16:17:03 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 14610) exited with exit code 1
2022-07-06 16:19:44 UTC::@:[7361]:LOG: background worker "pglogical apply
 16404:2880255011" (PID 21783) exited with exit code 1
```

- Kueri fungsi `pg_replication_origin`. Hubungkan ke basis data pada simpul pelanggan menggunakan `psql` dan query fungsi `pg_replication_origin`, sebagai berikut.

```
SELECT * FROM pg_replication_origin;
 roident | roname
---------+--------
(0 rows)
```

Kumpulan hasil kosong berarti replikasi telah terganggu. Umumnya, Anda akan melihat output seperti berikut ini.

```
  roident |                       roname
 ---------+-----------------------------------------------------
        1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
 (1 row)
```

- Kueri fungsi `pglogical.show_subscription_status` seperti yang ditunjukkan pada contoh berikut.

```
SELECT subscription_name,status,slot_name FROM pglogical.show_subscription_status();
     subscription_name | status |              slot_name
---====---------------+--------+-----------------------------------
 docs_lab_subscription | down   | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

Output ini menunjukkan bahwa replikasi telah terganggu. Statusnya adalah down. Biasanya, output menunjukkan status sebagai `replicating`.

Jika proses replikasi logis Anda telah terganggu, Anda dapat membangun kembali replikasi dengan mengikuti langkah-langkah ini.

Untuk membangun kembali replikasi logis antara simpul penerbit dan pelanggan

Untuk membangun kembali replikasi, pertama-tama Anda memutuskan sambungan pelanggan dari simpul penerbit dan kemudian membangun kembali langganan, seperti yang diuraikan dalam langkah-langkah ini.

1. Hubungkan ke simpul pelanggan menggunakan `psql` sebagai berikut.

```
psql --host=222222222222.aws-region.rds.amazonaws.com --port=5432 --
username=postgres --password --dbname=labdb
```

2. Nonaktifkan langganan dengan menggunakan fungsi `pglogical.alter_subscription_disable`.

```
SELECT pglogical.alter_subscription_disable('docs_lab_subscription',true);
 alter_subscription_disable
----------------------------
 t
(1 row)
```

3. Dapatkan identifier simpul penerbit dengan menanyakan `pg_replication_origin`, sebagai berikut.

```
SELECT * FROM pg_replication_origin;
 roident |               roname
---------+--------------------------------------
       1 | pgl_labdb_docs_labcb4fa94_docs_lab3de412c
(1 row)
```

4. Gunakan respons dari langkah sebelumnya dengan perintah `pg_replication_origin_create` untuk menetapkan pengenal yang dapat digunakan oleh langganan saat dibuat kembali.

```
SELECT pg_replication_origin_create('pgl_labdb_docs_labcb4fa94_docs_lab3de412c');
  pg_replication_origin_create
------------------------------
                             1
(1 row)
```

5. Nyalakan langganan dengan meneruskan namanya dengan status `true`, seperti yang ditunjukkan dalam contoh berikut.

```
SELECT pglogical.alter_subscription_enable('docs_lab_subscription',true);
  alter_subscription_enable
---------------------------
 t
(1 row)
```

Periksa status simpul. Statusnya harus `replicating` seperti yang ditunjukkan dalam contoh ini.

```
SELECT subscription_name,status,slot_name
```

```
  FROM pglogical.show_subscription_status();
            subscription_name |   status    |              slot_name
------------------------------+-------------+-------------------------------------
 docs_lab_subscription         | replicating |
 pgl_labdb_docs_lab98f517b_docs_lab3de412c
(1 row)
```

Periksa status slot replikasi pelanggan pada simpul penerbit. Kolom `active` slot harus kembali `t` (benar), menunjukkan bahwa replikasi telah dibuat kembali.

```
SELECT slot_name,plugin,slot_type,active
  FROM pg_replication_slots;
                   slot_name              |      plugin      | slot_type | active
------------------------------------------+------------------+-----------+--------
 pgl_labdb_docs_lab98f517b_docs_lab3de412c | pglogical_output | logical   | t
(1 row)
```

Mengelola slot replikasi logis untuk Aurora PostgreSQL

Sebelum Anda dapat melakukan peningkatan versi utama dari instans penulis klaster DB Aurora PostgreSQL yang disiapkan sebagai simpul penerbit untuk replikasi logis, Anda harus menghapus semua slot replikasi, bahkan yang tidak aktif. Proses pra-pemeriksaan peningkatan versi utama akan memberi tahu Anda bahwa peningkatan tidak dapat dilanjutkan sampai slot dihapuskan sementara.

Untuk mengidentifikasi slot replikasi yang dibuat menggunakan ekstensi `pglogical`, masuk ke setiap basis data dan dapatkan nama simpul. Saat Anda menanyakan simpul pelanggan, Anda mendapatkan penerbit dan simpul pelanggan dalam output, seperti yang ditunjukkan dalam contoh ini.

```
SELECT * FROM pglogical.node;
node_id   |     node_name
------------+-------------------
 2182738256 | docs_lab_target
 3410995529 | docs_lab_provider
(2 rows)
```

Anda bisa mendapatkan detail tentang langganan dengan kueri berikut.

```
SELECT sub_name,sub_slot_name,sub_target
  FROM pglogical.subscription;
```

```
 sub_name |         sub_slot_name          | sub_target
----------+--------------------------------+------------
  docs_lab_subscription     | pgl_labdb_docs_labcb4fa94_docs_lab3de412c | 2182738256
(1 row)
```

Anda sekarang dapat menghapus sementara langganan, sebagai berikut.

```
SELECT pglogical.drop_subscription(subscription_name := 'docs_lab_subscription');
 drop_subscription
-------------------
                 1
(1 row)
```

Setelah menghapus sementara langganan, Anda dapat menghapus simpul.

```
SELECT pglogical.drop_node(node_name := 'docs-lab-subscriber');
 drop_node
-----------
 t
(1 row)
```

Anda dapat memverifikasi bahwa simpul tidak ada lagi, sebagai berikut.

```
SELECT * FROM pglogical.node;
 node_id | node_name
---------+-----------
(0 rows)
```

Referensi parameter untuk ekstensi pglogical

Dalam tabel Anda dapat menemukan parameter yang terkait dengan ekstensi `pglogical`. Parameter seperti `pglogical.conflict_log_level` dan `pglogical.conflict_resolution` digunakan untuk menangani konflik pembaruan. Konflik dapat muncul ketika perubahan dilakukan secara lokal ke tabel yang sama yang berlangganan perubahan dari penerbit. Konflik juga dapat terjadi selama berbagai skenario, seperti replikasi dua arah atau ketika beberapa pelanggan mereplikasi dari penerbit yang sama. Untuk informasi lebih lanjut, lihat PostgreSQL bi-directional replication using pglogical.

Parameter	Deskripsi
pglogical.batch_inserts	Melakukan penyisipan batch jika memungkinkan. Tidak diatur secara default. Ubah ke '1' untuk menghidupkan, '0' untuk mematikan.
pglogical.conflict_log_level	Menetapkan tingkat log yang digunakan untuk mencatat log konflik yang diselesaikan. Nilai string yang didukung adalah debug5, debug4, debug3, debug2, debug1, info, notice, warning, error, log, fatal, panic.
pglogical.conflict_resolution	Menetapkan metode untuk digunakan untuk menyelesa ikan konflik ketika konflik dapat diselesaikan. Nilai string yang didukung adalah kesalahan, apply_remote, keep_local, last_update_wins, first_update_wins.
pglogical.extra_connection_ options	Opsi koneksi untuk ditambahkan ke semua koneksi simpul peer.
pglogical.synchronous_commit	Nilai komit sinkron spesifik pglogical
pglogical.use_spi	Gunakan SPI (antarmuka pemrograman server) alih-alih API tingkat rendah untuk menerapkan perubahan. Atur ke '1' untuk menghidupkan, '0' untuk mematikan. Untuk informasi SPI selengkapnya, lihat Server Programming Interface dalam dokumentasi PostgreSQL.

Bekerja dengan pembungkus data asing yang didukung untuk Amazon Aurora PostgreSQL

Pembungkus data asing (FDW) adalah jenis ekstensi tertentu yang memberikan akses ke data eksternal. Misalnya, ekstensi `oracle_fdw` memungkinkan instans DB Aurora PostgreSQL bekerja dengan basis data Oracle.

Selanjutnya, Anda dapat menemukan informasi tentang beberapa pembungkus data asing PostgreSQL yang didukung.

Topik

- Menggunakan ekstensi log_fdw untuk mengakses log DB dengan SQL
- Menggunakan ekstensi postgres_fdw untuk mengakses data eksternal
- Bekerja dengan basis data MySQL menggunakan ekstensi mysql_fdw
- Bekerja dengan basis data Oracle menggunakan ekstensi oracle_fdw
- Bekerja dengan basis data SQL Server menggunakan ekstensi tds_fdw

Menggunakan ekstensi log_fdw untuk mengakses log DB dengan SQL

Klaster DB Aurora PostgreSQL mendukung ekstensi `log_fdw` yang dapat Anda gunakan untuk mengakses log mesin basis data menggunakan antarmuka SQL. Ekstensi `log_fdw` ini memberikan dua fungsi yang memudahkan pembuatan tabel asing untuk log basis data:

- `list_postgres_log_files` – Mencantumkan file di direktori log basis data dan ukuran file dalam byte.
- `create_foreign_table_for_log_file(table_name text, server_name text, log_file_name text)` – Membangun tabel asing untuk file yang ditentukan dalam basis data saat ini.

Semua fungsi yang dibuat oleh `log_fdw` dimiliki oleh `rds_superuser`. Anggota dengan peran `rds_superuser` dapat memberikan akses ke fungsi ini kepada pengguna basis data lain.

Secara default, file log dibuat oleh Amazon Aurora dalam format `stderr` (kesalahan standar), seperti yang ditentukan pada parameter `log_destination`. Hanya ada dua opsi untuk parameter ini, yaitu `stderr` dan `csvlog` (nilai yang dipisahkan koma, CSV). Jika Anda menambahkan opsi `csvlog` ke parameter, Amazon Aurora akan membuat log `stderr` dan `csvlog`. Hal ini dapat mempengaruhi kapasitas penyimpanan pada klaster DB sehingga Anda perlu mengetahui parameter lain yang mempengaruhi penanganan log. Untuk informasi selengkapnya, lihat Mengatur tujuan log (`stderr`, `csvlog`).

Salah satu manfaat pembuatan log `csvlog` adalah ekstensi `log_fdw` memungkinkan Anda membangun tabel asing dengan data yang terbagi dengan rapi menjadi beberapa kolom. Untuk melakukannya, instans Anda harus terkait dengan grup parameter DB khusus sehingga Anda dapat mengubah pengaturan `log_destination`. Untuk informasi selengkapnya tentang cara melakukannya, lihat Bekerja dengan grup parameter.

Contoh berikut mengasumsikan bahwa parameter `log_destination` mencakup `cvslog`.

Untuk menggunakan ekstensi log_fdw

1. Instal ekstensi `log_fdw`.

```
postgres=> CREATE EXTENSION log_fdw;
CREATE EXTENSION
```

2. Buat server log sebagai pembungkus data asing.

```
postgres=> CREATE SERVER log_server FOREIGN DATA WRAPPER log_fdw;
CREATE SERVER
```

3. Pilih semua dari daftar file log.

```
postgres=> SELECT * FROM list_postgres_log_files() ORDER BY 1;
```

Respons sampel sebagai berikut.

```
          file_name           | file_size_bytes
------------------------------+-----------------
 postgresql.log.2023-08-09-22.csv |            1111
 postgresql.log.2023-08-09-23.csv |            1172
 postgresql.log.2023-08-10-00.csv |            1744
 postgresql.log.2023-08-10-01.csv |            1102
(4 rows)
```

4. Membuat tabel dengan satu kolom 'log_entry' untuk file yang dipilih.

```
postgres=> SELECT create_foreign_table_for_log_file('my_postgres_error_log',
     'log_server', 'postgresql.log.2023-08-09-22.csv');
```

Respons tersebut tidak memberikan detail selain memberitahukan bahwa tabel telah ada.

```
----------------------------------
(1 row)
```

5. Pilih sampel file log. Kode berikut mengambil waktu log dan deskripsi pesan kesalahan.

```
postgres=> SELECT log_time, message FROM my_postgres_error_log ORDER BY 1;
```

Respons sampel sebagai berikut.

```
             log_time             |                                  message
----------------------------------
+-------------------------------------------------------------------------------
Tue Aug 09 15:45:18.172 2023 PDT | ending log output to stderr
Tue Aug 09 15:45:18.175 2023 PDT | database system was interrupted; last known up
 at 2023-08-09 22:43:34 UTC
Tue Aug 09 15:45:18.223 2023 PDT | checkpoint record is at 0/90002E0
Tue Aug 09 15:45:18.223 2023 PDT | redo record is at 0/90002A8; shutdown FALSE
Tue Aug 09 15:45:18.223 2023 PDT | next transaction ID: 0/1879; next OID: 24578
Tue Aug 09 15:45:18.223 2023 PDT | next MultiXactId: 1; next MultiXactOffset: 0
Tue Aug 09 15:45:18.223 2023 PDT | oldest unfrozen transaction ID: 1822, in
 database 1
(7 rows)
```

Menggunakan ekstensi postgres_fdw untuk mengakses data eksternal

Anda dapat mengakses data dalam tabel di server basis data jarak jauh dengan ekstensi postgres_fdw. Jika Anda mengatur koneksi jarak jauh dari instans DB PostgreSQL, akses juga tersedia untuk replika baca Anda.

Untuk menggunakan postgres_fdw agar dapat mengakses server basis data jarak jauh

1. Instal ekstensi postgres_fdw.

```
CREATE EXTENSION postgres_fdw;
```

2. Buat server data asing menggunakan CREATE SERVER.

```
CREATE SERVER foreign_server
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (host 'xxx.xx.xxx.xx', port '5432', dbname 'foreign_db');
```

3. Buat pemetaan pengguna untuk mengidentifikasi peran yang akan digunakan pada server jarak jauh.

```
CREATE USER MAPPING FOR local_user
SERVER foreign_server
OPTIONS (user 'foreign_user', password 'password');
```

4. Buat tabel yang memetakan ke tabel pada server jarak jauh.

```
CREATE FOREIGN TABLE foreign_table (
        id integer NOT NULL,
        data text)
SERVER foreign_server
OPTIONS (schema_name 'some_schema', table_name 'some_table');
```

Bekerja dengan basis data MySQL menggunakan ekstensi mysql_fdw

Untuk mengakses basis data yang kompatibel dengan MySQL dari klaster DB Aurora PostgreSQL , Anda dapat menginstal dan menggunakan ekstensi `mysql_fdw`. Pembungkus data asing ini memungkinkan Anda bekerja dengan RDS for MySQL, Aurora MySQL, MariaDB, dan basis data MySQL lainnya yang kompatibel. Koneksi dari klaster DB Aurora PostgreSQL ke basis data MySQL dienkripsi secara optimal, tergantung pada konfigurasi klien dan server. Namun, Anda dapat menerapkan enkripsi jika diinginkan. Untuk informasi selengkapnya, lihat Menggunakan enkripsi bergerak dengan ekstensi.

Ekstensi `mysql_fdw` didukung pada Amazon Aurora PostgreSQL versi 15.4, 14.9, 13.12, 12.16, dan rilis yang lebih baru. Ekstensi ini mendukung untuk memilih, menyisipkan, memperbarui, dan menghapus dari DB RDS for PostgreSQL ke tabel pada instans basis data yang kompatibel dengan MySQL.

Topik

- Mengatur basis data Aurora PostgreSQL untuk menggunakan ekstensi mysql_fdw
- Contoh: Bekerja dengan basis data Aurora MySQL dari Aurora PostgreSQL
- Menggunakan enkripsi bergerak dengan ekstensi

Mengatur basis data Aurora PostgreSQL untuk menggunakan ekstensi mysql_fdw

Mengatur ekstensi `mysql_fdw` pada klaster DB Aurora PostgreSQL melibatkan pemuatan ekstensi pada klaster DB , lalu membuat koneksi yang mengarah ke instans DB MySQL. Untuk tugas tersebut, Anda harus memiliki detail instans DB MySQL berikut:

- Nama host atau titik akhir. Untuk klaster DB Aurora MySQL , Anda dapat menemukan titik akhir menggunakan Konsol. Pilih tab Konektivitas & keamanan, lalu lihat di bagian "Titik akhir dan port".
- Nomor port. Nomor port default untuk MySQL adalah 3306.
- Nama basis data. Pengidentifikasi DB.

Anda juga perlu memberikan akses di grup keamanan atau daftar kontrol akses (ACL) untuk port MySQL, 3306. Baik klaster DB Aurora PostgreSQL dan klaster DB Aurora MySQL memerlukan akses ke port 3306. Jika akses tidak dikonfigurasi dengan benar, Anda akan melihat pesan kesalahan yang serupa dengan berikut ini saat mencoba menghubungkan ke tabel yang kompatibel dengan MySQL:

```
ERROR: failed to connect to MySQL: Can't connect to MySQL server on 'hostname.aws-
region.rds.amazonaws.com:3306' (110)
```

Dalam prosedur berikut, Anda (sebagai akun `rds_superuser`) akan membuat server asing. Kemudian, Anda akan memberikan akses ke server asing untuk pengguna tertentu. Pengguna ini akan membuat pemetaan mereka sendiri ke akun pengguna MySQL yang sesuai untuk bekerja dengan instans DB MySQL.

Untuk menggunakan mysql_fdw agar dapat mengakses server basis data MySQL

1. Hubungkan ke instans DB PostgreSQL Anda menggunakan akun yang memiliki peran `rds_superuser`. Jika Anda menerima default saat membuat klaster DB Aurora PostgreSQL , nama pengguna adalah `postgres`, dan Anda dapat terhubung menggunakan alat baris perintah `psql` sebagai berikut:

   ```
   psql --host=your-DB-instance.aws-region.rds.amazonaws.com --port=5432 --
   username=postgres --password
   ```

2. Instal ekstensi `mysql_fdw` sebagai berikut:

   ```
   postgres=> CREATE EXTENSION mysql_fdw;
   CREATE EXTENSION
   ```

Setelah ekstensi diinstal pada klaster DB Aurora PostgreSQL , Anda dapat mengatur server asing yang memberikan koneksi ke basis data MySQL.

Untuk membuat server asing

Lakukan tugas-tugas ini pada klaster DB Aurora PostgreSQL . Langkah-langkah ini mengasumsikan bahwa Anda terhubung sebagai pengguna dengan hak istimewa `rds_superuser`, seperti `postgres`.

1. Membuat server asing pada klaster DB Aurora PostgreSQL :

```
postgres=> CREATE SERVER mysql-db FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'db-name.111122223333.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. Berikan akses pengguna yang sesuai ke server asing. Pengguna yang diberi akses harus pengguna non-administrator, yaitu pengguna tanpa peran `rds_superuser`.

```
postgres=> GRANT USAGE ON FOREIGN SERVER mysql-db to user1;
GRANT
```

Pengguna PostgreSQL membuat dan mengelola koneksi mereka ke basis data MySQL melalui server asing.

Contoh: Bekerja dengan basis data Aurora MySQL dari Aurora PostgreSQL

Misalnya, Anda memiliki tabel sederhana pada instans DB Aurora PostgreSQL . Pengguna Aurora PostgreSQL Anda ingin membuat kueri item (`SELECT`), `INSERT`, `UPDATE`, dan `DELETE` pada tabel tersebut. Asumsikan bahwa ekstensi `mysql_fdw` dibuat di instans DB RDS for PostgreSQL, seperti yang dijelaskan dalam prosedur sebelumnya. Setelah terhubung ke instans DB RDS for PostgreSQL sebagai pengguna yang memiliki hak istimewa `rds_superuser`, Anda dapat melanjutkan langkah-langkah berikut.

1. Pada instans DB Aurora PostgreSQL , buat server asing:

```
test=> CREATE SERVER mysqldb FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host 'your-DB.aws-region.rds.amazonaws.com', port '3306');
CREATE SERVER
```

2. Izinkan penggunaan kepada pengguna yang tidak memiliki izin `rds_superuser`, misalnya `user1`:

```
test=> GRANT USAGE ON FOREIGN SERVER mysqldb TO user1;
GRANT
```

3. Hubungkan sebagai *user1*, lalu buat pemetaan ke pengguna MySQL:

```
test=> CREATE USER MAPPING FOR user1 SERVER mysqldb OPTIONS (username 'myuser',
 password 'mypassword');
CREATE USER MAPPING
```

4. Buat tabel asing yang dihubungkan ke tabel MySQL:

```
test=> CREATE FOREIGN TABLE mytab (a int, b text) SERVER mysqldb OPTIONS (dbname
 'test', table_name '');
CREATE FOREIGN TABLE
```

5. Jalankan kueri sederhana pada tabel asing:

```
test=> SELECT * FROM mytab;
a |   b
---+-------
1 | apple
(1 row)
```

6. Anda dapat menambahkan, mengubah, dan menghapus data dari tabel MySQL. Misalnya:

```
test=> INSERT INTO mytab values (2, 'mango');
INSERT 0 1
```

   Jalankan lagi kueri `SELECT` untuk melihat hasilnya:

```
test=> SELECT * FROM mytab ORDER BY 1;
 a |   b
---+-------
1 | apple
2 | mango
(2 rows)
```

Menggunakan enkripsi bergerak dengan ekstensi

Koneksi ke MySQL dari Aurora PostgreSQL menggunakan enkripsi bergerak (TLS/SSL) secara default. Namun, koneksi akan kembali ke non-enkripsi saat konfigurasi klien dan server berbeda. Anda dapat menerapkan enkripsi untuk semua koneksi yang keluar dengan menentukan opsi `REQUIRE SSL` pada akun pengguna RDS for MySQL. Pendekatan yang sama juga bekerja untuk akun pengguna MariaDB dan Aurora MySQL.

Untuk akun pengguna MySQL yang dikonfigurasi ke `REQUIRE SSL`, upaya koneksi akan gagal jika koneksi tidak aman.

Untuk menerapkan enkripsi akun pengguna basis data MySQL yang ada, Anda dapat menggunakan perintah `ALTER USER`. Sintaksis dapat berbeda-beda, bergantung pada versi MySQL, seperti yang ditunjukkan pada tabel berikut. Untuk informasi selengkapnya, lihat ALTER USER pada Panduan Referensi MySQL.

MySQL 5.7, MySQL 8.0	MySQL 5.6
`ALTER USER 'user'@'%' REQUIRE SSL;`	`GRANT USAGE ON *.* to 'user'@'%' REQUIRE SSL;`

Untuk informasi selengkapnya tentang ekstensi `mysql_fdw`, lihat dokumentasi mysql_fdw.

Bekerja dengan basis data Oracle menggunakan ekstensi oracle_fdw

Untuk mengakses basis data Oracle dari klaster DB Aurora PostgreSQL , Anda dapat menginstal dan menggunakan ekstensi `oracle_fdw`. Ekstensi ini adalah pembungkus data asing untuk basis data Oracle. Untuk mempelajari selengkapnya tentang ekstensi ini, lihat dokumentasi oracle_fdw.

Ekstensi `oracle_fdw` didukung pada Aurora PostgreSQL 12.7 (Amazon Aurora rilis 4.2) dan versi yang lebih baru.

Topik

- Mengaktifkan ekstensi oracle_fdw
- Contoh: Menggunakan server asing yang terhubung ke basis data Amazon RDS for Oracle
- Bekerja dengan enkripsi bergerak
- Memahami tampilan dan izin pg_user_mappings

Mengaktifkan ekstensi oracle_fdw

Untuk menggunakan ekstensi oracle_fdw, lakukan prosedur berikut.

Untuk mengaktifkan ekstensi oracle_fdw

- Jalankan perintah berikut menggunakan akun yang memiliki izin `rds_superuser`.

```
CREATE EXTENSION oracle_fdw;
```

Contoh: Menggunakan server asing yang terhubung ke basis data Amazon RDS for Oracle

Contoh berikut ini menunjukkan penggunaan server asing yang terhubung ke basis data Amazon RDS for Oracle.

Untuk membuat server asing yang terhubung ke basis data RDS for Oracle

1. Perhatikan hal-hal berikut ini pada instans DB RDS for Oracle:
   - Titik akhir
   - Port
   - Nama basis data
2. Membuat server asing.

```
test=> CREATE SERVER oradb FOREIGN DATA WRAPPER oracle_fdw OPTIONS (dbserver
 '//endpoint:port/DB_name');
CREATE SERVER
```

3. Izinkan penggunaan kepada pengguna yang tidak memiliki hak istimewa `rds_superuser`, misalnya `user1`.

```
test=> GRANT USAGE ON FOREIGN SERVER oradb TO user1;
GRANT
```

4. Hubungkan sebagai `user1`, lalu buat pemetaan kepada pengguna Oracle.

```
test=> CREATE USER MAPPING FOR user1 SERVER oradb OPTIONS (user 'oracleuser',
 password 'mypassword');
CREATE USER MAPPING
```

5. Buat tabel asing yang dihubungkan ke tabel Oracle.

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER oradb OPTIONS (table 'MYTABLE');
CREATE FOREIGN TABLE
```

6. Buat kueri tabel asing.

```
test=>  SELECT * FROM mytab;
a
---
1
```

```
(1 row)
```

Jika kueri melaporkan kesalahan berikut, periksa grup keamanan dan daftar kontrol akses (ACL) untuk memastikan bahwa kedua instans dapat berkomunikasi.

```
ERROR: connection for foreign table "mytab" cannot be established
DETAIL: ORA-12170: TNS:Connect timeout occurred
```

Bekerja dengan enkripsi bergerak

Enkripsi PostgreSQL-to-Oracle bergerak didasarkan pada kombinasi parameter konfigurasi klien dan server. Untuk contoh menggunakan Oracle 21c, lihat Tentang Nilai untuk Melakukan Negosiasi Enkripsi dan Integritas pada dokumentasi Oracle. Klien yang digunakan untuk oracle_fdw di Amazon RDS dikonfigurasi dengan `ACCEPTED`, artinya enkripsi bergantung pada konfigurasi server basis data Oracle.

Jika basis data Anda menggunakan RDS for Oracle, lihat Enkripsi jaringan asli Oracle untuk mengonfigurasi enkripsi.

Memahami tampilan dan izin pg_user_mappings

Katalog PostgreSQL `pg_user_mapping` menyimpan pemetaan dari pengguna Aurora PostgreSQL ke pengguna di server data asing (jarak jauh). Akses ke katalog dibatasi, tetapi Anda menggunakan tampilan `pg_user_mappings` untuk melihat pemetaan. Berikut ini Anda dapat menemukan contoh yang menunjukkan bagaimana izin berlaku dengan contoh basis data Oracle, tetapi informasi ini biasanya berlaku untuk pembungkus data asing apa pun.

Pada output berikut, Anda dapat menemukan peran dan izin yang dipetakan kepada tiga pengguna contoh yang berbeda. Pengguna `rdssu1` dan `rdssu2` adalah anggota dengan peran `rds_superuser`, sedangkan `user1` tidak. Contoh menggunakan metacommand `psql` `\du` untuk daftar peran yang ada.

```
test=>  \du
                                                    List of roles
    Role name    |                         Attributes                         |
                  Member of
-----------------+------------------------------------------------------------
+-----------------------------------------------------------
 rdssu1          |                                                            |
 {rds_superuser}
```

```
 rdssu2          |                                                                |
 {rds_superuser}
 user1           |                                                                | {}
```

Semua pengguna, termasuk pengguna yang memiliki hak istimewa `rds_superuser`, diizinkan untuk melihat pemetaan pengguna (`umoptions`) mereka sendiri di tabel `pg_user_mappings`. Seperti yang ditunjukkan pada contoh berikut, saat `rdssu1` mencoba untuk mendapatkan semua pemetaan pengguna, kesalahan ditampilkan meskipun hak istimewa `rdssu1` `rds_superuser`:

```
test=> SELECT * FROM pg_user_mapping;
ERROR: permission denied for table pg_user_mapping
```

Berikut adalah beberapa contoh.

```
test=> SET SESSION AUTHORIZATION rdssu1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+---------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     | {user=oracleuser,password=mypwd}
 16424 | 16411 | oradb   |  16422 | rdssu2     |
 (3 rows)

test=> SET SESSION AUTHORIZATION rdssu2;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |            umoptions
-------+-------+---------+--------+------------+---------------------------------
 16414 | 16411 | oradb   |  16412 | user1      |
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     | {user=oracleuser,password=mypwd}
 (3 rows)

test=> SET SESSION AUTHORIZATION user1;
SET
test=> SELECT * FROM pg_user_mappings;
 umid  | srvid | srvname | umuser | usename    |           umoptions
-------+-------+---------+--------+------------+-------------------------------
 16414 | 16411 | oradb   |  16412 | user1      | {user=oracleuser,password=mypwd}
 16423 | 16411 | oradb   |  16421 | rdssu1     |
 16424 | 16411 | oradb   |  16422 | rdssu2     |
```

```
(3 rows)
```

Karena perbedaan implementasi antara `information_schema._pg_user_mappings` dan `pg_catalog.pg_user_mappings`, maka `rds_superuser` yang dibuat secara manual memerlukan izin tambahan untuk dapat melihat kata sandi di `pg_catalog.pg_user_mappings`.

`rds_superuser` tidak memerlukan izin tambahan untuk dapat melihat kata sandi di `information_schema._pg_user_mappings`.

Pengguna yang tidak memiliki peran `rds_superuser` dapat melihat kata sandi `pg_user_mappings` hanya dalam kondisi berikut:

- Pengguna saat ini adalah pengguna yang dipetakan dan memiliki server atau memegang hak istimewa `USAGE` atas server tersebut.
- Pengguna saat ini adalah pemilik server dan pemetaannya untuk `PUBLIC`.

Bekerja dengan basis data SQL Server menggunakan ekstensi tds_fdw

Anda dapat menggunakan ekstensi `tds_fdw` PostgreSQL untuk mengakses basis data yang mendukung protokol aliran data tabular (TDS), seperti basis data Sybase dan Microsoft SQL Server. Pembungkus data asing ini memungkinkan Anda terhubung dari klaster DB Aurora PostgreSQL ke basis data yang menggunakan protokol TDS, termasuk Amazon RDS for Microsoft SQL Server. Untuk informasi selengkapnya, lihat dokumentasi tds-fdw/tds_fdw pada GitHub.

Ekstensi `tds_fdw` didukung pada Amazon Aurora PostgreSQL versi 13.6 dan rilis yang lebih baru.

Mengatur DB Aurora PostgreSQL untuk menggunakan ekstensi tds_fdw

Dalam prosedur berikut, Anda dapat menemukan contoh pengaturan dan penggunaan `tds_fdw` dengan klaster DB Aurora PostgreSQL. Sebelum dapat terhubung ke basis data SQL Server menggunakan `tds_fdw`, Anda harus mendapatkan detail berikut untuk instans:

- Nama host atau titik akhir. Untuk instans DB RDS for SQL Server, Anda dapat menemukan titik akhir menggunakan Konsol. Pilih tab Konektivitas & keamanan, lalu lihat di bagian "Titik akhir dan port".
- Nomor port. Nomor port default untuk Microsoft SQL Server adalah 1433.
- Nama basis data. Pengidentifikasi DB.

Anda juga perlu memberikan akses di grup keamanan atau daftar kontrol akses (ACL) untuk port SQL Server, 1433. Baik klaster DB Aurora PostgreSQL dan instans DB RDS for SQL Server memerlukan akses ke port 1433. Jika akses tidak dikonfigurasi dengan benar, Anda akan melihat pesan kesalahan seperti berikut saat mencoba membuat kueri Microsoft SQL Server:

```
ERROR: DB-Library error: DB #: 20009, DB Msg: Unable to connect:
Adaptive Server is unavailable or does not exist (mssql2019.aws-
region.rds.amazonaws.com), OS #: 0, OS Msg: Success, Level: 9
```

Untuk menggunakan tds_fdw agar terhubung ke basis data SQL Server

1. Hubungkan ke instans utama klaster DB Aurora PostgreSQL menggunakan akun yang memiliki peran `rds_superuser`:

   ```
   psql --host=your-cluster-name-instance-1.aws-region.rds.amazonaws.com --port=5432
    --username=test --password
   ```

2. Instal ekstensi `tds_fdw`:

   ```
   test=> CREATE EXTENSION tds_fdw;
   CREATE EXTENSION
   ```

Setelah ekstensi diinstal pada klaster DB Aurora PostgreSQL , Anda dapat mengatur server asing.

Untuk membuat server asing

Lakukan tugas-tugas ini pada klaster DB Aurora PostgreSQL menggunakan akun yang memiliki hak istimewa `rds_superuser`.

1. Membuat server asing pada klaster DB Aurora PostgreSQL :

   ```
   test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS
    (servername 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database
    'tds_fdw_testing');
   CREATE SERVER
   ```

   Untuk mengakses data non-ASCII di sisi SQLServer, buat tautan server dengan opsi character_set pada klaster DB Aurora PostgreSQL :

```
test=> CREATE SERVER sqlserverdb FOREIGN DATA WRAPPER tds_fdw OPTIONS (servername
 'mssql2019.aws-region.rds.amazonaws.com', port '1433', database 'tds_fdw_testing',
 character_set 'UTF-8');
CREATE SERVER
```

2. Berikan izin kepada pengguna yang tidak memiliki hak istimewa peran `rds_superuser`, misalnya `user1`:

```
test=> GRANT USAGE ON FOREIGN SERVER sqlserverdb TO user1;
```

3. Hubungkan sebagai user1, lalu buat pemetaan kepada pengguna SQL Server:

```
test=> CREATE USER MAPPING FOR user1 SERVER sqlserverdb OPTIONS (username
 'sqlserveruser', password 'password');
CREATE USER MAPPING
```

4. Buat tabel asing yang dihubungkan ke tabel SQL Server:

```
test=> CREATE FOREIGN TABLE mytab (a int) SERVER sqlserverdb OPTIONS (table
 'MYTABLE');
CREATE FOREIGN TABLE
```

5. Buat kueri tabel asing:

```
test=> SELECT * FROM mytab;
 a
---
 1
(1 row)
```

Gunakan enkripsi bergerak untuk koneksi

Koneksi dari Aurora PostgreSQL ke SQL Server menggunakan enkripsi bergerak (TLS/SSL) bergantung pada konfigurasi basis data SQL Server. Jika SQL Server tidak dikonfigurasi untuk enkripsi, klien RDS for PostgreSQL yang membuat permintaan ke basis data SQL Server akan dikembalikan ke status tidak terenkripsi.

Anda dapat menerapkan enkripsi untuk koneksi ke instans DB RDS for SQL Server dengan mengatur parameter `rds.force_ssl`. Untuk mempelajari caranya, lihat Memaksa koneksi ke instans DB

untuk menggunakan SSL. Untuk informasi selengkapnya tentang konfigurasi SSL/TLS untuk RDS for SQL Server, lihat Menggunakan SSL dengan instans DB Microsoft SQL Server.

Bekerja dengan Ekstensi Bahasa Tepercaya untuk PostgreSQL

Ekstensi Bahasa Tepercaya untuk PostgreSQL adalah kit pengembangan sumber terbuka untuk mendesain ekstensi PostgreSQL. Ini memungkinkan Anda untuk membangun ekstensi PostgreSQL performa tinggi dan menjalankannya dengan aman di klaster DB Aurora PostgreSQL. Dengan menggunakan Ekstensi Bahasa Tepercaya (TLE) untuk PostgreSQL, Anda dapat membuat ekstensi PostgreSQL yang mengikuti pendekatan terdokumentasi untuk memperluas fungsionalitas PostgreSQL. Lihat informasi selengkapnya, lihat Mengemas Objek Terkait ke dalam Ekstensi dalam dokumentasi PostgreSQL.

Salah satu manfaat utama TLE adalah Anda dapat menggunakannya di lingkungan yang tidak menyediakan akses ke sistem file yang mendasari instans PostgreSQL. Sebelumnya, penginstalan ekstensi baru memerlukan akses ke sistem file. TLE menghilangkan kendala ini. Ini menyediakan lingkungan pengembangan untuk membuat ekstensi baru untuk basis data PostgreSQL apa pun, termasuk yang berjalan di klaster DB Aurora PostgreSQL.

TLE dirancang untuk mencegah akses ke sumber daya yang tidak aman untuk ekstensi yang Anda buat menggunakan TLE. Lingkungan runtime-nya membatasi dampak kerusakan ekstensi apa pun ke koneksi basis data tunggal. TLE juga memberi administrator basis data kontrol terperinci atas siapa saja yang dapat menginstal ekstensi, dan memberikan model izin untuk menjalankannya.

TLE didukung pada Aurora PostgreSQL versi 14.5 dan yang lebih tinggi.

Lingkungan pengembangan dan runtime Ekstensi Bahasa Tepercaya dikemas sebagai ekstensi PostgreSQL `pg_tle`, versi 1.0.1. Ini mendukung pembuatan ekstensi di JavaScript, Perl, Tcl, PL/PGSQL, dan SQL. Anda menginstal ekstensi `pg_tle` di klaster DB Aurora PostgreSQL dengan cara yang sama seperti Anda menginstal ekstensi PostgreSQL lainnya. Setelah `pg_tle` disiapkan, developer dapat menggunakannya untuk membuat ekstensi PostgreSQL baru, yang dikenal sebagai ekstensi TLE.

Dalam topik berikut, Anda dapat menemukan informasi tentang cara mengatur Ekstensi Bahasa Tepercaya dan cara memulai membuat ekstensi TLE Anda sendiri.

Topik

- Terminologi
- Persyaratan untuk menggunakan Ekstensi Bahasa Tepercaya untuk PostgreSQL
- Menyiapkan Ekstensi Bahasa Tepercaya di klaster DB Aurora PostgreSQL Anda

- Ikhtisar Ekstensi Bahasa Tepercaya untuk PostgreSQL
- Membuat ekstensi TLE untuk Aurora PostgreSQL
- Menghapus ekstensi TLE dari basis data
- Meng-uninstal Ekstensi Bahasa Tepercaya untuk PostgreSQL
- Menggunakan hook PostgreSQL dengan ekstensi TLE
- Referensi fungsi untuk Ekstensi Bahasa Tepercaya untuk PostgreSQL
- Referensi hook untuk Ekstensi Bahasa Tepercaya untuk PostgreSQL

Terminologi

Untuk membantu Anda lebih memahami Ekstensi Bahasa Tepercaya, lihat glosarium berikut untuk istilah yang digunakan dalam topik ini.

Ekstensi Bahasa Tepercaya untuk PostgreSQL

: Ekstensi Bahasa Tepercaya untuk PostgreSQL adalah nama resmi kit pengembangan sumber terbuka yang dikemas sebagai ekstensi `pg_tle`. Ini dapat digunakan pada sistem PostgreSQL apa pun. Untuk informasi selengkapnya, lihat aws/pg_tle di. GitHub

Ekstensi Bahasa Terpercaya

: Ekstensi Bahasa Tepercaya adalah singkatan dari Ekstensi Bahasa Tepercaya untuk PostgreSQL. Nama singkat ini dan singkatannya (TLE) juga digunakan dalam dokumentasi ini.

bahasa tepercaya

: Bahasa tepercaya adalah bahasa pemrograman atau skrip yang memiliki atribut keamanan tertentu. Misalnya, bahasa tepercaya biasanya membatasi akses ke sistem file, dan membatasi penggunaan properti jaringan tertentu. Kit pengembangan TLE dirancang untuk mendukung bahasa tepercaya. PostgreSQL mendukung beberapa bahasa yang berbeda yang digunakan untuk membuat ekstensi tepercaya atau tidak tepercaya. Sebagai contoh, lihat PL/Perl Tepercaya dan Tidak Tepercaya dalam dokumentasi PostgreSQL. Saat Anda membuat ekstensi menggunakan Ekstensi Bahasa Tepercaya, ekstensi tersebut akan menggunakan mekanisme bahasa tepercaya secara inheren.

Ekstensi TLE

: Ekstensi TLE adalah ekstensi PostgreSQL yang dibuat dengan menggunakan kit pengembangan Ekstensi Bahasa Tepercaya (TLE).

Persyaratan untuk menggunakan Ekstensi Bahasa Tepercaya untuk PostgreSQL

Berikut ini adalah persyaratan untuk menyiapkan dan menggunakan kit pengembangan TLE.

- Versi Aurora PostgreSQL – Ekstensi Bahasa Tepercaya didukung pada Aurora PostgreSQL versi 14.5 dan versi yang lebih tinggi saja.
  - Jika Anda perlu mengupgrade klaster DB Aurora PostgreSQL, lihat Meningkatkan klaster DB Amazon Aurora PostgreSQL.
  - Jika belum memiliki klaster DB Aurora yang menjalankan PostgreSQL, Anda dapat membuatnya. Untuk informasi selengkapnya, lihat Membuat dan terhubung ke klaster DB Aurora PostgreSQL.
- Memerlukan hak istimewa **`rds_superuser`** – Untuk menyiapkan dan mengonfigurasi ekstensi `pg_tle`, peran pengguna basis data Anda harus memiliki izin peran `rds_superuser`. Secara default, peran ini diberikan kepada `postgres` pengguna yang membuat klaster DB Aurora PostgreSQL.
- Memerlukan grup parameter DB kustom - Klaster DB Aurora PostgreSQL Anda harus dikonfigurasi dengan grup parameter DB kustom. Anda menggunakan grup parameter DB kustom untuk instans penulis klaster DB Aurora PostgreSQL.
  - Jika klaster DB Aurora PostgreSQL Anda tidak dikonfigurasi dengan grup parameter DB kustom, Anda harus membuatnya dan mengaitkannya dengan instans penulis klaster DB Aurora PostgreSQL Anda. Untuk ringkasan langkah-langkahnya, lihat Membuat dan menerapkan grup parameter DB kustom.
  - Jika klaster DB Aurora PostgreSQL Anda sudah dikonfigurasi menggunakan grup parameter DB kustom, Anda dapat menyiapkan Ekstensi Bahasa Tepercaya. Untuk detailnya, lihat Menyiapkan Ekstensi Bahasa Tepercaya di klaster DB Aurora PostgreSQL Anda.

Membuat dan menerapkan grup parameter DB kustom

Gunakan langkah-langkah berikut untuk membuat grup parameter DB kustom dan mengonfigurasi klaster DB Aurora PostgreSQL untuk menggunakannya.

Konsol

Untuk membuat grup parameter DB kustom dan menggunakannya dengan klaster DB Aurora PostgreSQL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih grup Parameter dari menu Amazon RDS.
3. Pilih Buat grup parameter.
4. Di halaman Detail grup parameter, masukkan informasi berikut.
   - Untuk Keluarga grup Parameter, pilih aurora-postgresql14.
   - Untuk Jenis, pilih Grup Parameter DB.
   - Untuk Nama grup, berikan grup parameter Anda nama yang bermakna dalam konteks operasi Anda.
   - Untuk Deskripsi, masukkan deskripsi yang berguna sehingga orang lain di tim Anda dapat dengan mudah menemukannya.
5. Pilih Buat. Grup parameter DB kustom Anda dibuat di Wilayah AWS. Anda sekarang dapat mengubah klaster DB Aurora PostgreSQL untuk menggunakannya dengan mengikuti langkah selanjutnya.
6. Pilih Basis data dari menu Amazon RDS.
7. Pilih klaster DB Aurora PostgreSQL yang ingin digunakan dengan TLE dari yang tercantum tersebut, lalu pilih Ubah.
8. Di halaman Ubah pengaturan klaster DB, cari Opsi basis data dan gunakan pemilih untuk memilih grup parameter DB kustom Anda.
9. Pilih Lanjutkan untuk menyimpan perubahan.
10. Pilih Terapkan langsung sehingga Anda dapat melanjutkan penyiapan klaster DB Aurora PostgreSQL untuk menggunakan TLE.

Untuk melanjutkan penyiapan sistem untuk Ekstensi Bahasa Tepercaya, lihat Menyiapkan Ekstensi Bahasa Tepercaya di klaster DB Aurora PostgreSQL Anda.

Untuk informasi selengkapnya tentang cara menggunakan Grup parameter DB dan klaster DB, lihat Bekerja dengan grup parameter klaster DB.

AWS CLI

Anda dapat menghindari penentuan argumen `--region` saat menggunakan perintah CLI dengan mengonfigurasi AWS CLI dengan Wilayah AWS default. Untuk informasi selengkapnya, lihat Dasar-dasar konfigurasi di Panduan Pengguna AWS Command Line Interface.

Untuk membuat grup parameter DB kustom dan menggunakannya dengan klaster DB Aurora PostgreSQL

1. Wilayah AWS Perhatikan bahwa dalam langkah ini Anda membuat grup parameter DB untuk diterapkan ke instans penulis klaster DB Aurora PostgreSQL Anda.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-parameter-group \
     --region aws-region \
     --db-parameter-group-name custom-params-for-pg-tle \
     --db-parameter-group-family aurora-postgresql14 \
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

   Untuk Windows:

   ```
   aws rds create-db-parameter-group ^
     --region aws-region ^
     --db-parameter-group-name custom-params-for-pg-tle ^
     --db-parameter-group-family aurora-postgresql14 ^
     --description "My custom DB parameter group for Trusted Language Extensions"
   ```

   Grup parameter DB kustom Anda tersedia di Wilayah AWS Anda, sehingga Anda dapat mengubah instans penulis klaster DB Aurora PostgreSQL untuk menggunakannya.

2. Gunakan modify-db-instanceAWS CLIperintah untuk menerapkan grup parameter DB kustom Anda ke instance penulis cluster Aurora PostgreSQL DB Anda. Perintah ini langsung me-reboot instans yang aktif.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds modify-db-instance \
     --region aws-region \
     --db-instance-identifier your-writer-instance-name \
     --db-parameter-group-name custom-params-for-pg-tle \
   ```

```
  --apply-immediately
```

Untuk Windows:

```
aws rds modify-db-instance ^
  --region aws-region ^
  --db-instance-identifier your-writer-instance-name ^
  --db-parameter-group-name custom-params-for-pg-tle ^
  --apply-immediately
```

Untuk melanjutkan penyiapan sistem untuk Ekstensi Bahasa Tepercaya, lihat Menyiapkan Ekstensi Bahasa Tepercaya di klaster DB Aurora PostgreSQL Anda.

Untuk informasi selengkapnya, lihat Bekerja dengan grup parameter DB dalam instance DB .

Menyiapkan Ekstensi Bahasa Tepercaya di klaster DB Aurora PostgreSQL Anda

Langkah-langkah berikut mengasumsikan bahwa klaster DB Aurora PostgreSQL Anda dikaitkan dengan grup parameter klaster DB kustom. Anda dapat menggunakan AWS Management Console atau AWS CLI untuk langkah ini.

Saat menyiapkan Ekstensi Bahasa Tepercaya di klaster DB Aurora PostgreSQL , Anda menginstalnya di basis data tertentu untuk digunakan oleh pengguna basis data yang memiliki izin pada basis data tersebut.

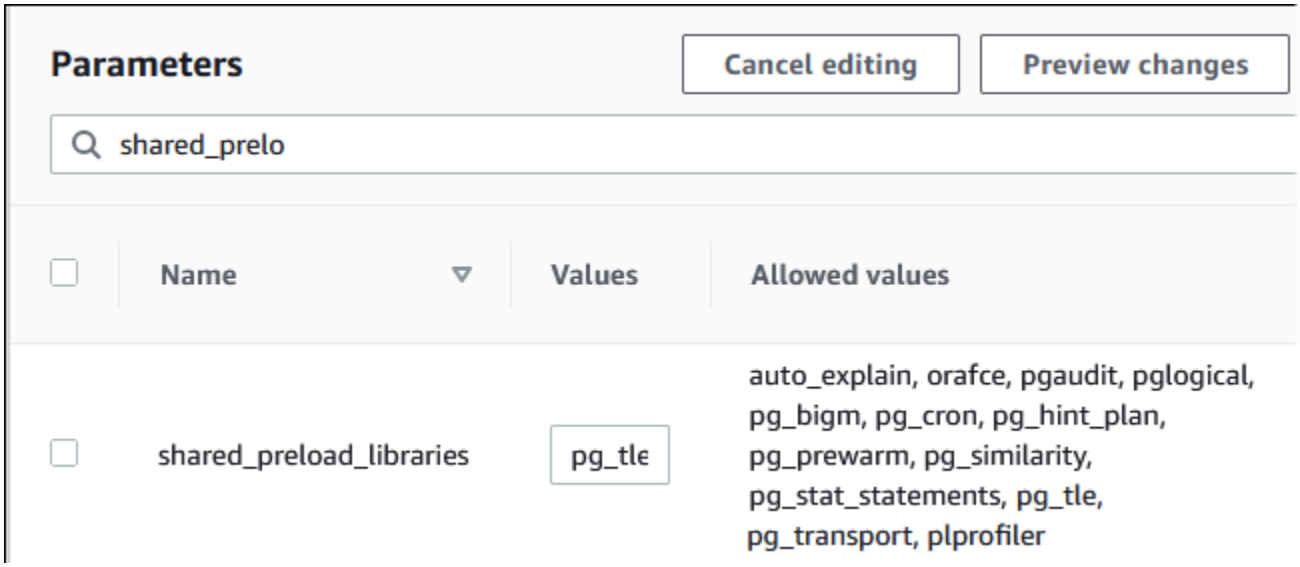
Konsol

Untuk menyiapkan Ekstensi Bahasa Tepercaya

Lakukan langkah-langkah berikut menggunakan akun yang merupakan anggota grup `rds_superuser` (peran).

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Instans penulis klaster DB Aurora PostgreSQL Anda.
3. Buka tab Konfigurasi untuk instans penulis klaster DB Aurora PostgreSQL. Di antara detail Instans, temukan tautan Grup parameter.

4. Pilih tautan untuk membuka parameter kustom yang terkait dengan klaster DB Aurora PostgreSQL.
5. Di kolom pencarian Parameter, ketik `shared_pre` untuk menemukan parameter `shared_preload_libraries`.
6. Pilih Edit parameter untuk mengakses nilai properti.
7. Tambahkan `pg_tle` ke daftar di kolom Nilai. Gunakan koma untuk memisahkan item dalam daftar nilai.



8. Reboot instans penulis klaster DB Aurora PostgreSQL Anda sehingga perubahan pada parameter `shared_preload_libraries` dapat diterapkan.
9. Ketika instans tersedia, verifikasi bahwa `pg_tle` telah diinisialisasi. Gunakan `psql` untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL, lalu jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

10. Dengan ekstensi `pg_tle` yang diinisialisasi, Anda kini dapat membuat ekstensi.

```
CREATE EXTENSION pg_tle;
```

Anda dapat memverifikasi bahwa ekstensi diinstal dengan menggunakan metacommand `psql` berikut.

```
labdb=> \dx
                          List of installed extensions
  Name   | Version |   Schema   |                Description
---------+---------+------------+-------------------------------------------
 pg_tle  | 1.0.1   | pgtle      | Trusted-Language Extensions for PostgreSQL
 plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
```

11. Berikan peran `pgtle_admin` ke nama pengguna utama yang Anda buat untuk klaster DB Aurora PostgreSQL jika Anda menyiapkannya. Jika Anda menerima opsi default-nya, berarti nilainya `postgres`.

```
labdb=> GRANT pgtle_admin TO postgres;
GRANT ROLE
```

Anda dapat memverifikasi bahwa pemberian telah terjadi dengan menggunakan metacommand `psql` seperti yang ditunjukkan pada contoh berikut. Hanya peran `pgtle_admin` dan `postgres` yang ditampilkan dalam output. Untuk informasi selengkapnya, lihat Memahami peran dan izin PostgreSQL.

```
labdb=> \du
                          List of roles
    Role name    |           Attributes            |               Member of
-----------------+---------------------------------
+---------------------------------
pgtle_admin     | Cannot login                     | {}
postgres        | Create role, Create DB          +| {rds_superuser,pgtle_admin}
                | Password valid until infinity    |...
```

12. Tutup sesi `psql` menggunakan metacommand `\q`.

```
\q
```

Untuk mulai membuat ekstensi TLE, lihat Contoh: Membuat ekstensi bahasa tepercaya menggunakan SQL.

AWS CLI

Anda dapat menghindari penentuan argumen `--region` saat menggunakan perintah CLI dengan mengonfigurasi AWS CLI dengan Wilayah AWS default. Untuk informasi selengkapnya, lihat Dasar-dasar konfigurasi di Panduan Pengguna AWS Command Line Interface.

Untuk menyiapkan Ekstensi Bahasa Tepercaya

1. Gunakan modify-db-parameter-groupAWS CLIperintah untuk `pg_tle` menambah `shared_preload_libraries` parameter.

```
aws rds modify-db-parameter-group \
   --db-parameter-group-name custom-param-group-name \
   --parameters
 "ParameterName=shared_preload_libraries,ParameterValue=pg_tle,ApplyMethod=pending-
reboot" \
   --region aws-region
```

2. Gunakan reboot-db-instanceAWS CLIperintah untuk me-reboot instance penulis Aurora PostgreSQL DB cluster instance PostgreSQL DB dan menginisialisasi perpustakaan. `pg_tle`

```
aws rds reboot-db-instance \
    --db-instance-identifier writer-instance \
    --region aws-region
```

3. Ketika instans tersedia, verifikasi bahwa `pg_tle` telah diinisialisasi. Gunakan `psql` untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL, lalu jalankan perintah berikut.

```
SHOW shared_preload_libraries;
shared_preload_libraries
--------------------------
rdsutils,pg_tle
(1 row)
```

   Dengan `pg_tle` diinisialisasi, Anda sekarang dapat membuat ekstensi.

```
CREATE EXTENSION pg_tle;
```

4. Berikan peran `pgtle_admin` ke nama pengguna utama yang Anda buat untuk klaster DB Aurora PostgreSQL jika Anda menyiapkannya. Jika Anda menerima opsi default-nya, berarti nilainya `postgres`.

```
GRANT pgtle_admin TO postgres;
GRANT ROLE
```

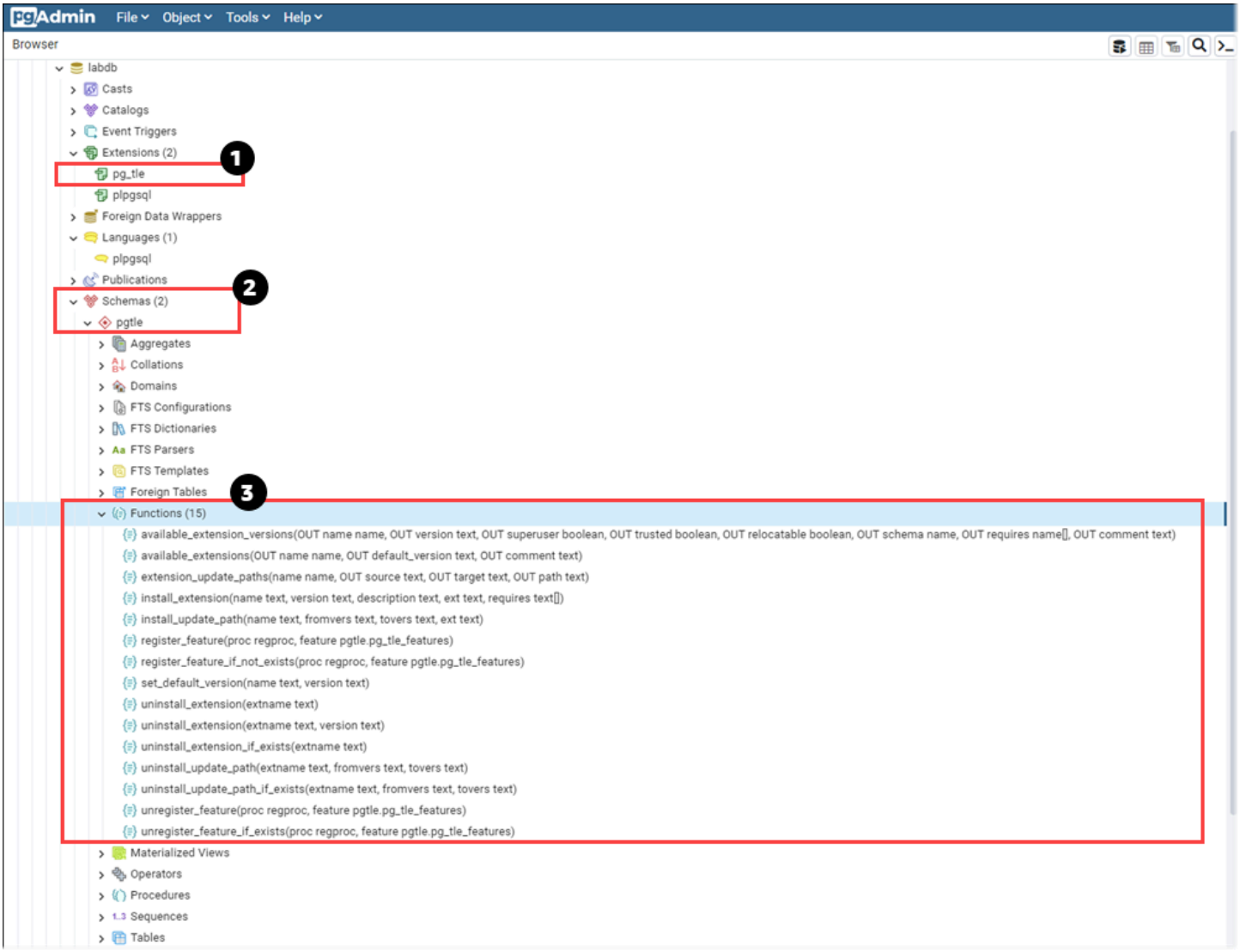
5. Tutup sesi `psql` seperti berikut.

```
labdb=> \q
```

Untuk mulai membuat ekstensi TLE, lihat Contoh: Membuat ekstensi bahasa tepercaya menggunakan SQL.

Ikhtisar Ekstensi Bahasa Tepercaya untuk PostgreSQL

Ekstensi Bahasa Tepercaya untuk PostgreSQL adalah ekstensi PostgreSQL yang Anda instal di klaster DB Aurora PostgreSQL dengan cara yang sama seperti Anda menyiapkan ekstensi PostgreSQL lainnya. Pada gambar contoh basis data berikut alat klien pgAdmin, Anda dapat melihat beberapa komponen yang terdiri atas ekstensi `pg_tle`.



Anda dapat melihat detail berikut.

1. Kit pengembangan Ekstensi Bahasa Tepercaya (TLE) untuk PostgreSQL dikemas sebagai ekstensi `pg_tle`. Dengan demikian, `pg_tle` ditambahkan ke ekstensi yang tersedia untuk basis data tempat kit ini diinstal.
2. TLE memiliki skemanya sendiri, `pgtle`. Skema ini berisi fungsi pembantu (3) untuk menginstal dan mengelola ekstensi yang Anda buat.
3. TLE dilengkapi dengan banyak fungsi pembantu untuk menginstal, mendaftarkan, dan mengelola ekstensi Anda. Untuk mempelajari selengkapnya tentang fungsi ini, lihat Referensi fungsi untuk Ekstensi Bahasa Tepercaya untuk PostgreSQL.

Komponen lain dari ekstensi `pg_tle` meliputi:

- Peran **`pgtle_admin`** – Peran `pgtle_admin` dibuat jika ekstensi `pg_tle` diinstal. Peran ini diistimewakan dan harus diperlakukan semestinya. Sebaiknya Anda mengikuti prinsip hak akses paling rendah saat memberikan peran `pgtle_admin` kepada pengguna basis data. Dengan kata lain, berikan peran `pgtle_admin` hanya kepada pengguna basis data yang diizinkan untuk membuat, menginstal, dan mengelola ekstensi TLE baru, seperti `postgres`.
- Tabel **`pgtle.feature_info`** – Tabel `pgtle.feature_info` adalah tabel yang dilindungi yang berisi informasi tentang TLE, hook, serta prosedur dan fungsi tersimpan kustom yang mereka gunakan. Jika Anda memiliki hak istimewa `pgtle_admin`, gunakan fungsi Ekstensi Bahasa Tepercaya berikut untuk menambahkan dan memperbarui informasi tersebut dalam tabel.
  - pgtle.register_feature
  - pgtle.register_feature_if_not_exists
  - pgtle.unregister_feature
  - pgtle.unregister_feature_if_exists

Membuat ekstensi TLE untuk Aurora PostgreSQL

Anda dapat menginstal ekstensi apa pun yang Anda buat dengan TLE di setiap klaster DB Aurora PostgreSQL yang ekstensi `pg_tle`-nya telah diinstal. Ekstensi `pg_tle` dicakup ke basis data PostgreSQL tempat ekstensi diinstal. Ekstensi yang Anda buat menggunakan TLE dicakup ke basis data yang sama.

Gunakan berbagai fungsi `pgtle` untuk menginstal kode yang membentuk ekstensi TLE Anda. Semua fungsi Ekstensi Bahasa Tepercaya berikut memerlukan peran `pgtle_admin`.

- pgtle.install_extension
- pgtle.install_update_path
- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension (nama)
- pgtle.uninstall_extension (nama, versi)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists

- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists

Contoh: Membuat ekstensi bahasa tepercaya menggunakan SQL

Contoh berikut menunjukkan cara membuat ekstensi TLE bernama `pg_distance` yang berisi beberapa fungsi SQL untuk menghitung jarak menggunakan berbagai formula. Dalam daftar, Anda dapat menemukan fungsi untuk menghitung jarak Manhattan dan fungsi untuk menghitung jarak Euclidean. Untuk informasi selengkapnya tentang perbedaan antara formula ini, lihat Geometri taksi dan Geometri Euclidean di Wikipedia.

Anda dapat menggunakan contoh ini di klaster DB Aurora PostgreSQL Anda sendiri jika Anda memiliki ekstensi `pg_tle` yang diatur seperti yang dijelaskan Menyiapkan Ekstensi Bahasa Tepercaya di klaster DB Aurora PostgreSQL Anda.

> **Note**
>
> Untuk mengikuti prosedur ini, Anda harus memiliki hak istimewa peran `pgtle_admin`.

Untuk membuat contoh ekstensi TLE

Langkah-langkah berikut menggunakan contoh basis data bernama `labdb`. Basis data ini milik pengguna utama `postgres`. Peran `postgres` juga memiliki izin peran `pgtle_admin`.

1. Gunakan `psql` untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL.

   ```
   psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
   --port=5432 --username=postgres --password --dbname=labdb
   ```

2. Buat ekstensi TLE bernama `pg_distance` dengan menyalin kode berikut dan menempelkannya ke konsol sesi `psql` Anda.

   ```
   SELECT pgtle.install_extension
   (
    'pg_distance',
    '0.1',
     'Distance functions for two points',
   $_pg_tle_$
       CREATE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8, norm int)
   ```

```
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL;

    CREATE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL;

    CREATE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL;
$_pg_tle_$
);
```

Anda akan melihat output seperti berikut.

```
install_extension
---------------
 t
(1 row)
```

Artefak yang membentuk ekstensi `pg_distance` sekarang diinstal di basis data Anda. Artefak ini mencakup file kontrol dan kode untuk ekstensi, yang merupakan item yang harus ada sehingga ekstensi dapat dibuat menggunakan perintah `CREATE EXTENSION`. Dengan kata lain, Anda masih perlu membuat ekstensi agar fungsinya tersedia bagi pengguna basis data.

3. Untuk membuat ekstensi, gunakan perintah `CREATE EXTENSION` seperti yang Anda lakukan untuk ekstensi lainnya. Seperti ekstensi lainnya, pengguna basis data harus memiliki izin `CREATE` dalam basis data.

```
CREATE EXTENSION pg_distance;
```

4. Untuk menguji ekstensi TLE `pg_distance`, Anda dapat menggunakannya untuk menghitung Jarak Manhattan antara empat titik.

```
labdb=> SELECT manhattan_dist(1, 1, 5, 5);
```

```
8
```

Untuk menghitung Jarak Euclidean antara kumpulan titik yang sama, Anda dapat menggunakan berikut.

```
labdb=> SELECT euclidean_dist(1, 1, 5, 5);
5.656854249492381
```

Ekstensi `pg_distance` memuat fungsi dalam basis data dan membuatnya tersedia bagi setiap pengguna dengan izin pada basis data.

Mengubah ekstensi TLE

Untuk meningkatkan performa kueri untuk fungsi yang dikemas dalam ekstensi TLE ini, tambahkan dua atribut PostgreSQL berikut ke spesifikasinya.

- `IMMUTABLE` – Atribut `IMMUTABLE` memastikan bahwa pengoptimal kueri dapat menggunakan pengoptimalan untuk meningkatkan waktu respons kueri. Untuk informasi selengkapnya, lihat Kategori Volatilitas Fungsi dalam dokumentasi PostgreSQL.
- `PARALLEL SAFE` – Atribut `PARALLEL SAFE` adalah atribut lain yang memungkinkan PostgreSQL menjalankan fungsi dalam mode paralel. Untuk informasi selengkapnya, lihat CREATE FUNCTION dalam dokumentasi PostgreSQL.

Dalam contoh berikut, Anda dapat melihat bagaimana fungsi `pgtle.install_update_path` digunakan untuk menambahkan atribut ini ke setiap fungsi guna membuat ekstensi TLE `pg_distance` versi `0.2`. Untuk informasi selengkapnya tentang fungsi ini, lihat pgtle.install_update_path. Anda harus memiliki peran `pgtle_admin` untuk melakukan tugas ini.

Untuk memperbarui ekstensi TLE yang ada dan menentukan versi default

1. Hubungkan ke instans penulis klaster DB Aurora PostgreSQL Anda menggunakan `psql` atau alat klien lainnya, seperti pgAdmin.

   ```
   psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
   --port=5432 --username=postgres --password --dbname=labdb
   ```

2. Buat ekstensi TLE yang ada dengan menyalin kode berikut dan menempelkannya ke konsol sesi `psql` Anda.

```
SELECT pgtle.install_update_path
(
 'pg_distance',
 '0.1',
 '0.2',
$_pg_tle_$
    CREATE OR REPLACE FUNCTION dist(x1 float8, y1 float8, x2 float8, y2 float8,
 norm int)
    RETURNS float8
    AS $$
      SELECT (abs(x2 - x1) ^ norm + abs(y2 - y1) ^ norm) ^ (1::float8 / norm);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION manhattan_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 1);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;

    CREATE OR REPLACE FUNCTION euclidean_dist(x1 float8, y1 float8, x2 float8, y2
 float8)
    RETURNS float8
    AS $$
      SELECT dist(x1, y1, x2, y2, 2);
    $$ LANGUAGE SQL IMMUTABLE PARALLEL SAFE;
$_pg_tle_$
);
```

Anda akan melihat hasil yang mirip dengan berikut ini.

```
install_update_path
---------------------
 t
(1 row)
```

Anda dapat menjadikan versi ekstensi ini sebagai versi default, sehingga pengguna basis data tidak perlu menentukan versi saat mereka membuat atau memperbarui ekstensi di basis data mereka.

3. Untuk menentukan bahwa versi modifikasi (versi 0.2) ekstensi TLE Anda adalah versi default, gunakan fungsi `pgtle.set_default_version` seperti yang ditunjukkan pada contoh berikut.

```
SELECT pgtle.set_default_version('pg_distance', '0.2');
```

   Untuk informasi selengkapnya tentang fungsi ini, lihat pgtle.set_default_version.

4. Dengan kode yang diterapkan, Anda dapat memperbarui ekstensi TLE yang diinstal seperti biasa, dengan menggunakan perintah `ALTER EXTENSION ... UPDATE`, seperti yang ditunjukkan di sini:

```
ALTER EXTENSION pg_distance UPDATE;
```

Menghapus ekstensi TLE dari basis data

Anda dapat menghapus ekstensi TLE Anda dengan menggunakan perintah `DROP EXTENSION` dengan cara yang sama seperti yang Anda lakukan untuk ekstensi PostgreSQL lainnya. Menghapus ekstensi tidak menghapus file penginstalan yang membentuk ekstensi, yang memungkinkan pengguna membuat ulang ekstensi. Untuk menghapus ekstensi dan file penginstalannya, lakukan proses dua langkah berikut.

Untuk menghilangkan ekstensi TLE dan menghapus file penginstalannya

1. Gunakan `psql` atau alat klien lain untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=dbname
```

2. Hilangkan ekstensi seperti yang Anda lakukan pada ekstensi PostgreSQL.

```
DROP EXTENSION your-TLE-extension
```

   Misalnya, jika Anda membuat ekstensi `pg_distance` seperti yang dijelaskan dalamContoh: Membuat ekstensi bahasa tepercaya menggunakan SQL, Anda dapat menghilangkan ekstensi sebagai berikut.

```
DROP EXTENSION pg_distance;
```

Anda melihat output yang mengonfirmasi bahwa ekstensi telah dihilangkan, sebagai berikut.

```
DROP EXTENSION
```

Pada titik ini, ekstensi tidak lagi aktif dalam basis data. Namun, file penginstalan dan file kontrolnya masih tersedia di basis data, sehingga pengguna basis data dapat membuat ekstensi lagi jika mereka menghendaki.

- Jika ingin membiarkan file ekstensi tetap utuh sehingga pengguna basis data dapat membuat ekstensi TLE, Anda dapat berhenti di sini.
- Jika Anda ingin menghapus semua file yang membentuk ekstensi, lanjutkan ke langkah berikutnya.

3. Untuk menghapus semua file penginstalan untuk ekstensi Anda, gunakan fungsi `pgtle.uninstall_extension`. Fungsi ini menghapus semua kode dan file kontrol untuk ekstensi Anda.

```
SELECT pgtle.uninstall_extension('your-tle-extension-name');
```

Misalnya, untuk menghapus semua file penginstalan `pg_distance`, gunakan perintah berikut.

```
SELECT pgtle.uninstall_extension('pg_distance');
 uninstall_extension
---------------------
 t
(1 row)
```

Meng-uninstal Ekstensi Bahasa Tepercaya untuk PostgreSQL

Jika tidak ingin lagi membuat ekstensi TLE Anda sendiri menggunakan TLE, Anda dapat menghilangkan ekstensi `pg_tle` dan menghapus semua artefak. Tindakan ini mencakup menghilangkan ekstensi TLE apa pun dalam basis data dan menghilangkan skema `pgtle`.

Untuk menghilangkan ekstensi **`pg_tle`** dan skema dari basis data

1. Gunakan `psql` atau alat klien lain untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=dbname
```

2. Hilangkan ekstensi `pg_tle` dari basis data. Jika basis data memiliki ekstensi TLE Anda sendiri yang masih berjalan di basis data, Anda juga harus menghilangkan ekstensi tersebut. Untuk melakukannya, Anda dapat menggunakan kata kunci `CASCADE`, seperti yang ditunjukkan berikut ini.

   ```
   DROP EXTENSION pg_tle CASCADE;
   ```

   Jika ekstensi `pg_tle` masih tidak aktif dalam basis data, Anda tidak perlu menggunakan kata kunci `CASCADE`.

3. Hilangkan skema `pgtle`. Tindakan ini akan menghapus semua fungsi manajemen dari basis data.

   ```
   DROP SCHEMA pgtle CASCADE;
   ```

   Perintah ini menampilkan hasil berikut setelah proses selesai.

   ```
   DROP SCHEMA
   ```

   Ekstensi `pg_tle`, skema dan fungsinya, serta semua artefak dihapus. Untuk membuat ekstensi baru menggunakan TLE, lanjutkan proses penyiapan lagi. Untuk informasi selengkapnya, lihat Menyiapkan Ekstensi Bahasa Tepercaya di klaster DB Aurora PostgreSQL Anda.

Menggunakan hook PostgreSQL dengan ekstensi TLE

Hook adalah mekanisme callback yang tersedia di PostgreSQL yang memungkinkan pengembang memanggil fungsi kustom atau rutinitas lainnya selama operasi basis data reguler. Kit pengembangan TLE mendukung hook PostgreSQL sehingga Anda dapat mengintegrasikan fungsi kustom dengan perilaku PostgreSQL saat runtime. Misalnya, Anda dapat menggunakan hook untuk mengaitkan proses autentikasi dengan kode kustom Anda sendiri, atau mengubah proses perencanaan dan eksekusi kueri untuk kebutuhan spesifik Anda.

Ekstensi TLE Anda dapat menggunakan hook. Jika hook cakupannya global, ini berlaku di semua basis data. Oleh karena itu, jika ekstensi TLE Anda menggunakan hook global, Anda perlu membuat ekstensi TLE di semua basis data yang dapat diakses pengguna.

Saat menggunakan ekstensi `pg_tle` untuk membuat Ekstensi Bahasa Tepercaya Anda sendiri, Anda dapat menggunakan hook yang tersedia dari SQL API untuk membangun fungsi ekstensi. Anda harus mendaftarkan hook apa pun di `pg_tle`. Untuk beberapa hook, Anda mungkin juga perlu mengatur berbagai parameter konfigurasi. Misalnya, hook pemeriksaan `passcode` dapat diatur ke aktif, nonaktif, wajib. Untuk informasi selengkapnya tentang persyaratan khusus untuk hook `pg_tle` yang tersedia, lihat Referensi hook untuk Ekstensi Bahasa Tepercaya untuk PostgreSQL.

Contoh: Membuat ekstensi yang menggunakan hook PostgreSQL

Contoh yang dibahas di bagian ini menggunakan hook PostgreSQL untuk memeriksa kata sandi yang diberikan selama operasi SQL tertentu dan mencegah pengguna basis data mengatur kata sandi mereka ke salah satu basis data yang tercantum dalam tabel `password_check.bad_passwords`. Tabel berisi sepuluh besar pilihan kata sandi yang paling umum digunakan, tetapi mudah dipecahkan.

Untuk menyiapkan contoh ini di klaster DB Aurora PostgreSQL, Anda harus sudah menginstal Ekstensi Bahasa Tepercaya. Untuk detailnya, lihat Menyiapkan Ekstensi Bahasa Tepercaya di klaster DB Aurora PostgreSQL Anda.

Untuk menyiapkan contoh hook pemeriksaan kata sandi

1. Gunakan `psql` untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL.

```
psql --host=db-instance-123456789012.aws-region.rds.amazonaws.com
--port=5432 --username=postgres --password --dbname=labdb
```

2. Salin kode dari Daftar kode hook pemeriksaan kata sandi dan tempel ke basis data Anda.

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;
```

```
 CREATE TABLE password_check.bad_passwords (plaintext) AS
 VALUES
   ('123456'),
   ('password'),
   ('12345678'),
   ('qwerty'),
   ('123456789'),
   ('12345'),
   ('1234'),
   ('111111'),
   ('1234567'),
   ('dragon');
 CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

 CREATE FUNCTION password_check.passcheck_hook(username text, password text,
password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
 RETURNS void AS $$
   DECLARE
     invalid bool := false;
   BEGIN
     IF password_type = 'PASSWORD_TYPE_MD5' THEN
       SELECT EXISTS(
         SELECT 1
         FROM password_check.bad_passwords bp
         WHERE ('md5' || md5(bp.plaintext || username)) = password
       ) INTO invalid;
       IF invalid THEN
         RAISE EXCEPTION 'Cannot use passwords from the common password
dictionary';
       END IF;
     ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
       SELECT EXISTS(
         SELECT 1
         FROM password_check.bad_passwords bp
         WHERE bp.plaintext = password
       ) INTO invalid;
       IF invalid THEN
         RAISE EXCEPTION 'Cannot use passwords from the common common password
dictionary';
       END IF;
     END IF;
   END
 $$ LANGUAGE plpgsql SECURITY DEFINER;
```

```
  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

Setelah ekstensi dimuat ke basis data, Anda akan melihat output seperti berikut.

```
 install_extension
-------------------
 t
(1 row)
```

3. Jika masih terhubung ke basis data, Anda kini dapat membuat ekstensi.

```
CREATE EXTENSION my_password_check_rules;
```

4. Anda dapat mengonfirmasi bahwa ekstensi telah dibuat dalam basis data dengan menggunakan metacommand `psql` berikut.

```
\dx
                        List of installed extensions
          Name           | Version |   Schema   |
 Description
-------------------------+---------+------------
+-------------------------------------------------------------
 my_password_check_rules | 1.0     | public     | Prevent use of any of the top-ten
 most common bad passwords
 pg_tle                  | 1.0.1   | pgtle      | Trusted-Language Extensions for
 PostgreSQL
 plpgsql                 | 1.0     | pg_catalog | PL/pgSQL procedural language
(3 rows)
```

5. Buka sesi terminal lain yang ingin digunakan dengan AWS CLI. Anda perlu mengubah grup parameter DB kustom untuk mengaktifkan hook pemeriksaan kata sandi. Untuk melakukannya, gunakan perintah [modify-db-parameter-group](#)CLI seperti yang ditunjukkan pada contoh berikut.

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
```

```
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

Mungkin diperlukan waktu beberapa menit agar perubahan pada pengaturan grup parameter diterapkan. Akan tetapi, parameter ini bersifat dinamis, jadi Anda tidak perlu memulai ulang instans penulis klaster DB Aurora PostgreSQL untuk menerapkan pengaturan.

6. Buka sesi `psql` dan kirim kueri ke basis data untuk memverifikasi bahwa hook password_check telah diaktifkan.

```
labdb=> SHOW pgtle.enable_password_check;
pgtle.enable_password_check
----------------------------
on
(1 row)
```

Hook password_check kini aktif. Anda dapat mengujinya dengan membuat peran baru dan menggunakan salah satu kata sandi yang buruk, seperti pada contoh berikut.

```
CREATE ROLE test_role PASSWORD 'password';
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 21 at RAISE
SQL statement "SELECT password_check.passcheck_hook(
    $1::pg_catalog.text,
    $2::pg_catalog.text,
    $3::pgtle.password_types,
    $4::pg_catalog.timestamptz,
    $5::pg_catalog.bool)"
```

Output ini telah diformat agar mudah dibaca.

Contoh berikut menunjukkan bahwa perilaku `\password` metacommand interaktif `pgsql` juga dipengaruhi oleh hook password_check.

```
postgres=> SET password_encryption TO 'md5';
SET
postgres=> \password
Enter new password for user "postgres":*****
Enter it again:*****
```

```
ERROR:  Cannot use passwords from the common password dictionary
CONTEXT:  PL/pgSQL function
 password_check.passcheck_hook(text,text,pgtle.password_types,timestamp with time
 zone,boolean) line 12 at RAISE
SQL statement "SELECT password_check.passcheck_hook($1::pg_catalog.text,
 $2::pg_catalog.text, $3::pgtle.password_types, $4::pg_catalog.timestamptz,
 $5::pg_catalog.bool)"
```

Anda dapat menghilangkan ekstensi TLE ini dan meng-uninstal file sumbernya jika menghendaki. Untuk informasi selengkapnya, lihat Menghapus ekstensi TLE dari basis data.

Daftar kode hook pemeriksaan kata sandi

Kode contoh yang ditampilkan di sini menentukan spesifikasi untuk ekstensi TLE `my_password_check_rules`. Jika kode ini disalin dan ditempelkan ke basis data, kode untuk ekstensi `my_password_check_rules` akan dimuat ke dalam basis data, dan hook `password_check` akan didaftarkan untuk digunakan oleh ekstensi.

```
SELECT pgtle.install_extension (
  'my_password_check_rules',
  '1.0',
  'Do not let users use the 10 most commonly used passwords',
$_pgtle_$
  CREATE SCHEMA password_check;
  REVOKE ALL ON SCHEMA password_check FROM PUBLIC;
  GRANT USAGE ON SCHEMA password_check TO PUBLIC;

  CREATE TABLE password_check.bad_passwords (plaintext) AS
  VALUES
    ('123456'),
    ('password'),
    ('12345678'),
    ('qwerty'),
    ('123456789'),
    ('12345'),
    ('1234'),
    ('111111'),
    ('1234567'),
    ('dragon');
  CREATE UNIQUE INDEX ON password_check.bad_passwords (plaintext);

  CREATE FUNCTION password_check.passcheck_hook(username text, password text,
 password_type pgtle.password_types, valid_until timestamptz, valid_null boolean)
```

```
  RETURNS void AS $$
    DECLARE
      invalid bool := false;
    BEGIN
      IF password_type = 'PASSWORD_TYPE_MD5' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE ('md5' || md5(bp.plaintext || username)) = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common password dictionary';
        END IF;
      ELSIF password_type = 'PASSWORD_TYPE_PLAINTEXT' THEN
        SELECT EXISTS(
          SELECT 1
          FROM password_check.bad_passwords bp
          WHERE bp.plaintext = password
        ) INTO invalid;
        IF invalid THEN
          RAISE EXCEPTION 'Cannot use passwords from the common common password
 dictionary';
        END IF;
      END IF;
    END
  $$ LANGUAGE plpgsql SECURITY DEFINER;

  GRANT EXECUTE ON FUNCTION password_check.passcheck_hook TO PUBLIC;

  SELECT pgtle.register_feature('password_check.passcheck_hook', 'passcheck');
$_pgtle_$
);
```

Referensi fungsi untuk Ekstensi Bahasa Tepercaya untuk PostgreSQL

Lihat dokumentasi referensi berikut tentang fungsi yang tersedia di Ekstensi Bahasa Tepercaya untuk PostgreSQL. Gunakan fungsi-fungsi ini untuk menginstal, mendaftarkan, memperbarui, dan mengelola ekstensi TLE, yaitu ekstensi PostgreSQL yang Anda kembangkan menggunakan kit pengembangan Ekstensi Bahasa Tepercaya.

Topik

- pgtle.available_extensions

- pgtle.available_extension_versions
- pgtle.extension_update_paths
- pgtle.install_extension
- pgtle.install_update_path
- pgtle.register_feature
- pgtle.register_feature_if_not_exists
- pgtle.set_default_version
- pgtle.uninstall_extension (nama)
- pgtle.uninstall_extension (nama, versi)
- pgtle.uninstall_extension_if_exists
- pgtle.uninstall_update_path
- pgtle.uninstall_update_path_if_exists
- pgtle.unregister_feature
- pgtle.unregister_feature_if_exists

pgtle.available_extensions

Fungsi `pgtle.available_extensions` adalah fungsi penampilan set. Fungsi ini menampilkan semua ekstensi TLE yang tersedia dalam basis data. Setiap baris yang ditampilkan berisi informasi tentang ekstensi TLE tunggal.

Prototipe fungsi

```
pgtle.available_extensions()
```

Peran

Tidak ada.

Argumen

Tidak ada.

Output

- `name` – Nama ekstensi TLE.

- `default_version` – Versi ekstensi TLE untuk digunakan ketika `CREATE EXTENSION` dipanggil tanpa versi yang ditentukan.
- `description` – Penjelasan yang lebih mendetail tentang ekstensi TLE.

Contoh penggunaan

```
SELECT * FROM pgtle.available_extensions();
```

pgtle.available_extension_versions

Fungsi `available_extension_versions` adalah fungsi penampilan set. Fungsi ini menampilkan daftar semua ekstensi TLE dan versinya yang tersedia. Setiap baris berisi informasi tentang versi tertentu dari ekstensi TLE yang diberikan, termasuk apakah versi tersebut memerlukan peran tertentu.

Prototipe fungsi

```
pgtle.available_extension_versions()
```

Peran

Tidak ada.

Argumen

Tidak ada.

Output

- `name` – Nama ekstensi TLE.
- `version` – Versi ekstensi TLE.
- `superuser` – Nilai ini selalu `false` untuk ekstensi TLE Anda. Izin yang diperlukan untuk membuat ekstensi TLE atau memperbaruinya sama dengan izin untuk membuat objek lain dalam basis data tertentu.
- `trusted` – Nilai ini selalu `false` untuk ekstensi TLE.
- `relocatable` – Nilai ini selalu `false` untuk ekstensi TLE.
- `schema` – Menentukan nama skema di mana ekstensi TLE diinstal.

- `requires` – Array yang berisi nama-nama ekstensi lain yang dibutuhkan oleh ekstensi TLE ini.
- `description` – Penjelasan mendetail tentang ekstensi TLE.

Lihat informasi selengkapnya tentang nilai output di Mengemas Objek Terkait ke dalam Ekstensi > File Ekstensi dalam dokumentasi PostgreSQL.

Contoh penggunaan

```
SELECT * FROM pgtle.available_extension_versions();
```

pgtle.extension_update_paths

Fungsi `extension_update_paths` adalah fungsi penampilan set. Fungsi ini menampilkan daftar semua jalur pembaruan yang mungkin untuk ekstensi TLE. Setiap baris menyertakan peningkatan atau penurunan versi yang tersedia untuk ekstensi TLE tersebut.

Prototipe fungsi

```
pgtle.extension_update_paths(name)
```

Peran

Tidak ada.

Argumen

`name` – Nama ekstensi TLE untuk mendapatkan jalur peningkatan.

Output

- `source` – Versi sumber untuk pembaruan.
- `target` – Versi target untuk pembaruan.
- `path` – Jalur peningkatan yang digunakan untuk memperbarui ekstensi TLE dari versi `source` ke versi `target`, misalnya, `0.1--0.2`.

Contoh penggunaan

```
SELECT * FROM pgtle.extension_update_paths('your-TLE');
```

pgtle.install_extension

Fungsi `install_extension` memungkinkan Anda menginstal artefak yang membentuk ekstensi TLE Anda di basis data. Selanjutnya, ekstensi ini dapat dibuat menggunakan perintah `CREATE EXTENSION`.

Prototipe fungsi

```
pgtle.install_extension(name text, version text, description text, ext text, requires
 text[] DEFAULT NULL::text[])
```

Peran

Tidak ada.

Argumen

- `name` – Nama ekstensi TLE. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.
- `version` – Versi ekstensi TLE.
- `description` – Penjelasan mendetail tentang ekstensi TLE. Deskripsi ini ditampilkan di kolom `comment` pada `pgtle.available_extensions()`.
- `ext` – Isi ekstensi TLE. Nilai ini berisi objek seperti fungsi.
- `requires` – Parameter opsional yang menentukan dependensi untuk ekstensi TLE ini. Ekstensi `pg_tle` secara otomatis ditambahkan sebagai dependensi.

Banyak dari argumen ini sama dengan yang disertakan dalam file kontrol ekstensi untuk menginstal ekstensi PostgreSQL pada sistem file instans PostgreSQL. Untuk informasi selengkapnya, lihat File Ekstensi dalam Mengemas Objek Terkait menjadi Ekstensi di dokumentasi PostgreSQL.

Output

Fungsi ini akan menampilkan `OK` jika berhasil, dan `NULL` jika terjadi kesalahan.

- `OK` – Ekstensi TLE telah berhasil diinstal di basis data.
- `NULL` – Ekstensi TLE belum berhasil diinstal di basis data.

Contoh penggunaan

```
SELECT pgtle.install_extension(
```

```
  'pg_tle_test',
  '0.1',
  'My first pg_tle extension',
$_pgtle_$
  CREATE FUNCTION my_test()
  RETURNS INT
  AS $$
    SELECT 42;
  $$ LANGUAGE SQL IMMUTABLE;
$_pgtle_$
);
```

pgtle.install_update_path

Fungsi `install_update_path` menyediakan jalur pembaruan antara dua versi ekstensi TLE yang berbeda. Fungsi ini memungkinkan pengguna ekstensi TLE Anda untuk memperbarui versinya dengan menggunakan sintaks `ALTER EXTENSION ... UPDATE`.

Prototipe fungsi

```
pgtle.install_update_path(name text, fromvers text, tovers text, ext text)
```

Peran

`pgtle_admin`

Argumen

- `name` – Nama ekstensi TLE. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.
- `fromvers` – Versi sumber ekstensi TLE untuk peningkatan.
- `tovers` – Versi tujuan ekstensi TLE untuk peningkatan.
- `ext` – Isi pembaruan. Nilai ini berisi objek seperti fungsi.

Output

Tidak ada.

Contoh penggunaan

```
SELECT pgtle.install_update_path('pg_tle_test', '0.1', '0.2',
```

```
  $_pgtle_$
    CREATE OR REPLACE FUNCTION my_test()
    RETURNS INT
    AS $$
      SELECT 21;
    $$ LANGUAGE SQL IMMUTABLE;
  $_pgtle_$
);
```

pgtle.register_feature

Fungsi `register_feature` menambahkan fitur PostgreSQL internal yang ditentukan ke tabel `pgtle.feature_info`. Hook PostgreSQL adalah contoh dari fitur PostgreSQL internal. Kit pengembangan Ekstensi Bahasa Tepercaya mendukung penggunaan hook PostgreSQL. Saat ini, fungsi ini mendukung fitur berikut.

- `passcheck` – Mendaftarkan hook pemeriksaan kata sandi dengan prosedur atau fungsi Anda yang menyesuaikan perilaku pemeriksaan kata sandi PostgreSQL.

Prototipe fungsi

```
pgtle.register_feature(proc regproc, feature pg_tle_feature)
```

Peran

`pgtle_admin`

Argumen

- `proc` – Nama prosedur atau fungsi yang disimpan untuk digunakan untuk fitur tersebut.
- `feature` – Nama fitur `pg_tle` (seperti `passcheck`) untuk mendaftar dengan fungsi.

Output

Tidak ada.

Contoh penggunaan

```
SELECT pgtle.register_feature('pw_hook', 'passcheck');
```

pgtle.register_feature_if_not_exists

Fungsi `pgtle.register_feature_if_not_exists` menambahkan fitur PostgreSQL yang ditentukan ke tabel `pgtle.feature_info` dan mengidentifikasi ekstensi TLE atau prosedur atau fungsi lain yang menggunakan fitur tersebut. Untuk informasi selengkapnya tentang hook dan Ekstensi Bahasa Tepercaya, lihat Menggunakan hook PostgreSQL dengan ekstensi TLE.

Prototipe fungsi

```
pgtle.register_feature_if_not_exists(proc regproc, feature pg_tle_feature)
```

Peran

`pgtle_admin`

Argumen

- `proc` – Nama prosedur atau fungsi tersimpan yang berisi logika (kode) untuk digunakan sebagai fitur bagi ekstensi TLE Anda. Misalnya, kode `pw_hook`.
- `feature` – Nama fitur PostgreSQL untuk mendaftar untuk fungsi TLE. Saat ini, satu-satunya fitur yang tersedia adalah hook `passcheck`. Untuk informasi selengkapnya, lihat Hook pemeriksaan kata sandi (passcheck).

Output

Menampilkan `true` setelah mendaftarkan fitur untuk ekstensi yang ditentukan. Menampilkan `false` jika fitur sudah terdaftar.

Contoh penggunaan

```
SELECT pgtle.register_feature_if_not_exists('pw_hook', 'passcheck');
```

pgtle.set_default_version

Fungsi `set_default_version` memungkinkan Anda menentukan `default_version` untuk ekstensi TLE Anda. Anda dapat menggunakan fungsi ini untuk menentukan jalur peningkatan dan menetapkan versi sebagai default untuk ekstensi TLE Anda. Ketika pengguna basis data menentukan ekstensi TLE Anda dalam perintah `CREATE EXTENSION` dan `ALTER EXTENSION ... UPDATE`, versi ekstensi TLE Anda ini dibuat dalam basis data untuk pengguna tersebut.

Fungsi ini menampilkan `true` jika berhasil. Jika ekstensi TLE yang ditentukan dalam argumen `name` tidak ada, fungsi akan menampilkan pesan kesalahan. Demikian pula, jika `version` ekstensi TLE tidak ada, fungsi akan menampilkan pesan kesalahan.

Prototipe fungsi

```
pgtle.set_default_version(name text, version text)
```

Peran

`pgtle_admin`

Argumen

- `name` – Nama ekstensi TLE. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.
- `version` – Versi ekstensi TLE untuk mengatur default.

Output

- `true` – Ketika pengaturan versi default berhasil, fungsi menampilkan `true`.
- `ERROR` – Menampilkan pesan kesalahan jika ekstensi TLE dengan nama atau versi tertentu tidak ada.

Contoh penggunaan

```
SELECT * FROM pgtle.set_default_version('my-extension', '1.1');
```

pgtle.uninstall_extension (nama)

Fungsi `uninstall_extension` menghapus semua versi ekstensi TLE dari basis data. Fungsi ini mencegah panggilan `CREATE EXTENSION` mendatang untuk menginstal ekstensi TLE. Jika ekstensi TLE tidak ada dalam basis data, pesan kesalahan akan muncul.

Fungsi `uninstall_extension` tidak akan menghapus ekstensi TLE yang saat ini aktif dalam basis data. Untuk menghapus ekstensi TLE yang saat ini aktif, Anda perlu memanggil `DROP EXTENSION` secara eksplisit untuk menghapusnya.

Prototipe fungsi

```
pgtle.uninstall_extension(extname text)
```

Peran

`pgtle_admin`

Argumen

- `extname` – Nama ekstensi TLE yang akan dihapus instalasinya. Nama ini sama dengan yang digunakan `CREATE EXTENSION` untuk memuat ekstensi TLE untuk digunakan dalam basis data tertentu.

Output

Tidak ada.

Contoh penggunaan

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test');
```

pgtle.uninstall_extension (nama, versi)

Fungsi `uninstall_extension(name, version)` menghapus versi ekstensi TLE tertentu dari basis data. Fungsi ini mencegah `CREATE EXTENSION` dan `ALTER EXTENSION` dari menginstal atau memperbarui ekstensi TLE ke versi yang ditentukan. Fungsi ini juga menghapus semua jalur pembaruan untuk ekstensi TLE tertentu. Fungsi ini tidak akan menghapus instalasi ekstensi TLE jika ekstensi tersebut saat ini aktif dalam basis data. Anda harus secara eksplisit memanggil `DROP EXTENSION` untuk menghapus ekstensi TLE. Untuk menghapus semua versi ekstensi TLE, lihat pgtle.uninstall_extension (nama).

Prototipe fungsi

```
pgtle.uninstall_extension(extname text, version text)
```

Peran

`pgtle_admin`

Argumen

- `extname` – Nama ekstensi TLE. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.
- `version` – Versi ekstensi TLE untuk menghapus instalasi dari basis data.

Output

Tidak ada.

Contoh penggunaan

```
SELECT * FROM pgtle.uninstall_extension('pg_tle_test', '0.2');
```

pgtle.uninstall_extension_if_exists

Fungsi `uninstall_extension_if_exists` menghapus semua versi ekstensi TLE dari basis data tertentu. Jika ekstensi TLE tidak ada, fungsi akan menampilkan pemberitahuan secara diam-diam (tidak ada pesan kesalahan yang muncul). Jika ekstensi yang ditentukan saat ini aktif dalam basis data, fungsi ini tidak akan menghapusnya. Anda harus secara eksplisit memanggil `DROP EXTENSION` untuk menghapus ekstensi TLE sebelum menggunakan fungsi ini untuk menghapus instalasi artefaknya.

Prototipe fungsi

```
pgtle.uninstall_extension_if_exists(extname text)
```

Peran

`pgtle_admin`

Argumen

- `extname` – Nama ekstensi TLE. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.

Output

Fungsi `uninstall_extension_if_exists` menampilkan `true` setelah menghapus instalasi ekstensi yang ditentukan. Jika ekstensi yang ditentukan tidak ada, fungsi akan menampilkan `false`.

- `true` – Menampilkan `true` setelah menghapus instalasi ekstensi TLE.

- `false` – Menampilkan `false` ketika ekstensi TLE tidak ada dalam basis data.

Contoh penggunaan

```
SELECT * FROM pgtle.uninstall_extension_if_exists('pg_tle_test');
```

pgtle.uninstall_update_path

Fungsi `uninstall_update_path` menghapus jalur pembaruan yang ditentukan dari ekstensi TLE. Fungsi ini mencegah `ALTER EXTENSION ... UPDATE TO` dari menggunakan ekstensi ini sebagai jalur pembaruan.

Jika ekstensi TLE saat ini digunakan oleh salah satu versi di jalur pembaruan ini, ekstensi TLE tetap ada di basis data.

Jika jalur pembaruan yang ditentukan tidak ada, fungsi ini akan memunculkan pesan kesalahan.

Prototipe fungsi

```
pgtle.uninstall_update_path(extname text, fromvers text, tovers text)
```

Peran

`pgtle_admin`

Argumen

- `extname` – Nama ekstensi TLE. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.
- `fromvers` – Versi sumber ekstensi TLE yang digunakan pada jalur pembaruan.
- `tovers` – Versi tujuan ekstensi TLE yang digunakan pada jalur pembaruan.

Output

Tidak ada.

Contoh penggunaan

```
SELECT * FROM pgtle.uninstall_update_path('pg_tle_test', '0.1', '0.2');
```

pgtle.uninstall_update_path_if_exists

Fungsi `uninstall_update_path_if_exists` mirip dengan `uninstall_update_path` yang menghapus jalur pembaruan yang ditentukan dari ekstensi TLE. Namun, jika jalur pembaruan tidak ada, fungsi ini tidak memunculkan pesan kesalahan. Sebaliknya, fungsi menampilkan `false`.

Prototipe fungsi

```
pgtle.uninstall_update_path_if_exists(extname text, fromvers text, tovers text)
```

Peran

`pgtle_admin`

Argumen

- `extname` – Nama ekstensi TLE. Nilai ini digunakan saat memanggil `CREATE EXTENSION`.
- `fromvers` – Versi sumber ekstensi TLE yang digunakan pada jalur pembaruan.
- `tovers` – Versi tujuan ekstensi TLE yang digunakan pada jalur pembaruan.

Output

- `true` – Fungsi telah berhasil memperbarui jalur untuk ekstensi TLE.
- `false` – Fungsi tidak dapat memperbarui jalur untuk ekstensi TLE.

Contoh penggunaan

```
SELECT * FROM pgtle.uninstall_update_path_if_exists('pg_tle_test', '0.1', '0.2');
```

pgtle.unregister_feature

Fungsi `unregister_feature` menyediakan cara untuk menghapus fungsi yang terdaftar untuk menggunakan fitur `pg_tle`, seperti hook. Untuk informasi tentang cara mendaftarkan fitur, lihatpgtle.register_feature.

Prototipe fungsi

```
pgtle.unregister_feature(proc regproc, feature pg_tle_features)
```

Peran

`pgtle_admin`

Argumen

- `proc` – Nama fungsi tersimpan untuk mendaftar dengan fitur `pg_tle`.
- `feature` – Nama fitur `pg_tle` untuk mendaftar dengan fungsi. Misalnya, `passcheck` adalah fitur yang dapat didaftarkan untuk digunakan oleh ekstensi bahasa tepercaya yang Anda kembangkan. Untuk informasi selengkapnya, lihat Hook pemeriksaan kata sandi (passcheck).

Output

Tidak ada.

Contoh penggunaan

```
SELECT * FROM pgtle.unregister_feature('pw_hook', 'passcheck');
```

pgtle.unregister_feature_if_exists

Fungsi `unregister_feature` menyediakan cara untuk menghapus fungsi yang terdaftar untuk menggunakan fitur `pg_tle`, seperti hook. Untuk informasi selengkapnya, lihat Menggunakan hook PostgreSQL dengan ekstensi TLE. Menampilkan `true` setelah berhasil membatalkan pendaftaran fitur. Menampilkan `false` jika fitur tidak terdaftar.

Untuk mengetahui informasi tentang mendaftarkan fitur `pg_tle` untuk ekstensi TLE Anda, lihat pgtle.register_feature.

Prototipe fungsi

```
pgtle.unregister_feature_if_exists('proc regproc', 'feature pg_tle_features')
```

Peran

`pgtle_admin`

Argumen

- `proc` – Nama fungsi tersimpan yang terdaftar untuk menyertakan fitur `pg_tle`.

- `feature` – Nama fitur `pg_tle` yang terdaftar dengan ekstensi bahasa tepercaya.

Output

Menampilkan `true` atau `false`, sebagai berikut.

- `true` – Fungsi telah berhasil membatalkan pendaftaran fitur dari ekstensi.
- `false` – Fungsi tidak dapat membatalkan pendaftaran fitur dari ekstensi TLE.

Contoh penggunaan

```
SELECT * FROM pgtle.unregister_feature_if_exists('pw_hook', 'passcheck');
```

Referensi hook untuk Ekstensi Bahasa Tepercaya untuk PostgreSQL

Ekstensi Bahasa Tepercaya untuk PostgreSQL mendukung hook PostgreSQL. Hook adalah mekanisme panggilan balik internal yang tersedia bagi pengembang untuk memperluas fungsionalitas inti PostgreSQL. Dengan menggunakan hook, pengembang dapat mengimplementasikan fungsi atau prosedurnya sendiri untuk digunakan dalam berbagai operasi basis data, sehingga mengubah perilaku PostgreSQL dalam beberapa cara. Misalnya, Anda dapat menggunakan hook `passcheck` untuk menyesuaikan cara PostgreSQL menangani kata sandi yang diberikan saat membuat atau mengubah kata sandi untuk pengguna (peran).

Lihat dokumentasi berikut untuk mempelajari hook yang tersedia untuk ekstensi TLE Anda.

Topik

- Hook pemeriksaan kata sandi (passcheck)

Hook pemeriksaan kata sandi (passcheck)

Hook `passcheck` digunakan untuk menyesuaikan perilaku PostgreSQL selama proses pemeriksaan kata sandi untuk perintah SQL dan metacommand `psql` berikut.

- `CREATE ROLE` *`username`* `...PASSWORD` – Untuk informasi selengkapnya, lihat CREATE ROLE dalam dokumentasi PostgreSQL.
- `ALTER ROLE` *`username`*`...PASSWORD` – Untuk informasi selengkapnya, lihat ALTER ROLE dalam dokumentasi PostgreSQL.

- `\password` *`username`* – Metacommand `psql` interaktif ini secara aman mengubah kata sandi untuk pengguna yang ditentukan dengan hashing kata sandi sebelum menggunakan sintaks `ALTER ROLE ... PASSWORD` secara transparan. Metacommand adalah pembungkus aman untuk perintah `ALTER ROLE ... PASSWORD`, sehingga hook dapat diterapkan untuk perilaku metacommand `psql`.

Sebagai contoh, lihat Daftar kode hook pemeriksaan kata sandi.

Prototipe fungsi

```
passcheck_hook(username text, password text, password_type pgtle.password_types,
 valid_until timestamptz, valid_null boolean)
```

Argumen

Fungsi hook `passcheck` memiliki argumen berikut.

- `username` – Nama (sebagai teks) dari peran (nama pengguna) yang mengatur kata sandi.
- `password` – Kata sandi yang di-hash atau teks biasa. Kata sandi yang dimasukkan harus sesuai dengan jenis yang ditentukan dalam `password_type`.
- `password_type` – Menentukan format `pgtle.password_type` kata sandi. Format ini dapat berupa salah satu opsi berikut.
  - `PASSWORD_TYPE_PLAINTEXT` – Kata sandi teks biasa.
  - `PASSWORD_TYPE_MD5` – Kata sandi yang telah di-hash menggunakan algoritma MD5 (message digest 5).
  - `PASSWORD_TYPE_SCRAM_SHA_256` – Kata sandi yang telah di-hash menggunakan algoritma SCRAM-SHA-256.
- `valid_until` – Menentukan waktu kapan kata sandi menjadi tidak valid. Argumen ini opsional. Jika Anda menggunakan argumen ini, tentukan waktu sebagai nilai `timestamptz`.
- `valid_null` – Jika Boolean ini diatur ke `true`, opsi `valid_until` diatur ke `NULL`.

Konfigurasi

Fungsi `pgtle.enable_password_check` mengontrol apakah hook passcheck aktif. Hook passcheck memiliki tiga opsi pengaturan.

- `off` – Menonaktifkan hook pemeriksaan kata sandi `passcheck`. Ini adalah nilai default.

- on – Mengaktifkan hook pemeriksaan kata sandi `passcode` sehingga kata sandi diperiksa berdasarkan tabel.
- `require` – Mewajibkan penentuan hook pemeriksaan kata sandi.

Catatan penggunaan

Untuk mengaktifkan atau menonaktifkan hook `passcheck`, Anda perlu memodifikasi grup parameter DB kustom untuk instans penulis klaster DB Aurora PostgreSQL Anda.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-parameter-group \
    --region aws-region \
    --db-parameter-group-name your-custom-parameter-group \
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-parameter-group ^
    --region aws-region ^
    --db-parameter-group-name your-custom-parameter-group ^
    --parameters
 "ParameterName=pgtle.enable_password_check,ParameterValue=on,ApplyMethod=immediate"
```

Referensi Amazon Aurora PostgreSQL

Topik

- Kolasi Aurora PostgreSQL untuk EBCDIC dan migrasi mainframe lainnya
- Kolasi yang didukung di Aurora PostgreSQL
- Referensi fungsi Aurora PostgreSQL
- Parameter Amazon Aurora PostgreSQL
- Peristiwa tunggu Amazon Aurora PostgreSQL

Kolasi Aurora PostgreSQL untuk EBCDIC dan migrasi mainframe lainnya

Migrasi aplikasi mainframe ke platform baru seperti AWS idealnya mempertahankan perilaku aplikasi. Untuk mempertahankan perilaku aplikasi pada platform baru persis seperti pada mainframe mengharuskan data yang dimigrasi dikolasi menggunakan aturan kolasi dan pengurutan yang sama. Misalnya, banyak solusi migrasi Db2 menggeser nilai nol ke u0180 (posisi Unicode 0180), jadi kolasi ini mengurutkan u0180 terlebih dahulu. Ini adalah salah satu contoh bagaimana kolasi dapat bervariasi dari sumber mainframe mereka dan mengapa perlu memilih kolasi yang memetakan lebih baik ke kolasi EBCDIC asli.

Aurora PostgreSQL 14.3 dan versi yang lebih tinggi menyediakan banyak kolasi ICU dan EBCDIC untuk mendukung migrasi tersebut ke AWS menggunakan layanan AWS Mainframe Modernization. Untuk mempelajari selengkapnya tentang layanan ini, lihat Apa itu AWS Mainframe Modernization?

Pada tabel berikut, Anda dapat menemukan kolasi yang disediakan Aurora PostgreSQL. Kolasi ini mengikuti aturan EBCDIC dan memastikan bahwa aplikasi mainframe berfungsi sama pada AWS seperti yang dilakukan di lingkungan mainframe. Nama kolasi mencakup halaman kode yang relevan, (cpnnnn), sehingga Anda dapat memilih kolasi yang sesuai untuk sumber mainframe Anda. Misalnya, gunakan `en-US-cp037-x-icu` untuk mencapai perilaku kolasi data EBCDIC yang berasal dari aplikasi mainframe yang menggunakan kode halaman 037.

Kolasi EBCDIC	Kolasi AWS Blu Age	Kolasi AWS Micro Focus
da-DK-cp1142-x-icu	da-DK-cp1142b-x-icu	da-DK-cp1142m-x-icu
da-DK-cp277-x-icu	da-DK-cp277b-x-icu	–

Kolasi EBCDIC	Kolasi AWS Blu Age	Kolasi AWS Micro Focus
de-DE-cp1141-x-icu	de-DE-cp1141b-x-icu	de-DE-cp1141m-x-icu
de-DE-cp273-x-icu	de-DE-cp273b-x-icu	–
en-GB-cp1146-x-icu	en-GB-cp1146b-x-icu	en-GB-cp1146m-x-icu
en-GB-cp285-x-icu	en-GB-cp285b-x-icu	–
en-US-cp037-x-icu	en-US-cp037b-x-icu	–
en-US-cp1140-x-icu	en-US-cp1140b-x-icu	en-US-cp1140m-x-icu
es-ES-cp1145-x-icu	es-ES-cp1145b-x-icu	es-ES-cp1145m-x-icu
es-ES-cp284-x-icu	es-ES-cp284b-x-icu	–
fi-FI-cp1143-x-icu	fi-FI-cp1143b-x-icu	fi-FI-cp1143m-x-icu
fi-FI-cp278-x-icu	fi-FI-cp278b-x-icu	–
fr-FR-cp1147-x-icu	fr-FR-cp1147b-x-icu	fr-FR-cp1147m-x-icu
fr-FR-cp297-x-icu	fr-FR-cp297b-x-icu	–
it-IT-cp1144-x-icu	it-IT-cp1144b-x-icu	it-IT-cp1144m-x-icu
it-IT-cp280-x-icu	it-IT-cp280b-x-icu	–
nl-BE-cp1148-x-icu	nl-BE-cp1148b-x-icu	nl-BE-cp1148m-x-icu
nl-BE-cp500-x-icu	nl-BE-cp500b-x-icu	–

Untuk mempelajari selengkapnya tentang AWS Blu Age, lihat Tutorial: Runtime Terkelola untuk AWS Blu Age di Panduan Pengguna AWS Mainframe Modernization.

Untuk informasi selengkapnya tentang bekerja dengan AWS Micro Focus, lihat Tutorial: Runtime Terkelola untuk Micro Focus di Panduan Pengguna AWS Mainframe Modernization.

Untuk mempelajari selengkapnya tentang mengelola kolasi PostgreSQL, lihat Dukungan Kolasi dalam dokumentasi PostgreSQL.

Kolasi yang didukung di Aurora PostgreSQL

Kolasi adalah seperangkat aturan yang menentukan bagaimana string karakter yang disimpan dalam basis data diurutkan dan dibandingkan. Kolasi memiliki peran mendasar dalam sistem komputer dan dimasukkan sebagai bagian dari sistem operasi. Kolasi berubah dari waktu ke waktu ketika karakter baru ditambahkan ke bahasa atau ketika aturan urutan berubah.

Pustaka kolasi menentukan aturan dan algoritma khusus untuk kolasi. Pustaka kolasi paling populer yang digunakan dalam PostgreSQL adalah GNU C (glibc) dan komponen Internasionalisasi untuk Unicode (ICU). Secara default, Aurora PostgreSQL menggunakan kolasi glibc yang mencakup urutan karakter unicode untuk urutan karakter multi-byte.

Saat Anda membuat klaster DB Aurora PostgreSQL baru, ini akan memeriksa sistem operasi untuk kolasi yang tersedia. Parameter PostgreSQL dari perintah `CREATE DATABASE`, `LC_COLLATE`, dan `LC_CTYPE` digunakan untuk menentukan kolasi, yang merupakan kolasi default dalam basis data tersebut. Atau, Anda juga dapat menggunakan parameter `LOCALE` di `CREATE DATABASE` untuk menetapkan parameter ini. Parameter ini menentukan kolasi default untuk string karakter dalam basis data serta aturan untuk mengklasifikasikan karakter sebagai huruf, angka, atau simbol. Anda juga dapat memilih kolasi untuk digunakan pada kolom, indeks, atau kueri.

Aurora PostgreSQL bergantung pada pustaka glibc di sistem operasi untuk dukungan kolasi. Instans Aurora PostgreSQL diperbarui secara berkala dengan versi terbaru sistem operasi. Pembaruan ini terkadang menyertakan versi pustaka glibc yang lebih baru. Jarang sekali versi glibc yang lebih baru mengubah tata urutan atau kolasi beberapa karakter, yang dapat menyebabkan data diurutkan secara berbeda atau menghasilkan entri indeks yang tidak valid. Jika Anda menemukan masalah terkait tata urutan kolasi selama pembaruan, Anda mungkin perlu membuat ulang indeks.

Untuk mengurangi kemungkinan dampak pembaruan glibc, Aurora PostgreSQL kini menyertakan pustaka kolasi default independen. Pustaka kolasi ini tersedia di Aurora PostgreSQL 14.6, 13.9, 12.13, 11.18 dan rilis versi minor yang lebih baru. Pustaka ini kompatibel dengan glibc 2.26-59.amzn2, dan menyediakan stabilitas tata urutan untuk mencegah hasil kueri yang salah.

Referensi fungsi Aurora PostgreSQL

Berikut ini, Anda dapat menemukan daftar fungsi Aurora PostgreSQL yang tersedia untuk klaster DB Aurora Anda yang menjalankan mesin DB Aurora PostgreSQL-Compatible Edition. Fungsi Aurora

PostgreSQL ini merupakan tambahan dari fungsi PostgreSQL standar. Untuk informasi selengkapnya tentang fungsi PostgreSQL standar, lihat PostgreSQL–Fungsi dan Operator.

Ikhtisar

Anda dapat menggunakan fungsi berikut untuk instans DB Amazon RDS yang menjalankan Aurora PostgreSQL:

- aurora_db_instance_identifier
- aurora_ccm_status
- aurora_global_db_instance_status
- aurora_global_db_status
- aurora_list_builtins
- aurora_replica_status
- aurora_stat_activity
- aurora_stat_backend_waits
- aurora_stat_bgwriter
- aurora_stat_database
- aurora_stat_dml_activity
- aurora_stat_get_db_commit_latency
- aurora_stat_logical_wal_cache
- aurora_stat_memctx_usage
- aurora_stat_optimized_reads_cache
- aurora_stat_plans
- aurora_stat_reset_wal_cache
- aurora_stat_statement
- aurora_stat_system_waits
- aurora_stat_wait_event
- aurora_stat_wait_type
- aurora_version
- aurora_volume_logical_start_lsn
- aurora_wait_report

aurora_db_instance_identifier

Melaporkan nama instans DB yang terhubung dengan Anda.

Sintaksis

```
aurora_db_instance_identifier()
```

Argumen

Tidak ada

Jenis pengembalian

String VARCHAR

Catatan penggunaan

Fungsi ini menampilkan nama instans DB klaster Aurora Edisi Kompatibel PostgreSQL untuk klien basis data atau koneksi aplikasi Anda.

Fungsi ini tersedia dimulai dengan rilis Aurora PostgreSQL versi 13.7, 12.11, 11.16, dan 10.21 serta untuk semua versi lain yang lebih baru.

Contoh

Contoh berikut menampilkan hasil pemanggilan fungsi `aurora_db_instance_identifier`.

```
=> SELECT aurora_db_instance_identifier();
aurora_db_instance_identifier
------------------------------
 test-my-instance-name
```

Anda dapat menggabungkan hasil fungsi ini dengan fungsi `aurora_replica_status` untuk mendapatkan detail tentang instans DB untuk koneksi Anda. aurora_replica_status sendiri tidak memberikan tampilan instans DB mana yang Anda gunakan. Contoh kode berikut ini menunjukkan caranya.

```
=> SELECT *
```

```
    FROM aurora_replica_status() rt,
         aurora_db_instance_identifier() di
    WHERE rt.server_id = di;
-[ RECORD 1 ]----------------------+----------------------
server_id                          | test-my-instance-name
session_id                         | MASTER_SESSION_ID
durable_lsn                        | 88492069
highest_lsn_rcvd                   |
current_read_lsn                   |
cur_replay_latency_in_usec         |
active_txns                        |
is_current                         | t
last_transport_error               | 0
last_error_timestamp               |
last_update_timestamp              | 2022-06-03 11:18:25+00
feedback_xmin                      |
feedback_epoch                     |
replica_lag_in_msec                |
log_stream_speed_in_kib_per_second | 0
log_buffer_sequence_number         | 0
oldest_read_view_trx_id            |
oldest_read_view_lsn               |
pending_read_ios                   | 819
```

aurora_ccm_status

Menampilkan status pengelola cache klaster.

Sintaksis

```
aurora_ccm_status()
```

Argumen

Tidak ada.

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `buffers_sent_last_minute` – Jumlah buffer yang dikirim ke pembaca yang ditunjuk dalam beberapa menit terakhir.

- `buffers_found_last_minute` – Jumlah buffer yang sering diakses diidentifikasi selama beberapa menit terakhir.
- `buffers_sent_last_scan` – Jumlah buffer yang dikirim ke pembaca yang ditunjuk selama pemindaian lengkap cache buffer.
- `buffers_found_last_scan` – Jumlah buffer yang sering diakses yang dikirim selama pemindaian lengkap cache buffer. Buffer yang sudah di-cache di pembaca yang ditunjuk tidak akan dikirimkan.
- `buffers_sent_current_scan` – Jumlah buffer yang dikirim selama pemindaian saat ini.
- `buffers_found_current_scan` – Jumlah buffer yang sering diakses yang diidentifikasi dalam pemindaian saat ini.
- `current_scan_progress` – Jumlah buffer yang dikunjungi sejauh ini selama pemindaian saat ini.

Catatan penggunaan

Anda dapat menggunakan fungsi ini untuk memeriksa dan memantau fitur manajemen cache klaster (CCM). Fungsi ini hanya berfungsi jika CCM aktif di klaster DB Aurora PostgreSQL Anda. Untuk menggunakan fungsi ini, Anda menghubungkan ke instans Write DB di klaster Aurora PostgreSQL Anda.

Anda mengaktifkan CCM untuk klaster DB Aurora PostgreSQL dengan mengatur `apg_ccm_enabled` ke 1 di grup parameter klaster DB kustom klaster. Untuk mempelajari caranya, lihat Mengonfigurasi manajemen cache klaster.

Manajemen cache klaster aktif pada klaster DB Aurora PostgreSQL ketika klaster memiliki instans Pembaca Aurora yang dikonfigurasi sebagai berikut:

- Instans Pembaca Aurora menggunakan jenis dan ukuran kelas instans DB yang sama dengan instans Penulis klaster.
- Instans Pembaca Aurora dikonfigurasi sebagai Tier-0 untuk klaster. Jika klaster memiliki lebih dari satu Pembaca, ini adalah satu-satunya Pembaca Tier-0.

Mengatur lebih dari satu Pembaca ke Tier-0 menonaktifkan CCM. Ketika CCM dinonaktifkan, memanggil fungsi ini mengembalikan pesan kesalahan berikut:

```
ERROR: Cluster Cache Manager is disabled
```

Anda juga dapat menggunakan ekstensi pg_buffercache PostgreSQL untuk menganalisis cache buffer. Untuk informasi selengkapnya, lihat pg_buffercache di dokumentasi PostgreSQL.

Untuk informasi selengkapnya, lihat Pengantar manajemen cache klaster Aurora PostgreSQL.

Contoh

Contoh berikut menampilkan hasil pemanggilan fungsi `aurora_ccm_status`. Contoh pertama ini menunjukkan statistik CCM.

```
=> SELECT * FROM aurora_ccm_status();
 buffers_sent_last_minute | buffers_found_last_minute | buffers_sent_last_scan |
 buffers_found_last_scan | buffers_sent_current_scan | buffers_found_current_scan |
 current_scan_progress
--------------------------+---------------------------+------------------------
+-------------------------+---------------------------+----------------------------
+------------------------
                   2242000 |                   2242003 |               17920442 |
         17923410 |                  14098000 |                  14100964 |
   15877443
```

Untuk detail yang lebih lengkap, Anda dapat menggunakan tampilan yang diperluas, seperti yang ditunjukkan berikut:

```
\x
Expanded display is on.
SELECT *  FROM aurora_ccm_status();
[ RECORD 1 ]----------------------+---------
buffers_sent_last_minute          | 2242000
buffers_found_last_minute         | 2242003
buffers_sent_last_scan            | 17920442
buffers_found_last_scan           | 17923410
buffers_sent_current_scan         | 14098000
buffers_found_current_scan        | 14100964
current_scan_progress             | 15877443
```

Contoh ini menunjukkan cara memeriksa tingkat hangat dan persentase hangat.

```
=> SELECT buffers_sent_last_minute * 8/60 AS warm_rate_kbps,
100 * (1.0-buffers_sent_last_scan/buffers_found_last_scan) AS warm_percent
FROM aurora_ccm_status ();
 warm_rate_kbps | warm_percent
```

```
----------------+--------------
 16523 |        100.0
```

aurora_global_db_instance_status

Menampilkan status semua instans Aurora, termasuk replika dalam klaster DB global Aurora.

Sintaksis

```
aurora_global_db_instance_status()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `server_id` – Pengidentifikasi instans DB.
- `session_id` – Pengidentifikasi unik untuk sesi saat ini. Nilai `MASTER_SESSION_ID` mengidentifikasi instans DB Penulis (primer).
- `aws_region` – Wilayah AWS tempat instans DB global ini berjalan. Untuk daftar Wilayah, lihat Ketersediaan wilayah.
- `durable_lsn` – Nomor urutan log (LSN) dibuat tahan lama di penyimpanan. Nomor urutan log (LSN) adalah nomor urut unik yang mengidentifikasi catatan di log transaksi basis data. LSN dipesan sedemikian rupa sehingga LSN yang lebih besar mewakili transaksi berikutnya.
- `highest_lsn_rcvd` – LSN tertinggi yang diterima oleh instans DB dari instans DB penulis.
- `feedback_epoch` – Masa yang digunakan instans DB saat menghasilkan informasi hot standby. Hot standby adalah instans DB yang mendukung koneksi dan kueri saat DB primer berada dalam mode pemulihan atau mode siaga. Informasi hot standby mencakup masa (titik waktu) dan detail lain tentang instans DB yang digunakan sebagai hot stanby. Untuk informasi selengkapnya, lihat Hot standby di dokumentasi PostgreSQL.
- `feedback_xmin` – ID transaksi aktif minimum (terlawas) yang digunakan oleh instans DB.
- `oldest_read_view_lsn` – LSN terlawas yang digunakan oleh instans DB untuk membaca dari penyimpanan.

- `visibility_lag_in_msec` – Seberapa jauh instans DB ini tetap tertinggal di belakang instans DB penulis dalam milidetik.

Catatan penggunaan

Fungsi ini menunjukkan statistik replikasi untuk klaster DB Aurora. Untuk setiap instans DB Aurora PostgreSQL di klaster, fungsi menunjukkan barisan data yang menyertakan replika lintas Wilayah dalam konfigurasi basis data global.

Anda dapat menjalankan fungsi ini dari instans apa pun di klaster DB Aurora PostgreSQL atau basis data global Aurora PostgreSQL. Fungsi mengembalikan detail tentang lag untuk semua instans replika.

Untuk mempelajari selengkapnya tentang pemantauan lag menggunakan fungsi ini (`aurora_global_db_instance_status`) atau dengan menggunakan `aurora_global_db_status`, lihat Memantau basis data global berbasis Aurora PostgreSQL.

Untuk informasi selengkapnya tentang basis data global Aurora, lihat Gambaran umum basis data global Amazon Aurora.

Untuk memulai dengan basis data global Aurora, lihat Memulai basis data global Amazon Aurora atau lihat FAQ Amazon Aurora.

Contoh

Contoh ini menunjukkan statistik instans lintas Wilayah.

```
=> SELECT *
  FROM aurora_global_db_instance_status();
               server_id                |              session_id              |
  aws_region  | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
----------------------------------------+--------------------------------------
+-------------+-------------+------------------+----------------+---------------
+----------------------+------------------------
 db-119-001-instance-01                  | MASTER_SESSION_ID                    | eu-
west-1    |  2534560273 |           [NULL] |         [NULL] |        [NULL] |
     [NULL] |                [NULL]
 db-119-001-instance-02                  | 4ecff34d-d57c-409c-ba28-278b31d6fc40 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                     6
```

```
 db-119-001-instance-03                  | 3e8a20fc-be86-43d5-95e5-bdf19d27ad6b | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560266 |                      6
 db-119-001-instance-04                  | fc1b0023-e8b4-4361-bede-2a7e926cead6 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                     23
 db-119-001-instance-05                  | 30319b74-3f08-4e13-9728-e02aa1aa8649 | eu-
west-1    |  2534560266 |       2534560273 |              0 |      19669196 |
 2534560254 |                     23
 db-119-001-global-instance-1            | a331ffbb-d982-49ba-8973-527c96329c60 | eu-
central-1 |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                    996
 db-119-001-global-instance-1            | e0955367-7082-43c4-b4db-70674064a9da | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                     14
 db-119-001-global-instance-1-eu-west-2a  | 1248dc12-d3a4-46f5-a9e2-85850491a897 | eu-
west-2    |  2534560254 |       2534560266 |              0 |      19669196 |
 2534560247 |                      0
```

Contoh ini menunjukkan cara memeriksa lag replika global dalam milidetik.

```
=> SELECT CASE
          WHEN 'MASTER_SESSION_ID' = session_id THEN 'Primary'
         ELSE 'Secondary'
      END AS global_role,
      aws_region,
      server_id,
      visibility_lag_in_msec
  FROM aurora_global_db_instance_status()
  ORDER BY 1, 2, 3;
   global_role |  aws_region  |               server_id                |
 visibility_lag_in_msec
-------------+--------------+----------------------------------------
+------------------------
 Primary     | eu-west-1    | db-119-001-instance-01                 |
 [NULL]
 Secondary   | eu-central-1 | db-119-001-global-instance-1           |
     13
 Secondary   | eu-west-1    | db-119-001-instance-02                 |
     10
 Secondary   | eu-west-1    | db-119-001-instance-03                 |
      9
```

```
 Secondary   | eu-west-1    | db-119-001-instance-04                  |
      2
 Secondary   | eu-west-1    | db-119-001-instance-05                  |
     18
 Secondary   | eu-west-2    | db-119-001-global-instance-1            |
     14
 Secondary   | eu-west-2    | db-119-001-global-instance-1-eu-west-2a |
     13
```

Contoh ini menunjukkan cara memeriksa min, maks, dan rata-rata lag per Wilayah AWS dari konfigurasi basis data global.

```
=> SELECT 'Secondary' global_role,
       aws_region,
       min(visibility_lag_in_msec) min_lag_in_msec,
       max(visibility_lag_in_msec) max_lag_in_msec,
       round(avg(visibility_lag_in_msec),0) avg_lag_in_msec
  FROM aurora_global_db_instance_status()
 WHERE aws_region NOT IN (SELECT   aws_region
                            FROM aurora_global_db_instance_status()
                           WHERE session_id='MASTER_SESSION_ID')
                        GROUP BY aws_region
UNION ALL
SELECT  'Primary' global_role,
        aws_region,
        NULL,
        NULL,
        NULL
    FROM aurora_global_db_instance_status()
   WHERE session_id='MASTER_SESSION_ID'
ORDER BY 1, 5;
 global_role |  aws_region  | min_lag_in_msec | max_lag_in_msec | avg_lag_in_msec
-------------+--------------+-----------------+-----------------+-----------------
 Primary     | eu-west-1    |          [NULL] |          [NULL] |          [NULL]
 Secondary   | eu-central-1 |             133 |             133 |             133
 Secondary   | eu-west-2    |               0 |             495 |             248
```

aurora_global_db_status

Menampilkan informasi tentang berbagai aspek lag basis data global Aurora, khususnya, lag penyimpanan Aurora yang mendasarinya (disebut lag daya tahan) dan lag antara sasaran titik pemulihan (RPO).

Sintaksis

```
aurora_global_db_status()
```

Argumen

Tidak ada.

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `aws_region` – Wilayah AWS tempat klaster DB ini berada. Untuk daftar lengkap Wilayah AWS berdasarkan mesin, lihat Wilayah dan Zona Ketersediaan.
- `highest_lsn_written` – nomor urutan log (LSN) tertinggi yang saat ini ada di klaster DB ini. Nomor urutan log (LSN) adalah nomor urut unik yang mengidentifikasi catatan di log transaksi basis data. LSN dipesan sedemikian rupa sehingga LSN yang lebih besar mewakili transaksi berikutnya.
- `durability_lag_in_msec` – Perbedaan nilai stempel waktu antara `highest_lsn_written` di klaster DB sekunder dan `highest_lsn_written` klaster DB primer. Nilai -1 mengidentifikasi klaster DB primer dari basis data global Aurora.
- `rpo_lag_in_msec` – Lag sasaran titik pemulihan (RPO). Lag RPO adalah waktu yang dibutuhkan COMMIT transaksi pengguna terbaru untuk disimpan di klaster DB sekunder setelah disimpan di klaster DB primer dari basis data global Aurora. Nilai -1 menunjukkan klaster DB primer (dan dengan demikian, lag tidak relevan).

  Secara sederhana, metrik ini menghitung tujuan titik pemulihan untuk setiap klaster DB Aurora PostgreSQL di basis data global Aurora, yaitu, berapa banyak data yang mungkin hilang jika ada pemadaman. Seperti halnya lag, RPO diukur dalam waktu.
- `last_lag_calculation_time` – Stempel waktu yang menentukan kapan nilai terakhir dihitung untuk `durability_lag_in_msec` dan `rpo_lag_in_msec`. Nilai waktu seperti `1970-01-01 00:00:00+00` berarti ini adalah klaster DB primer.
- `feedback_epoch` – Masa yang digunakan klaster DB sekunder saat menghasilkan informasi hot standby. Hot standby adalah instans DB yang mendukung koneksi dan kueri saat DB primer berada dalam mode pemulihan atau mode siaga. Informasi hot standby mencakup masa (titik waktu) dan detail lain tentang instans DB yang digunakan sebagai hot stanby. Untuk informasi selengkapnya, lihat Hot standby di dokumentasi PostgreSQL.

- `feedback_xmin` – ID transaksi aktif minimum (terlawas) yang digunakan oleh klaster DB sekunder.

Catatan penggunaan

Fungsi ini menunjukkan statistik replikasi untuk basis data global Aurora. Hal ini menampilkan satu baris untuk setiap klaster DB dalam basis data global Aurora PostgreSQL. Anda dapat menjalankan fungsi ini dari instans apa pun di basis data global Aurora PostgreSQL.

Untuk mengevaluasi lag replikasi basis data global Aurora, yang merupakan lag data yang terlihat, lihat aurora_global_db_instance_status.

Untuk mempelajari selengkapnya tentang menggunakan `aurora_global_db_status` dan `aurora_global_db_instance_status` memantau lag basis data global Aurora, lihat Memantau basis data global berbasis Aurora PostgreSQL. Untuk informasi selengkapnya tentang basis data global Aurora, lihat Gambaran umum basis data global Amazon Aurora.

Contoh

Contoh ini menunjukkan cara menampilkan statistik penyimpanan lintas wilayah.

```
=> SELECT CASE
          WHEN '-1' = durability_lag_in_msec THEN 'Primary'
          ELSE 'Secondary'
       END AS global_role,
       *
  FROM aurora_global_db_status();
 global_role | aws_region | highest_lsn_written | durability_lag_in_msec |
 rpo_lag_in_msec | last_lag_calculation_time  | feedback_epoch | feedback_xmin
-------------+------------+---------------------+------------------------
+-----------------+----------------------------+----------------+--------------
 Primary     | eu-west-1  |           131031557 |                     -1 |
 -1 | 1970-01-01 00:00:00+00     |              0 |              0
 Secondary   | eu-west-2  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |              0 |         12640
 Secondary   | eu-west-3  |           131031554 |                    410 |
  0 | 2021-06-01 18:59:36.124+00 |              0 |         12640
```

aurora_list_builtins

Daftar semua fungsi bawaan Aurora PostgreSQL yang tersedia, bersama dengan deskripsi singkat dan detail fungsi.

Sintaksis

```
aurora_list_builtins()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF

Contoh

Contoh berikut menampilkan hasil dari pemanggilan fungsi `aurora_list_builtins`.

```
=> SELECT *
FROM aurora_list_builtins();

               Name                | Result data type |                   Argument data
 types                     | Type | Volatility |  Parallel  | Security |
             Description
-----------------------------------+------------------
+-----------------------------------------------------------+------+------------
+------------+----------
+-------------------------------------------------------------------------
 aurora_version                    | text             |
                           | func | stable     | safe       | invoker  | Amazon Aurora
 PostgreSQL-Compatible Edition version string
 aurora_stat_wait_type             | SETOF record     | OUT type_id smallint, OUT
 type_name text                 | func | volatile   | restricted | invoker  | Lists all
 supported wait types
 aurora_stat_wait_event            | SETOF record     | OUT type_id smallint, OUT
 event_id integer, OUT event_na.| func | volatile   | restricted | invoker  | Lists all
 supported wait events
                                   |                  |.me text
                           |      |            |            |          |
 aurora_list_builtins              | SETOF record     | OUT "Name" text, OUT "Result
 data type" text, OUT "Argum.| func | stable     | safe       | invoker  | Lists all
 Aurora built-in functions
                                   |                  |.ent data types" text, OUT
 "Type" text, OUT "Volatility" .|      |            |            |          |
```

```
                                    |                  |.text, OUT "Parallel" text, OUT
"Security" text, OUT "Des.|      |            |            |          |
                                    |                  |.cription" text
                          |      |            |            |          |
.
.
.
aurora_stat_file                    | SETOF record     | OUT filename text, OUT
allocated_bytes bigint, OUT used_.| func | stable     | safe       | invoker  | Lists
all files present in Aurora storage
                                    |                  |.bytes bigint
                          |      |            |            |          |
aurora_stat_get_db_commit_latency | bigint           | oid
                           | func | stable     | restricted | invoker  | Per DB commit
latency in microsecs
```

aurora_replica_status

Menampilkan status semua simpul pembaca Aurora PostgreSQL.

Sintaksis

```
aurora_replica_status()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `server_id` – ID instans DB (pengidentifikasi).
- `session_id` – Pengidentifikasi unik untuk sesi saat ini, dikembalikan untuk instans utama dan instans pembaca sebagai berikut:
  - Untuk contoh instans primer, `session_id` selalu `` `MASTER_SESSION_ID' ``.
  - Untuk instans pembaca, `session_id` selalu merupakan `UUID` (pengidentifikasi unik universal) dari instans pembaca.
- `durable_lsn` – Nomor urutan log (LSN) yang disimpan di penyimpanan.
  - Untuk volume primer, volume primer LSN (VDL) tahan lama yang saat ini berlaku.

- Untuk setiap volume sekunder, VDL primer yang sekundernya telah berhasil diterapkan.

> **Note**
>
> Nomor urutan log (LSN) adalah nomor urut unik yang mengidentifikasi catatan di log transaksi basis data. LSN dipesan sedemikian rupa sehingga LSN yang lebih besar mewakili transaksi yang terjadi berikutnya dalam urutan.

- `highest_lsn_rcvd` – LSN tertinggi (terbaru) yang diterima oleh instans DB dari instans DB penulis.
- `current_read_lsn` – LSN dari snapshot terbaru yang telah diterapkan untuk semua pembaca.
- `cur_replay_latency_in_usec` – Jumlah mikrodetik yang diharapkan untuk memutar ulang log di sekunder.
- `active_txns` – Jumlah transaksi yang aktif saat ini.
- `is_current` – Tidak digunakan.
- `last_transport_error` – Kode kesalahan replikasi terakhir.
- `last_error_timestamp` – Stempel waktu kesalahan replikasi terakhir.
- `last_update_timestamp` – Stempel waktu pembaruan terakhir ke status replika. Dari Aurora PostgreSQL 13.9, nilai `last_update_timestamp` untuk instans DB yang terhubung dengan Anda diatur ke `NULL`.
- `feedback_xmin` – Hot standby feedback_xmin dari replika. ID transaksi aktif minimum (terlawas) yang digunakan oleh instans DB.
- `feedback_epoch` – Masa yang digunakan instans DB saat menghasilkan informasi hot standby.
- `replica_lag_in_msec` – Waktu instans pembaca mengalami lag dari instans penulis, dalam milidetik.
- `log_stream_speed_in_kib_per_second` – Throughput log stream dalam kilobyte per detik.
- `log_buffer_sequence_number` – Nomor urutan buffer log.
- `oldest_read_view_trx_id` – Tidak digunakan.
- `oldest_read_view_lsn` – LSN terlawas yang digunakan oleh instans DB untuk membaca dari penyimpanan.
- `pending_read_ios` – Pembacaan halaman luar biasa yang masih tertunda pada replika.
- `read_ios` – Jumlah pembacaan halaman pada replika.
- `iops` – Tidak digunakan.

- `cpu` – Penggunaan CPU dengan proses replika. Perhatikan bahwa ini bukan penggunaan CPU menurut instans melainkan prosesnya. Untuk informasi tentang penggunaan CPU oleh instans, lihatMetrik tingkat instans untuk Amazon Aurora.

Catatan penggunaan

Fungsi `aurora_replica_status` mengembalikan nilai dari manajer status replika klaster DB Aurora PostgreSQL. Anda dapat menggunakan fungsi ini untuk mendapatkan informasi tentang status replikasi pada klaster DB PostgreSQL Aurora Anda, termasuk metrik untuk semua instans DB di klaster DB Aurora Anda. Misalnya, Anda dapat melakukan hal berikut:

- Mendapatkan informasi tentang jenis instans (penulis, pembaca) di klaster DB Aurora PostgreSQL – Anda dapat memperoleh informasi ini dengan memeriksa nilai-nilai kolom berikut:
  - `server_id` – Berisi nama instans yang Anda tentukan saat membuat instans. Dalam beberapa kasus, seperti untuk instans primer (penulis), nama biasanya dibuat untuk Anda dengan menambahkan -instance-1 ke nama yang Anda buat untuk klaster DB Aurora PostgreSQL Anda.
  - `session_id` – Bidang `session_id` menunjukkan apakah instans adalah pembaca atau penulis. Untuk instans penulis, `session_id` selalu diatur ke `"MASTER_SESSION_ID"`. Untuk instans pembaca, `session_id` diatur ke `UUID` pembaca tertentu.
- Mendiagnosis masalah replikasi umum, seperti lag replika – Lag replica adalah waktu dalam milidetik cache halaman dari instans pembaca berada di belakang instans penulis. Lag ini terjadi karena klaster Aurora menggunakan replikasi asinkron, seperti yang dijelaskan dalam Replikasi dengan Amazon Aurora. Lagi ini ditampilkan di kolom `replica_lag_in_msec` dalam hasil yang dikembalikan oleh fungsi ini. Lag juga dapat terjadi ketika kueri dibatalkan karena konflik dengan pemulihan pada server siaga. Anda dapat memeriksa `pg_stat_database_conflicts()` untuk memverifikasi bahwa konflik seperti itu menyebabkan lag replika (atau tidak). Untuk informasi selengkapnya, lihat Pengumpul Statistik di Dokumentasi PostgreSQL. Untuk mempelajari selengkapnya tentang ketersediaan dan replikasi tinggi, lihat FAQ Amazon Aurora.

  Amazon CloudWatch menyimpan `replica_lag_in_msec` hasil dari waktu ke waktu, sebagai metrik `AuroraReplicaLag`. Untuk informasi tentang melihat metrik CloudWatch untuk Aurora, lihat Memantau metrik Amazon Aurora dengan Amazon CloudWatch

Untuk mempelajari selengkapnya tentang pemecahan masalah replika dan pemulaian ulang pembacaan Aurora, lihat Mengapa replika pembacaan Amazon Aurora saya tertinggal dan memulai ulang? di Pusat AWS Support.

Contoh

Contoh berikut menunjukkan cara mendapatkan status replikasi dari semua instans di klaster DB Aurora PostgreSQL:

```
=> SELECT *
FROM aurora_replica_status();
```

Contoh berikut menunjukkan contoh instans penulis di klaster DB Aurora PostgreSQL `docs-lab-apg-main`:

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role
FROM aurora_replica_status()
WHERE session_id = 'MASTER_SESSION_ID';
        server_id       | instance_role
------------------------+---------------
 db-119-001-instance-01 | writer
```

Contoh berikut mencantumkan semua instans pembaca dalam sebuah klaster:

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role
FROM aurora_replica_status()
WHERE session_id <> 'MASTER_SESSION_ID';
        server_id       | instance_role
------------------------+---------------
db-119-001-instance-02  | reader
db-119-001-instance-03  | reader
db-119-001-instance-04  | reader
db-119-001-instance-05  | reader
(4 rows)
```

Contoh berikut mencantumkan semua instans, seberapa jauh setiap instans tertinggal dari penulis, dan berapa lama sejak pembaruan terakhir:

```
=> SELECT server_id,
    CASE
        WHEN 'MASTER_SESSION_ID' = session_id THEN 'writer'
        ELSE 'reader'
    END AS instance_role,
    replica_lag_in_msec AS replica_lag_ms,
    round(extract (epoch FROM (SELECT age(clock_timestamp(), last_update_timestamp))) *
 1000) AS last_update_age_ms
FROM aurora_replica_status()
ORDER BY replica_lag_in_msec NULLS FIRST;
       server_id        | instance_role | replica_lag_ms | last_update_age_ms
------------------------+---------------+----------------+--------------------
 db-124-001-instance-03 | writer        |         [NULL] |               1756
 db-124-001-instance-01 | reader        |             13 |               1756
 db-124-001-instance-02 | reader        |             13 |               1756
(3 rows)
```

aurora_stat_activity

Mengembalikan satu baris per proses server, menampilkan informasi yang terkait dengan aktivitas saat ini dari proses itu.

Sintaks

```
aurora_stat_activity();
```

Argumen

Tidak ada

Jenis pengembalian

Mengembalikan satu baris per proses server. Selain `pg_stat_activity` kolom, bidang berikut ditambahkan:

- planid - pengidentifikasi rencana

Catatan penggunaan

Tampilan tambahan untuk `pg_stat_activity` mengembalikan kolom yang sama dengan `plan_id` kolom tambahan yang menunjukkan rencana eksekusi kueri saat ini.

`aurora_compute_plan_id`harus diaktifkan agar tampilan mengembalikan plan_id.

Fungsi ini tersedia dari Aurora PostgreSQL versi 14.10, 15.5, dan untuk semua versi lain yang lebih baru.

Contoh-contoh

Contoh kueri di bawah ini menggabungkan beban teratas dengan query_id dan plan_id.

```
db1=# select count(*), query_id, plan_id
db1-# from aurora_stat_activity() where state = 'active'
db1-# and pid <> pg_backend_pid()
db1-# group by query_id, plan_id
db1-# order by 1 desc;

count |  query_id             |  plan_id
-------+----------------------+-------------
 11    | -5471422286312252535 | -2054628807
 3     | -6907107586630739258 | -815866029
 1     | 5213711845501580017  |  300482084
(3 rows)
```

Jika rencana yang digunakan untuk query_id berubah, plan_id baru akan dilaporkan oleh aurora_stat_activity.

```
count  |  query_id             |  plan_id
-------+----------------------+-------------
 10    | -5471422286312252535 | 1602979607
 1     | -6907107586630739258 | -1809935983
 1     | -2446282393000597155 | -207532066
(3 rows)
```

aurora_stat_backend_waits

Menampilkan statistik untuk aktivitas menunggu untuk proses backend tertentu.

Sintaksis

```
aurora_stat_backend_waits(pid)
```

Argumen

`pid` – ID untuk proses backend. Anda dapat memperoleh ID proses dengan menggunakan tampilan `pg_stat_activity`.

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `type_id` – Angka yang menunjukkan jenis peristiwa tunggu, seperti 1 untuk kunci ringan (`LWLock`), 3 untuk kunci, atau 6 untuk sesi klien, untuk menyebutkan beberapa contoh. Nilai-nilai ini menjadi bermakna ketika Anda menggabungkan hasil fungsi ini dengan kolom dari fungsi `aurora_stat_wait_type`, seperti yang ditunjukkan dalam Contoh.
- `event_id` – Nomor pengidentifikasi untuk peristiwa tunggu. Gabungkan nilai ini dengan kolom dari `aurora_stat_wait_event` untuk mendapatkan nama peristiwa yang bermakna.
- `waits` – Hitungan jumlah tunggu yang terakumulasi untuk ID proses yang ditentukan.
- `wait_time` – Waktu tunggu dalam milidetik.

Catatan penggunaan

Anda dapat menggunakan fungsi ini untuk menganalisis peristiwa tunggu backend (sesi) tertentu yang terjadi sejak koneksi dibuka. Untuk mendapatkan informasi yang lebih bermakna tentang nama dan jenis peristiwa tunggu, Anda dapat menggabungkan fungsi `aurora_stat_wait_type` dan `aurora_stat_wait_event` ini, dengan menggunakan JOIN seperti yang ditunjukkan dalam contoh.

Contoh

Contoh ini menunjukkan semua tunggu, jenis, dan nama peristiwa untuk proses backend ID 3027.

```
=> SELECT type_name, event_name, waits, wait_time
        FROM aurora_stat_backend_waits(3027)
NATURAL JOIN aurora_stat_wait_type()
NATURAL JOIN aurora_stat_wait_event();
```

```
type_name |       event_name       | waits | wait_time
-----------+------------------------+-------+------------
 LWLock    | ProcArrayLock          |     3 |         27
 LWLock    | wal_insert             |   423 |      16336
 LWLock    | buffer_content         | 11840 |    1033634
 LWLock    | lock_manager           | 23821 |    5664506
 Lock      | tuple                  | 10258 |  152280165
 Lock      | transactionid          | 78340 | 1239808783
 Client    | ClientRead             | 34072 |   17616684
 IO        | ControlFileSyncUpdate  |     2 |          0
 IO        | ControlFileWriteUpdate |     4 |         32
 IO        | RelationMapRead        |     2 |        795
 IO        | WALWrite               | 36666 |      98623
 IO        | XactSync               |  4867 |    7331963
```

Contoh ini menunjukkan jenis tunggu saat ini dan kumulatif serta peristiwa tunggu untuk semua sesi aktif (`pg_stat_activity state <> 'idle'`) (tetapi tanpa sesi saat ini yang menjalankan fungsi (`pid <> pg_backend_pid()`).

```
=> SELECT a.pid,
            a.usename,
            a.app_name,
            a.current_wait_type,
            a.current_wait_event,
            a.current_state,
            wt.type_name AS wait_type,
            we.event_name AS wait_event,
            a.waits,
            a.wait_time
       FROM (SELECT pid,
                    usename,
                    left(application_name,16) AS app_name,
                    coalesce(wait_event_type,'CPU') AS current_wait_type,
                    coalesce(wait_event,'CPU') AS current_wait_event,
                    state AS current_state,
                    (aurora_stat_backend_waits(pid)).*
               FROM pg_stat_activity
              WHERE pid <> pg_backend_pid()
                AND state <> 'idle') a
NATURAL JOIN aurora_stat_wait_type() wt
NATURAL JOIN aurora_stat_wait_event() we;
  pid  | usename  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |       wait_event       | waits | wait_time
```

```
-------+----------+----------+------------------+--------------------+--------------
+-----------+------------------------+-------+-----------
 30099 | postgres | pgbench  | Lock             | transactionid      | active        |
 LWLock    | wal_insert             |  1937 |     29975
 30099 | postgres | pgbench  | Lock             | transactionid      | active        |
 LWLock    | buffer_content         | 22903 |    760498
 30099 | postgres | pgbench  | Lock             | transactionid      | active        |
 LWLock    | lock_manager           | 10012 |    223207
 30099 | postgres | pgbench  | Lock             | transactionid      | active        |
 Lock      | tuple                  | 20315 |  63081529
 .
 .
 .
 30099 | postgres | pgbench  | Lock             | transactionid      | active        |
 IO        | WALWrite               | 93293 |    237440
 30099 | postgres | pgbench  | Lock             | transactionid      | active        |
 IO        | XactSync               | 13010 |  19525143
 30100 | postgres | pgbench  | Lock             | transactionid      | active        |
 LWLock    | ProcArrayLock          |     6 |        53
 30100 | postgres | pgbench  | Lock             | transactionid      | active        |
 LWLock    | wal_insert             |  1913 |     25450
 30100 | postgres | pgbench  | Lock             | transactionid      | active        |
 LWLock    | buffer_content         | 22874 |    778005
 .
 .
 .
 30109 | postgres | pgbench  | IO               | XactSync           | active        |
 LWLock    | ProcArrayLock          |     3 |        71
 30109 | postgres | pgbench  | IO               | XactSync           | active        |
 LWLock    | wal_insert             |  1940 |     27741
 30109 | postgres | pgbench  | IO               | XactSync           | active        |
 LWLock    | buffer_content         | 22962 |    776352
 30109 | postgres | pgbench  | IO               | XactSync           | active        |
 LWLock    | lock_manager           |  9879 |    218826
 30109 | postgres | pgbench  | IO               | XactSync           | active        |
 Lock      | tuple                  | 20401 |  63581306
 30109 | postgres | pgbench  | IO               | XactSync           | active        |
 Lock      | transactionid          | 50769 | 211645008
 30109 | postgres | pgbench  | IO               | XactSync           | active        |
 Client    | ClientRead             | 89901 |  44192439
```

Contoh ini menunjukkan jenis tunggu kumulatif saat ini dan tiga teratas (3) serta peristiwa tunggu kumulatif untuk semua sesi aktif (`pg_stat_activity state <> 'idle'`) tidak termasuk sesi saat ini (`pid <>pg_backend_pid()`).

```
=> SELECT top3.*
       FROM (SELECT a.pid,
                    a.usename,
                    a.app_name,
                    a.current_wait_type,
                    a.current_wait_event,
                    a.current_state,
                    wt.type_name AS wait_type,
                    we.event_name AS wait_event,
                    a.waits,
                    a.wait_time,
                    RANK() OVER (PARTITION BY pid ORDER BY a.wait_time DESC)
               FROM (SELECT pid,
                            usename,
                            left(application_name,16) AS app_name,
                            coalesce(wait_event_type,'CPU') AS current_wait_type,
                            coalesce(wait_event,'CPU') AS current_wait_event,
                            state AS current_state,
                            (aurora_stat_backend_waits(pid)).*
                       FROM pg_stat_activity
                      WHERE pid <> pg_backend_pid()
                        AND state <> 'idle') a
       NATURAL JOIN aurora_stat_wait_type() wt
       NATURAL JOIN aurora_stat_wait_event() we) top3
 WHERE RANK <=3;
  pid  | usename  | app_name | current_wait_type | current_wait_event | current_state |
 wait_type |   wait_event    |  waits  | wait_time  | rank
-------+----------+----------+-------------------+--------------------+--------------
+-----------+-----------------+---------+------------+------
 20567 | postgres | psql     | CPU               | CPU                | active        |
 LWLock    | wal_insert      |   25000 |   67512003 |    1
 20567 | postgres | psql     | CPU               | CPU                | active        |
 IO        | WALWrite        | 3071758 |    1016961 |    2
 20567 | postgres | psql     | CPU               | CPU                | active        |
 IO        | BufFileWrite    |   20750 |     184559 |    3
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | transactionid   |  237350 | 1265580011 |    1
 27743 | postgres | pgbench  | Lock              | transactionid      | active        |
 Lock      | tuple           |   93641 |  341472318 |    2
```

```
27743 | postgres | pgbench  | Lock              | transactionid      | active        |
Client    | ClientRead      |  417556 |  204796837 |    3
.
.
.
27745 | postgres | pgbench  | IO                | XactSync           | active        |
Lock      | transactionid   |  238068 | 1265816822 |    1
27745 | postgres | pgbench  | IO                | XactSync           | active        |
Lock      | tuple           |   93210 |  338312247 |    2
27745 | postgres | pgbench  | IO                | XactSync           | active        |
Client    | ClientRead      |  419157 |  207836533 |    3
27746 | postgres | pgbench  | Lock              | transactionid      | active        |
Lock      | transactionid   |  237621 | 1264528811 |    1
27746 | postgres | pgbench  | Lock              | transactionid      | active        |
Lock      | tuple           |   93563 |  339799310 |    2
27746 | postgres | pgbench  | Lock              | transactionid      | active        |
Client    | ClientRead      |  417304 |  208372727 |    3
```

aurora_stat_bgwriter

`aurora_stat_bgwriter` adalah tampilan statistik yang menampilkan informasi tentang penulisan cache Optimized Reads.

Sintaksis

```
aurora_stat_bgwriter()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan semua kolom `pg_stat_bgwriter` dan kolom tambahan berikut. Untuk informasi selengkapnya tentang kolom `pg_stat_bgwriter`, lihat `pg_stat_bgwriter`.

Anda dapat mengatur ulang statistik untuk fungsi ini menggunakan `pg_stat_reset_shared("bgwriter")`.

- `orcache_blks_written` – Jumlah total blok data cache pembacaan yang dioptimalkan ditulis.

- `orcache_blk_write_time` – Jika `track_io_timing` diaktifkan, fungsi ini melacak total waktu yang dihabiskan untuk menulis blok file data cache yang dioptimalkan, dalam milidetik. Untuk informasi selengkapnya, lihat track_io_timing.

Catatan penggunaan

Fungsi ini tersedia di versi Aurora PostgreSQL berikut:

- Versi 15.4 dan versi 15 yang lebih tinggi
- Versi 14.9 dan versi 14 yang lebih tinggi

Contoh

```
=> select * from aurora_stat_bgwriter();
-[ RECORD 1 ]-----------------+-----------
orcache_blks_written          | 246522
orcache_blk_write_time        | 339276.404
```

aurora_stat_database

Fungsi ini membawa semua kolom pg_stat_database dan menambahkan kolom baru pada akhirnya.

Sintaksis

```
aurora_stat_database()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan semua kolom `pg_stat_database` dan kolom tambahan berikut. Untuk informasi selengkapnya tentang kolom `pg_stat_database`, lihat `pg_stat_database`.

- `storage_blks_read` – Jumlah total blok bersama yang dibaca dari penyimpanan aurora dalam basis data ini.

- `orcache_blks_hit` – Jumlah total hit cache pembacaan yang dioptimalkan dalam basis data ini.
- `local_blks_read` – Jumlah total blok lokal yang dibaca dalam basis data ini.
- `storage_blk_read_time` – Jika `track_io_timing` diaktifkan, total waktu yang dihabiskan untuk membaca blok file data dari penyimpanan aurora dilacak dalam milidetik. Jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.
- `local_blk_read_time` – Jika `track_io_timing` diaktifkan, total waktu yang dihabiskan untuk membaca blok file data lokal dilacak dalam milidetik. Jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.
- `orcache_blk_read_time` – Jika `track_io_timing` diaktifkan, total waktu yang dihabiskan untuk membaca blok file data dari cache pembacaan yang dioptimalkan dilacak dalam milidetik. Jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.

> **Note**
>
> Nilai `blks_read` adalah jumlah dari `storage_blks_read`, `orcache_blks_hit`, dan `local_blks_read`.
> Nilai `blk_read_time` adalah jumlah dari `storage_blk_read_time`, `orcache_blk_read_time`, dan `local_blk_read_time`.

Catatan penggunaan

Fungsi ini tersedia di versi Aurora PostgreSQL berikut:

- Versi 15.4 dan versi 15 yang lebih tinggi
- Versi 14.9 dan versi 14 yang lebih tinggi

Contoh

Contoh berikut menunjukkan bagaimana fungsi ini membawa semua kolom `pg_stat_database` dan menambahkan 6 kolom baru pada akhirnya:

```
=> select * from aurora_stat_database() where datid=14717;
-[ RECORD 1 ]------------+------------------------------
datid                    | 14717
datname                  | postgres
```

```
numbackends               | 1
xact_commit               | 223
xact_rollback             | 4
blks_read                 | 1059
blks_hit                  | 11456
tup_returned              | 27746
tup_fetched               | 5220
tup_inserted              | 165
tup_updated               | 42
tup_deleted               | 91
conflicts                 | 0
temp_files                | 0
temp_bytes                | 0
deadlocks                 | 0
checksum_failures         |
checksum_last_failure     |
blk_read_time             | 3358.689
blk_write_time            | 0
session_time              | 1076007.997
active_time               | 3684.371
idle_in_transaction_time | 0
sessions                  | 10
sessions_abandoned        | 0
sessions_fatal            | 0
sessions_killed           | 0
stats_reset               | 2023-01-12 20:15:17.370601+00
orcache_blks_hit          | 425
orcache_blk_read_time     | 89.934
storage_blks_read         | 623
storage_blk_read_time     | 3254.914
local_blks_read           | 0
local_blk_read_time       | 0
```

aurora_stat_dml_activity

Melaporkan aktivitas kumulatif untuk setiap jenis operasi bahasa manipulasi data (DHTML) pada basis data dalam klaster Aurora PostgreSQL.

Sintaksis

```
aurora_stat_dml_activity(database_oid)
```

Argumen

database_oid

ID Objek (OID) dari basis data di klaster Aurora PostgreSQL.

Jenis pengembalian

Catatan SETOF

Catatan penggunaan

Fungsi `aurora_stat_dml_activity` hanya tersedia dengan Aurora PostgreSQL rilis 3.1 yang kompatibel dengan mesin PostgreSQL 11.6 dan yang lebih baru.

Gunakan fungsi ini pada klaster Aurora PostgreSQL dengan sejumlah besar basis data untuk mengidentifikasi basis data mana yang memiliki aktivitas DHTML lebih banyak atau lebih lambat, atau keduanya.

Fungsi `aurora_stat_dml_activity` mengembalikan berapa kali operasi dijalankan dan latensi kumulatif dalam mikrodetik untuk operasi SELECT, INSERT, UPDATE, dan DELETE. Laporan ini hanya mencakup operasi DHTML yang berhasil.

Anda dapat mengatur ulang statistik ini dengan menggunakan fungsi akses statistik PostgreSQL `pg_stat_reset`. Anda dapat memeriksa kapan terakhir kali statistik ini diatur ulang dengan menggunakan fungsi `pg_stat_get_db_stat_reset_time`. Untuk informasi selengkapnya tentang fungsi akses statistik PostgreSQL, lihat Pengumpul Statistik dalam dokumentasi PostgreSQL.

Contoh

Contoh berikut menunjukkan cara melaporkan statistik aktivitas DML untuk basis data yang terhubung.

```
--Define the oid variable from connected database by using \gset
=> SELECT oid,
          datname
     FROM pg_database
    WHERE datname=(select current_database()) \gset
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
select_count | select_latency_microsecs | insert_count | insert_latency_microsecs |
 update_count | update_latency_microsecs | delete_count | delete_latency_microsecs
```

```
--------------+--------------------------+-------------+--------------------------
+--------------+--------------------------+-------------+--------------------------
       178957 |                 66684115 |       171065 |                 28876649 |
    519538 |            1454579206167 |            1 |                        53027


-- Showing the same results with expanded display on
=> SELECT *
     FROM aurora_stat_dml_activity(:oid);
-[ RECORD 1 ]------------+-------------
select_count             | 178957
select_latency_microsecs | 66684115
insert_count             | 171065
insert_latency_microsecs | 28876649
update_count             | 519538
update_latency_microsecs | 1454579206167
delete_count             | 1
delete_latency_microsecs | 53027
```

Contoh berikut menunjukkan statistik aktivitas DML untuk semua basis data di klaster Aurora PostgreSQL. Klaster ini memiliki dua basis data, `postgres` dan `mydb`. Daftar yang dipisahkan koma sesuai dengan bidang `select_count`, `select_latency_microsecs`, `insert_count`, `insert_latency_microsecs`, `update_count`, `update_latency_microsecs`, `delete_count`, dan `delete_latency_microsecs`.

Aurora PostgreSQL membuat dan menggunakan basis data sistem bernama `rdsadmin` untuk mendukung operasi administratif seperti pencadangan, pemulihan, pemeriksaan kondisi, replikasi, dan sebagainya. Operasi DML ini tidak berdampak pada klaster Aurora PostgreSQL Anda.

```
=> SELECT oid,
    datname,
    aurora_stat_dml_activity(oid)
    FROM pg_database;
oid  |    datname     |                   aurora_stat_dml_activity
-------+---------------
+-------------------------------------------------------------------
 14006 | template0      | (,,,,,,,)
 16384 | rdsadmin       | (2346623,1211703821,4297518,817184554,0,0,0,0)
     1 | template1      | (,,,,,,,)
 14007 | postgres       |
 (178961,66716329,171065,28876649,519538,1454579206167,1,53027)
 16401 | mydb           | (200246,64302436,200036,107101855,600000,83659417514,0,0)
```

Contoh berikut menunjukkan statistik aktivitas DML untuk semua basis data, diatur dalam kolom untuk keterbacaan yang lebih baik.

```
SELECT db.datname,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 1), '()') AS select_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 2), '()') AS select_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 3), '()') AS insert_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 4), '()') AS insert_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 5), '()') AS update_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 6), '()') AS update_latency_microsecs,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 7), '()') AS delete_count,
       BTRIM(SPLIT_PART(db.asdmla::TEXT, ',', 8), '()') AS delete_latency_microsecs
FROM  (SELECT datname,
              aurora_stat_dml_activity(oid) AS asdmla
         FROM pg_database
      ) AS db;

    datname     | select_count | select_latency_microsecs | insert_count |
 insert_latency_microsecs | update_count | update_latency_microsecs | delete_count |
 delete_latency_microsecs
---------------+--------------+--------------------------+--------------
+--------------------------+--------------+--------------------------+--------------
+--------------------------
 template0      |              |                          |              |
            |              |                          |              |
 rdsadmin       | 4206523      | 2478812333               | 7009414      | 1338482258
            | 0            | 0                        | 0            | 0
 template1      |              |                          |              |
            |              |                          |              |
 fault_test     | 66           | 452099                   | 0            | 0
            | 0            | 0                        | 0            | 0
 db_access_test | 1            | 5982                     | 0            | 0
            | 0            | 0                        | 0            | 0
 postgres       | 42035        | 95179203                 | 5752         | 2678832898
            | 21157        | 441883182488             | 2            | 1520
 mydb           | 71           | 453514                   | 0            | 0
            | 1            | 190                      | 1            | 152
```

Contoh berikut menunjukkan latensi kumulatif rata-rata (latensi kumulatif dibagi dengan hitungan) untuk setiap operasi DML untuk basis data dengan OID 16401.

```
=> SELECT select_count,
          select_latency_microsecs,
```

```
          select_latency_microsecs/NULLIF(select_count,0) select_latency_per_exec,
          insert_count,
          insert_latency_microsecs,
          insert_latency_microsecs/NULLIF(insert_count,0) insert_latency_per_exec,
          update_count,
          update_latency_microsecs,
          update_latency_microsecs/NULLIF(update_count,0) update_latency_per_exec,
          delete_count,
          delete_latency_microsecs,
          delete_latency_microsecs/NULLIF(delete_count,0) delete_latency_per_exec
     FROM aurora_stat_dml_activity(16401);
-[ RECORD 1 ]------------+-------------
select_count             | 451312
select_latency_microsecs | 80205857
select_latency_per_exec  | 177
insert_count             | 451001
insert_latency_microsecs | 123667646
insert_latency_per_exec  | 274
update_count             | 1353067
update_latency_microsecs | 200900695615
update_latency_per_exec  | 148478
delete_count             | 12
delete_latency_microsecs | 448
delete_latency_per_exec  | 37
```

aurora_stat_get_db_commit_latency

Mendapatkan latensi commit kumulatif dalam mikrodetik untuk basis data Aurora PostgreSQL. Latensi commit diukur sebagai waktu antara saat klien mengirimkan permintaan commit dan saat menerima pengakuan commit.

Sintaksis

```
aurora_stat_get_db_commit_latency(database_oid)
```

Argumen

database_oid

ID Objek (OID) dari basis data Aurora PostgreSQL.

Jenis pengembalian

Catatan SETOF

Catatan penggunaan

Amazon CloudWatch menggunakan fungsi ini untuk menghitung latensi commit rata-rata. Untuk informasi selengkapnya tentang metrik Amazon CloudWatch dan cara memecahkan masalah latensi commit tinggi, lihat Melihat metrik di konsol Amazon RDS dan Membuat keputusan yang lebih baik tentang Amazon RDS dengan metrik Amazon CloudWatch.

Anda dapat mengatur ulang statistik ini dengan menggunakan fungsi akses statistik PostgreSQL `pg_stat_reset`. Anda dapat memeriksa kapan terakhir kali statistik ini diatur ulang dengan menggunakan fungsi `pg_stat_get_db_stat_reset_time`. Untuk informasi selengkapnya tentang fungsi akses statistik PostgreSQL, lihat Pengumpul Statistik dalam dokumentasi PostgreSQL.

Contoh

Contoh berikut mendapatkan latensi commit kumulatif untuk setiap basis data di klaster `pg_database`.

```
=> SELECT oid,
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database;

  oid  |    datname     | aurora_stat_get_db_commit_latency
-------+----------------+-----------------------------------
 14006 | template0      |                                 0
 16384 | rdsadmin       |                         654387789
     1 | template1      |                                 0
 16401 | mydb           |                            229556
 69768 | postgres       |                             22011
```

Contoh berikut mendapatkan latensi commit kumulatif untuk basis data yang terhubung saat ini. Sebelum memanggil fungsi `aurora_stat_get_db_commit_latency`, contoh pertama menggunakan `\gset` guna mendefinisikan variabel untuk argumen `oid` dan menetapkan nilainya dari basis data yang terhubung.

```
--Get the oid value from the connected database before calling
 aurora_stat_get_db_commit_latency
```

```
=> SELECT oid
     FROM pg_database
    WHERE datname=(SELECT current_database()) \gset
=> SELECT *
     FROM aurora_stat_get_db_commit_latency(:oid);

 aurora_stat_get_db_commit_latency
-----------------------------------
                        1424279160
```

Contoh berikut mendapatkan latensi commit kumulatif untuk basis data `mydb` di klaster `pg_database`. Setelah itu, contoh tersebut mengatur ulang statistik ini dengan menggunakan fungsi `pg_stat_reset` dan menampilkan hasilnya. Terakhir, contoh tersebut menggunakan fungsi `pg_stat_get_db_stat_reset_time` untuk memeriksa kapan terakhir kali statistik diatur ulang.

```
=> SELECT oid,
    datname,
    aurora_stat_get_db_commit_latency(oid)
    FROM pg_database
    WHERE datname = 'mydb';

  oid  |  datname  | aurora_stat_get_db_commit_latency
-------+-----------+-----------------------------------
 16427 | mydb      |                           3320370


=> SELECT pg_stat_reset();
 pg_stat_reset
---------------


=> SELECT oid,
          datname,
          aurora_stat_get_db_commit_latency(oid)
     FROM pg_database
    WHERE datname = 'mydb';
  oid  |  datname  | aurora_stat_get_db_commit_latency
-------+-----------+-----------------------------------
 16427 | mydb      |                                 6


=> SELECT *
     FROM pg_stat_get_db_stat_reset_time(16427);
```

```
 pg_stat_get_db_stat_reset_time
-------------------------------
 2021-04-29 21:36:15.707399+00
```

aurora_stat_logical_wal_cache

Menampilkan penggunaan cache log write-ahead (WAL) per slot.

Sintaksis

```
SELECT * FROM aurora_stat_logical_wal_cache()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `name` – Nama slot replikasi.
- `active_pid` – ID proses walsender.
- `cache_hit` – Jumlah total hit cache wal sejak atur ulang terakhir.
- `cache_miss` – Jumlah total hilang cache wal sejak atur ulang terakhir.
- `blks_read` – Jumlah total permintaan baca cache wal.
- `hit_rate` – Laju hit cache WAL (cache_hit/blks_read).
- `last_reset_timestamp` – Terakhir kali penghitung diatur ulang.

Catatan penggunaan

Fungsi ini tersedia untuk versi berikut.

- Aurora PostgreSQL 14.7
- Aurora PostgreSQL versi 13.8 dan lebih tinggi
- Aurora PostgreSQL versi 12.12 dan lebih tinggi

- Aurora PostgreSQL versi 11.17 dan lebih tinggi

Contoh

Contoh berikut menampilkan dua slot replikasi dengan hanya satu yang aktif. Fungsi `aurora_stat_logical_wal_cache`.

```
=> SELECT *
     FROM aurora_stat_logical_wal_cache();
    name    | active_pid | cache_hit | cache_miss | blks_read | hit_rate |
 last_reset_timestamp
------------+------------+-----------+------------+-----------+----------
+-------------------------------
 test_slot1 |      79183 |        24 |          0 |        24 | 100.00%  | 2022-08-05
 17:39:56.830635+00
 test_slot2 |            |         1 |          0 |         1 | 100.00%  | 2022-08-05
 17:34:04.036795+00
(2 rows)
```

aurora_stat_memctx_usage

Melaporkan penggunaan konteks memori untuk setiap proses PostgreSQL.

Sintaksis

```
aurora_stat_memctx_usage()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `pid` – ID dari proses.
- `name` – Nama konteks memori.
- `allocated` – Jumlah byte yang diperoleh dari subsistem memori yang mendasarinya oleh konteks memori.

- `used` – Jumlah byte yang dijalankan untuk klien dari konteks memori.
- `instances` – Hitungan konteks yang ada saat ini dari jenis ini.

Catatan penggunaan

Fungsi ini menampilkan penggunaan konteks memori untuk setiap proses PostgreSQL. Beberapa proses diberi label `anonymous`. Proses tidak diekspos karena mengandung kata kunci yang dibatasi.

Fungsi ini tersedia mulai dengan versi Aurora PostgreSQL berikut:

- Versi 15.3 dan versi 15 yang lebih tinggi
- Versi 14.8 dan versi 14 yang lebih tinggi
- Versi 13.11 dan versi 13 yang lebih tinggi
- Versi 12.15 dan versi 12 yang lebih tinggi
- Versi 11.20 dan versi 11 yang lebih tinggi

Contoh

Contoh berikut menampilkan hasil pemanggilan fungsi `aurora_stat_memctx_usage`.

```
=> SELECT *
     FROM aurora_stat_memctx_usage();

    pid| name                             | allocated |   used  | instances
-------+----------------------------------+-----------+---------+-----------
123864 | Miscellaneous                    |     19520 |   15064 |         3
123864 | Aurora File Context              |      8192 |     616 |         1
123864 | Aurora WAL Context               |      8192 |     296 |         1
123864 | CacheMemoryContext               |    524288 |  422600 |         1
123864 | Catalog tuple context            |     16384 |   13736 |         1
123864 | ExecutorState                    |     32832 |   28304 |         1
123864 | ExprContext                      |      8192 |    1720 |         1
123864 | GWAL record construction         |      1024 |     832 |         1
123864 | MdSmgr                           |      8192 |     296 |         1
123864 | MessageContext                   |    532480 |  353832 |         1
123864 | PortalHeapMemory                 |      1024 |     488 |         1
123864 | PortalMemory                     |      8192 |     576 |         1
123864 | printtup                         |      8192 |     296 |         1
123864 | RelCache hash table entries      |      8192 |    8152 |         1
```

```
123864 | RowDescriptionContext           |      8192 |    1344 |         1
123864 | smgr relation context           |      8192 |     296 |         1
123864 | Table function arguments        |      8192 |     352 |         1
123864 | TopTransactionContext           |      8192 |     632 |         1
123864 | TransactionAbortContext         |     32768 |     296 |         1
123864 | WAL record construction         |     50216 |   43904 |         1
123864 | hash table                      |     65536 |   52744 |         6
123864 | Relation metadata               |    191488 |  124240 |        87
104992 | Miscellaneous                   |      9280 |    7728 |         3
104992 | Aurora File Context             |      8192 |     376 |         1
104992 | Aurora WAL Context              |      8192 |     296 |         1
104992 ||Autovacuum Launcher             |      8192 |     296 |         1
104992 | Autovacuum database list        |     16384 |     744 |         2
104992 | CacheMemoryContext              |    262144 |  140288 |         1
104992 | Catalog tuple context           |      8192 |     296 |         1
104992 | GWAL record construction        |      1024 |     832 |         1
104992 | MdSmgr                          |      8192 |     296 |         1
104992 | PortalMemory                    |      8192 |     296 |         1
104992 | RelCache hash table entries     |      8192 |     296 |         1
104992 | smgr relation context           |      8192 |     296 |         1
104992 | Autovacuum start worker (tmp)   |      8192 |     296 |         1
104992 | TopTransactionContext           |     16384 |     592 |         2
104992 | TransactionAbortContext         |     32768 |     296 |         1
104992 | WAL record construction         |     50216 |   43904 |         1
104992 | hash table                      |     49152 |   34024 |         4
(39 rows)
```

Beberapa kata kunci terbatas akan disembunyikan dan output akan terlihat sebagai berikut:

```
postgres=>SELECT *
     FROM aurora_stat_memctx_usage();

    pid| name                            | allocated |   used   | instances
-------+---------------------------------+-----------+----------+-----------
  5482 | anonymous                       |      8192 |     456 |         1
  5482 | anonymous                       |      8192 |     296 |         1
```

aurora_stat_optimized_reads_cache

Fungsi ini menunjukkan statistik cache berjenjang.

Sintaksis

```
aurora_stat_optimized_reads_cache()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- `total_size` – Total ukuran cache bacaan yang dioptimalkan.
- `used_size` – Ukuran halaman yang digunakan dalam cache bacaan yang dioptimalkan.

Catatan penggunaan

Fungsi ini tersedia di versi Aurora PostgreSQL berikut:

- Versi 15.4 dan versi 15 yang lebih tinggi
- Versi 14.9 dan versi 14 yang lebih tinggi

Contoh

Contoh berikut menampilkan output pada contoh instans r6gd.8xlarge:

```
=> select pg_size_pretty(total_size) as total_size, pg_size_pretty(used_size)
                as used_size from aurora_stat_optimized_reads_cache();
total_size | used_size
-----------+-----------
1054 GB    | 975 GB
```

aurora_stat_plans

Mengembalikan baris untuk setiap rencana eksekusi dilacak.

Sintaks

```
aurora_stat_plans(
    showtext
)
```

Argumen

- showtext - Tampilkan kueri dan rencana teks. Nilai yang valid adalah NULL, benar atau salah. True akan menampilkan kueri dan rencana teks.

Jenis pengembalian

Mengembalikan baris untuk setiap rencana dilacak yang berisi semua kolom dari `aurora_stat_statements` dan kolom tambahan berikut.

- planid - pengidentifikasi rencana
- explain_plan - jelaskan teks rencana
- plan_type:
  - `no plan`- tidak ada rencana yang ditangkap
  - `estimate`- rencana ditangkap dengan perkiraan biaya
  - `actual`- rencana ditangkap dengan EXPLORE ANALYSIS
- plan_captured_time — terakhir kali rencana ditangkap

Catatan penggunaan

`aurora_compute_plan_id`harus diaktifkan dan `pg_stat_statements` harus dalam `shared_preload_libraries` agar rencana dilacak.

Jumlah paket yang tersedia dikendalikan oleh nilai yang ditetapkan dalam `pg_stat_statements.max` parameter. Saat `compute_plan_id` diaktifkan, Anda dapat melacak paket hingga nilai yang ditentukan ini di`aurora_stat_plans`.

Fungsi ini tersedia dari Aurora PostgreSQL versi 14.10, 15.5, dan untuk semua versi lain yang lebih baru.

Contoh-contoh

Dalam contoh di bawah ini, dua rencana yang untuk pengidentifikasi kueri -5471422286312252535 ditangkap dan statistik pernyataan dilacak oleh planid.

```
db1=# select calls, total_exec_time, planid, plan_captured_time, explain_plan
db1-# from aurora_stat_plans(true)
```

```
db1-# where queryid = '-5471422286312252535'

calls    |  total_exec_time   |   planid    |      plan_captured_time       |
                explain_plan
---------+--------------------+-------------+-------------------------------
+-------------------------------------------------------------------
 1532632 |  3209846.097107853 |  1602979607 | 2023-10-31 03:27:16.925497+00 | Update on
 pgbench_branches                                        +
         |                    |             |                               |   ->
 Bitmap Heap Scan on pgbench_branches                       +
         |                    |             |                               |
 Recheck Cond: (bid = 76)                                 +
         |                    |             |                               |         -
>  Bitmap Index Scan on pgbench_branches_pkey          +
         |                    |             |                               |
     Index Cond: (bid = 76)
   61365 | 124078.18012200127 | -2054628807 | 2023-10-31 03:20:09.85429+00  | Update on
 pgbench_branches                                        +
         |                    |             |                               |   ->
 Index Scan using pgbench_branches_pkey on pgbench_branches+
         |                    |             |                               |
 Index Cond: (bid = 17)
```

aurora_stat_reset_wal_cache

Mengatur ulang penghitung untuk cache wal logis.

Sintaksis

Cara mengatur ulang slot tertentu

```
SELECT * FROM aurora_stat_reset_wal_cache('slot_name')
```

Cara mengatur ulang semua slot

```
SELECT * FROM aurora_stat_reset_wal_cache(NULL)
```

Argumen

`NULL` atau `slot_name`

Jenis pengembalian

Pesan status, string teks

- Atur ulang penghitung cache wal logis - Pesan sukses. Teks ini dikembalikan ketika fungsi berhasil.
- Slot replikasi tidak ditemukan. Coba lagi. – Pesan kegagalan. Teks ini dikembalikan ketika fungsi tidak berhasil.

Catatan penggunaan

Fungsi ini tersedia untuk versi berikut.

- Aurora PostgreSQL versi 14.5 dan lebih tinggi
- Aurora PostgreSQL versi 13.8 dan lebih tinggi
- Aurora PostgreSQL versi 12.12 dan lebih tinggi
- Aurora PostgreSQL versi 11.17 dan lebih tinggi

Contoh

Contoh berikut menggunakan fungsi `aurora_stat_reset_wal_cache` untuk mengatur ulang slot bernama `test_results`, dan kemudian mencoba untuk mengatur ulang slot yang tidak ada.

```
=> SELECT *
     FROM aurora_stat_reset_wal_cache('test_slot');
aurora_stat_reset_wal_cache
------------------------------------
 Reset the logical wal cache counter.
(1 row)
=> SELECT *
     FROM aurora_stat_reset_wal_cache('slot-not-exist');
aurora_stat_reset_wal_cache
-----------------------------------------------
 Replication slot not found. Please try again.
(1 row)
```

aurora_stat_statement

Menampilkan semua kolom `pg_stat_statements` dan menambahkan lebih banyak kolom pada akhirnya.

Sintaksis

```
aurora_stat_statements(showtext boolean)
```

Argumen

showtext boolean

Jenis pengembalian

Catatan SETOF dengan semua kolom `pg_stat_statements` dan kolom tambahan berikut. Untuk informasi selengkapnya tentang kolom `pg_stat_statements`, lihat `pg_stat_statements`.

Anda dapat mengatur ulang statistik untuk fungsi ini menggunakan `pg_stat_statements_reset()`.

- `storage_blks_read` – Jumlah total blok bersama yang dibaca dari penyimpanan aurora oleh pernyataan ini.
- `orcache_blks_hit` – Jumlah total hit cache pembacaan yang dioptimalkan oleh pernyataan ini.
- `storage_blk_read_time` – Jika `track_io_timing` diaktifkan, total waktu pernyataan yang dihabiskan untuk membaca blok file data dari penyimpanan aurora dilacak dalam milidetik. Jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.
- `local_blk_read_time` – Jika `track_io_timing` diaktifkan, total waktu pernyataan yang dihabiskan untuk membaca blok file data lokal dilacak dalam milidetik. Jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.
- `orcache_blk_read_time` – Jika `track_io_timing` diaktifkan, total waktu pernyataan yang dihabiskan untuk membaca blok file data dari cache pembacaan yang dioptimalkan dilacak dalam milidetik. Jika tidak, nilainya nol. Untuk informasi selengkapnya, lihat track_io_timing.

Catatan penggunaan

Anda harus membuat ekstensi `pg_stat_statements` terlebih dahulu untuk menerima data yang benar dari fungsi virtual ini.

Fungsi ini tersedia di versi Aurora PostgreSQL berikut:

- Versi 15.4 dan versi 15 yang lebih tinggi

- Versi 14.9 dan versi 14 yang lebih tinggi

Contoh-contoh

Contoh berikut menampilkan bagaimana fungsi ini membawa semua kolom pg_stat_statement dan menambahkan 5 kolom baru pada akhirnya:

```
=> select * from aurora_stat_statements(true) where queryid=-7342090857217643794;
-[ RECORD 1 ]---------+--------------------------------------------------
userid                | 10
dbid                  | 16419
toplevel              | t
queryid               | -7342090857217643794
query                 | CREATE TABLE quad_point_tbl AS                   +
                      |     SELECT point(unique1,unique2) AS p FROM tenk1
plans                 | 0
total_plan_time       | 0
min_plan_time         | 0
max_plan_time         | 0
mean_plan_time        | 0
stddev_plan_time      | 0
calls                 | 1
total_exec_time       | 571.844376
min_exec_time         | 571.844376
max_exec_time         | 571.844376
mean_exec_time        | 571.844376
stddev_exec_time      | 0
rows                  | 10000
shared_blks_hit       | 462
shared_blks_read      | 422
shared_blks_dirtied   | 0
shared_blks_written   | 55
local_blks_hit        | 0
local_blks_read       | 0
local_blks_dirtied    | 0
local_blks_written    | 0
temp_blks_read        | 0
temp_blks_written     | 0
blk_read_time         | 170.634621
blk_write_time        | 0
wal_records           | 0
wal_fpi               | 0
wal_bytes             | 0
```

```
storage_blks_read     | 47
orcache_blks_hit      | 375
storage_blk_read_time | 124.505772
local_blk_read_time   | 0
orcache_blk_read_time | 44.684038
```

aurora_stat_system_waits

Melaporkan informasi peristiwa tunggu untuk instans DB Aurora PostgreSQL.

Sintaksis

```
aurora_stat_system_waits()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF

Catatan penggunaan

Fungsi ini mengembalikan jumlah tunggu kumulatif dan waktu tunggu kumulatif untuk setiap peristiwa tunggu yang dihasilkan oleh instans DB yang saat ini terhubung.

Recordset yang dikembalikan mencakup kolom-kolom berikut:

- `type_id` – ID dari jenis peristiwa tunggu.
- `event_id` – ID dari peristiwa tunggu.
- `waits` – Jumlah peristiwa menunggu terjadi.
- `wait_time` – Jumlah total waktu dalam mikrodetik yang dihabiskan menunggu peristiwa ini.

Statistik yang dikembalikan oleh fungsi ini diatur ulang ketika instans DB dimulai ulang.

Contoh

Contoh berikut menampilkan hasil dari pemanggilan fungsi `aurora_stat_system_waits`.

```
=> SELECT *
```

```
    FROM aurora_stat_system_waits();
 type_id | event_id  |   waits   |  wait_time
---------+-----------+-----------+-------------
       1 |  16777219 |        11 |        12864
       1 |  16777220 |       501 |       174473
       1 |  16777270 |     53171 |     23641847
       1 |  16777271 |        23 |       319668
       1 |  16777274 |        60 |        12759
.
.
.
      10 | 167772231 |    204596 |    790945212
      10 | 167772232 |         2 |        47729
      10 | 167772234 |         1 |          888
      10 | 167772235 |         2 |           64
```

Contoh berikut menampilkan cara menggunakan fungsi ini bersama-sama dengan `aurora_stat_wait_event` dan `aurora_stat_wait_type` untuk menghasilkan hasil yang lebih mudah dibaca.

```
=> SELECT type_name,
            event_name,
            waits,
            wait_time
        FROM aurora_stat_system_waits()
NATURAL JOIN aurora_stat_wait_event()
NATURAL JOIN aurora_stat_wait_type();

 type_name |       event_name       |  waits   |  wait_time
-----------+------------------------+----------+-------------
 LWLock    | XidGenLock             |       11 |        12864
 LWLock    | ProcArrayLock          |      501 |       174473
 LWLock    | buffer_content         |    53171 |     23641847
 LWLock    | rdsutils               |        2 |        12764
 Lock      | tuple                  |    75686 |   2033956052
 Lock      | transactionid          |  1765147 |  47267583409
 Activity  | AutoVacuumMain         |   136868 |  56305604538
 Activity  | BgWriterHibernate      |     7486 |  55266949471
 Activity  | BgWriterMain           |     7487 |   1508909964
.
.
.
 IO        | SLRURead               |        3 |        11756
```

```
IO        | WALWrite               | 52544463 |    388850428
IO        | XactSync               |   187073 |    597041642
IO        | ClogRead               |        2 |        47729
IO        | OutboundCtrlRead       |        1 |          888
IO        | OutboundCtrlWrite      |        2 |           64
```

aurora_stat_wait_event

Daftar semua peristiwa tunggu yang didukung untuk Aurora PostgreSQL. Untuk informasi tentang perstiwa tunggu Aurora PostgreSQL, lihat Peristiwa tunggu Amazon Aurora PostgreSQL.

Sintaksis

```
aurora_stat_wait_event()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- type_id – ID dari jenis peristiwa tunggu.
- event_id – ID dari peristiwa tunggu.
- type_name – Nama jenis tunggu
- event_name – Nama peristiwa tunggu

Catatan penggunaan

Untuk melihat nama peristiwa dengan jenis peristiwa, bukan ID, gunakan fungsi ini bersama dengan fungsi lain seperti `aurora_stat_wait_type` dan `aurora_stat_system_waits`. Nama peristiwa tunggu yang dikembalikan oleh fungsi ini sama dengan yang dikembalikan oleh fungsi `aurora_wait_report`.

Contoh

Contoh berikut menampilkan hasil dari pemanggilan fungsi `aurora_stat_wait_event`.

```
=>  SELECT *
    FROM aurora_stat_wait_event();

 type_id | event_id  |               event_name
---------+-----------+-----------------------------------------
       1 |  16777216 | <unassigned:0>
       1 |  16777217 | ShmemIndexLock
       1 |  16777218 | OidGenLock
       1 |  16777219 | XidGenLock
.
.
.
       9 | 150994945 | PgSleep
       9 | 150994946 | RecoveryApplyDelay
      10 | 167772160 | BufFileRead
      10 | 167772161 | BufFileWrite
      10 | 167772162 | ControlFileRead
.
.
.
      10 | 167772226 | WALInitWrite
      10 | 167772227 | WALRead
      10 | 167772228 | WALSync
      10 | 167772229 | WALSyncMethodAssign
      10 | 167772230 | WALWrite
      10 | 167772231 | XactSync
.
.
.
      11 | 184549377 | LsnAllocate
```

Contoh berikut menggabungkan `aurora_stat_wait_type` dan `aurora_stat_wait_event` untuk mengembalikan nama jenis dan nama peristiwa untuk meningkatkan keterbacaan.

```
=> SELECT *
    FROM aurora_stat_wait_type() t
    JOIN aurora_stat_wait_event() e
      ON t.type_id = e.type_id;

 type_id | type_name | type_id | event_id  |               event_name
---------+-----------+---------+-----------+-----------------------------------------
       1 | LWLock    |       1 |  16777216 | <unassigned:0>
       1 | LWLock    |       1 |  16777217 | ShmemIndexLock
```

```
       1 | LWLock     |       1 |  16777218 | OidGenLock
       1 | LWLock     |       1 |  16777219 | XidGenLock
       1 | LWLock     |       1 |  16777220 | ProcArrayLock
.
.
.
       3 | Lock       |       3 |  50331648 | relation
       3 | Lock       |       3 |  50331649 | extend
       3 | Lock       |       3 |  50331650 | page
       3 | Lock       |       3 |  50331651 | tuple
.
.
.
      10 | IO         |      10 | 167772214 | TimelineHistorySync
      10 | IO         |      10 | 167772215 | TimelineHistoryWrite
      10 | IO         |      10 | 167772216 | TwophaseFileRead
      10 | IO         |      10 | 167772217 | TwophaseFileSync
.
.
.
      11 | LSN        |      11 | 184549376 | LsnDurable
```

aurora_stat_wait_type

Daftar semua jenis tunggu yang didukung untuk Aurora PostgreSQL.

Sintaksis

```
aurora_stat_wait_type()
```

Argumen

Tidak ada

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- type_id – ID dari jenis peristiwa tunggu.
- type_name – Nama jenis tunggu.

Catatan penggunaan

Untuk melihat nama peristiwa tunggu dengan jenis peristiwa tunggu, bukan ID, gunakan fungsi ini bersama dengan fungsi lain seperti `aurora_stat_wait_event` dan `aurora_stat_system_waits`. Nama peristiwa tunggu yang dikembalikan oleh fungsi ini sama dengan yang dikembalikan oleh fungsi `aurora_wait_report`.

Contoh

Contoh berikut menampilkan hasil dari pemanggilan fungsi `aurora_stat_wait_type`.

```
=> SELECT *
     FROM aurora_stat_wait_type();
 type_id | type_name
---------+-----------
       1 | LWLock
       3 | Lock
       4 | BufferPin
       5 | Activity
       6 | Client
       7 | Extension
       8 | IPC
       9 | Timeout
      10 | IO
      11 | LSN
```

aurora_version

Mengembalikan nilai string nomor versi Amazon Aurora Edisi Kompatibel PostgreSQL.

Sintaksis

```
aurora_version()
```

Argumen

Tidak ada

Jenis pengembalian

String CHAR atau VARCHAR

Catatan penggunaan

Fungsi ini menampilkan versi mesin basis data Amazon Aurora Edisi Kompatibel PostgreSQL. Nomor versi dikembalikan sebagai string yang diformat sebagai *mayor*.*minor*.*patch*. Untuk informasi selengkapnya tentang nomor versi Aurora PostgreSQL, lihat Nomor versi Aurora.

Anda dapat memilih kapan harus menerapkan peningkatan versi minor dengan mengatur jendela pemeliharaan untuk klaster DB Aurora PostgreSQL. Untuk mempelajari caranya, lihat Memelihara klaster DB Amazon Aurora.

Mulai dengan rilis Aurora PostgreSQL versi 13.3, 12.8, 11.13, 10.18, dan untuk semua versi lain yang lebih baru, nomor versi Aurora mengikuti nomor versi PostgreSQL. Untuk informasi selengkapnya tentang semua rilis Aurora PostgreSQL, lihat Pembaruan Amazon Aurora PostgreSQL di Catatan Rilis untuk Aurora PostgreSQL.

Contoh

Contoh berikut menunjukkan hasil pemanggilan fungsi `aurora_version` pada klaster DB Aurora PostgreSQL yang menjalankan PostgreSQL 12.7, Aurora PostgreSQL rilis 4.2 dan kemudian menjalankan fungsi yang sama pada klaster yang menjalankan Aurora PostgreSQL versi 13.3.

```
=> SELECT * FROM aurora_version();
aurora_version
----------------
 4.2.2
SELECT * FROM aurora_version();
aurora_version
----------------
 13.3.0
```

Contoh ini menunjukkan cara menggunakan fungsi dengan berbagai opsi untuk mendapatkan detail selengkapnya tentang versi Aurora PostgreSQL. Contoh ini memiliki nomor versi Aurora yang berbeda dari nomor versi PostgreSQL.

```
=> SHOW SERVER_VERSION;
 server_version
----------------
 12.7
(1 row)
```

```
=> SELECT * FROM aurora_version();
aurora_version
----------------
 4.2.2
(1 row)

=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
--------------------------
 12.7
(1 row)

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
------------------------------------------------------------------------------------------------
 PostgreSQL 12.7 on aarch64-unknown-linux-gnu, compiled by aarch64-unknown-linux-gnu-
gcc (GCC) 7.4.0, 64-bit
(1 row)

=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-----------------------------------------------
 Aurora: 4.2.2 Compatible with PostgreSQL: 12.7
(1 row)
```

Contoh berikutnya ini menggunakan fungsi dengan opsi yang sama pada contoh sebelumnya. Contoh ini memiliki nomor versi Aurora yang berbeda dari nomor versi PostgreSQL.

```
=> SHOW SERVER_VERSION;
server_version
----------------
 13.3

=> SELECT * FROM aurora_version();
aurora_version
----------------
 13.3.0
=> SELECT current_setting('server_version') AS "PostgreSQL Compatiblility";
PostgreSQL Compatiblility
```

```
---------------------------
 13.3

=> SELECT version() AS "PostgreSQL Compatiblility Full String";
PostgreSQL Compatiblility Full String
------------------------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
=> SELECT 'Aurora: '
    || aurora_version()
    || ' Compatible with PostgreSQL: '
    || current_setting('server_version') AS "Instance Version";
Instance Version
-------------------------------------------------------
 Aurora: 13.3.0 Compatible with PostgreSQL: 13.3
```

aurora_volume_logical_start_lsn

Mengembalikan nomor urutan log (LSN) yang digunakan untuk mengidentifikasi awal catatan dalam aliran log write-ahead (WAL) logis dari volume klaster Aurora.

Sintaksis

```
aurora_volume_logical_start_lsn()
```

Argumen

Tidak ada

Jenis pengembalian

`pg_lsn`

Catatan penggunaan

Fungsi ini mengidentifikasi awal catatan dalam aliran WAL logis untuk volume klaster Aurora tertentu. Anda dapat menggunakan fungsi ini saat melakukan peningkatan versi mayor menggunakan replikasi logis dan kloning cepat Aurora untuk menentukan LSN tempat snapshot atau klona basis data diambil. Setelah itu, Anda dapat menggunakan replikasi logis untuk terus mengalirkan data baru yang dicatat setelah LSN dan menyinkronkan perubahan dari penerbit ke pelanggan.

Untuk informasi selengkapnya tentang penggunaan replikasi logis untuk meningkatkan versi mayor, lihat Menggunakan replikasi logis untuk melakukan upgrade versi mayor untuk Aurora PostgreSQL.

Fungsi ini tersedia di versi Aurora PostgreSQL berikut:

- Versi 15.2 dan versi 15 yang lebih tinggi
- Versi 14.3 dan versi 14 yang lebih tinggi
- Versi 13.6 dan versi 13 yang lebih tinggi
- Versi 12.10 dan versi 12 yang lebih tinggi
- Versi 11.15 dan versi 11 yang lebih tinggi
- Versi 10.20 dan versi 10 yang lebih tinggi

Contoh

Anda dapat memperoleh nomor urutan log (LSN) menggunakan kueri berikut:

```
postgres=> SELECT aurora_volume_logical_start_lsn();

aurora_volume_logical_start_lsn
---------------
0/402E2F0
(1 row)
```

aurora_wait_report

Fungsi ini menunjukkan aktivitas peristiwa tunggu selama periode waktu tertentu.

Sintaksis

```
aurora_wait_report([time])
```

Argumen

waktu (opsional)

Waktu dalam hitungan detik. Default-nya 10 detik.

Jenis pengembalian

Catatan SETOF dengan kolom berikut:

- type_name – Nama jenis tunggu
- event_name – Nama peristiwa tunggu
- wait – Jumlah tunggu
- wait_time - Waktu tunggu dalam milidetik
- ms_per_wait - Rata-rata milidetik berdasarkan jumlah sekali tunggu
- waits_per_xact – Rata-rata tunggu berdasarkan jumlah satu transaksi
- ms_per_xact - Rata-rata milidetik berdasarkan jumlah transaksi

Catatan penggunaan

Fungsi ini tersedia pada rilis Aurora PostgreSQL 1.1 kompatibel dengan PostgreSQL 9.6.6 dan versi yang lebih tinggi.

Untuk menggunakan fungsi ini, Anda harus terlebih dahulu membuat ekstensi `aurora_stat_utils` Aurora PostgreSQL, sebagai berikut:

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

Untuk informasi selengkapnya tentang versi ekstensi Aurora PostgreSQL, lihat Versi ekstensi untuk Amazon Aurora PostgreSQL di Catatan Rilis untuk Aurora PostgreSQL.

Fungsi ini menghitung peristiwa tunggu level instans dengan membandingkan dua snapshot data statistik dari fungsi aurora_stat_system_waits() dan PostgreSQL Statistics Views pg_stat_database.

Untuk informasi selengkapnya tentang `aurora_stat_system_waits()` dan `pg_stat_database`, lihat Pengumpul Statistik di dokumentasi PostgreSQL.

Saat dijalankan, fungsi ini mengambil snapshot awal, menunggu jumlah detik yang ditentukan, dan kemudian mengambil snapshot kedua. Fungsi tersebut membandingkan dua snapshot dan mengembalikan perbedaannya. Perbedaan ini mewakili aktivitas instans untuk interval waktu tersebut.

Pada instans penulis, fungsi ini juga menampilkan jumlah transaksi yang dilakukan dan TPS (transaksi per detik). Fungsi ini mengembalikan informasi pada tingkat instans dan mencakup semua basis data pada instans.

Contoh

Contoh ini menunjukkan cara membuat ekstensi aurora_stat_utils agar dapat menggunakan fungsi aurora_log_report.

```
=> CREATE extension aurora_stat_utils;
CREATE EXTENSION
```

Contoh ini menunjukkan cara memeriksa laporan tunggu selama 10 detik.

```
=> SELECT *
     FROM aurora_wait_report();
NOTICE:  committed 34 transactions in 10 seconds (tps 3)
 type_name |    event_name     | waits | wait_time | ms_per_wait | waits_per_xact |
 ms_per_xact
-----------+-------------------+-------+-----------+-------------+----------------
+-------------
 Client    | ClientRead        |    26 |  30003.00 |    1153.961 |           0.76 |
 882.441
 Activity  | WalWriterMain     |    50 |  10051.32 |     201.026 |           1.47 |
 295.627
 Timeout   | PgSleep           |     1 |  10049.52 |   10049.516 |           0.03 |
 295.574
 Activity  | BgWriterHibernate |     1 |  10048.15 |   10048.153 |           0.03 |
 295.534
 Activity  | AutoVacuumMain    |    18 |   9941.66 |     552.314 |           0.53 |
 292.402
 Activity  | BgWriterMain      |     1 |    201.09 |     201.085 |           0.03 |
   5.914
 IO        | XactSync          |    15 |     25.34 |       1.690 |           0.44 |
   0.745
 IO        | RelationMapRead   |    12 |      0.54 |       0.045 |           0.35 |
   0.016
 IO        | WALWrite          |    84 |      0.21 |       0.002 |           2.47 |
   0.006
 IO        | DataFileExtend    |     1 |      0.02 |       0.018 |           0.03 |
   0.001
```

Contoh ini menunjukkan cara memeriksa laporan tunggu selama 60 detik.

```
=> SELECT *
     FROM aurora_wait_report(60);
NOTICE:  committed 1544 transactions in 60 seconds (tps 25)
 type_name |       event_name       |  waits  | wait_time | ms_per_wait |
 waits_per_xact | ms_per_xact
-----------+------------------------+---------+-----------+-------------
+----------------+-------------
 Lock      | transactionid          |    6422 | 477000.53 |      74.276 |
 4.16 |     308.938
 Client    | ClientRead             |    8265 | 270752.99 |      32.759 |
 5.35 |     175.358
 Activity  | CheckpointerMain       |       1 |  60100.25 |   60100.246 |
 0.00 |      38.925
 Timeout   | PgSleep                |       1 |  60098.49 |   60098.493 |
 0.00 |      38.924
 Activity  | WalWriterMain          |     296 |  60010.99 |     202.740 |
 0.19 |      38.867
 Activity  | AutoVacuumMain         |     107 |  59827.84 |     559.139 |
 0.07 |      38.749
 Activity  | BgWriterMain           |     290 |  58821.83 |     202.834 |
 0.19 |      38.097
 IO        | XactSync               |    1295 |  55220.13 |      42.641 |
 0.84 |      35.764
 IO        | WALWrite               | 6602259 |  47810.94 |       0.007 |
 4276.07 |      30.966
 Lock      | tuple                  |     473 |  29880.67 |      63.173 |
 0.31 |      19.353
 LWLock    | buffer_mapping         |     142 |   3540.13 |      24.930 |
 0.09 |       2.293
 Activity  | BgWriterHibernate      |     290 |   1124.15 |       3.876 |
 0.19 |       0.728
 IO        | BufFileRead            |    7615 |    618.45 |       0.081 |
 4.93 |       0.401
 LWLock    | buffer_content         |      73 |    345.93 |       4.739 |
 0.05 |       0.224
 LWLock    | lock_manager           |      62 |    191.44 |       3.088 |
 0.04 |       0.124
 IO        | RelationMapRead        |      72 |      5.16 |       0.072 |
 0.05 |       0.003
 LWLock    | ProcArrayLock          |       1 |      2.01 |       2.008 |
 0.00 |       0.001
```

```
IO        | ControlFileWriteUpdate |       2 |      0.03 |       0.013 |
0.00 |       0.000
IO        | DataFileExtend         |       1 |      0.02 |       0.018 |
0.00 |       0.000
IO        | ControlFileSyncUpdate  |       1 |      0.00 |       0.000 |
0.00 |       0.000
```

Parameter Amazon Aurora PostgreSQL

Anda mengelola klaster DB Amazon Aurora Anda dengan cara yang sama seperti Anda mengelola instans DB Amazon RDS lain, dengan menggunakan parameter dalam grup parameter DB. Namun, Amazon Aurora berbeda dari Amazon RDS karena klaster DB Aurora memiliki beberapa instans DB. Beberapa parameter yang Anda gunakan untuk mengelola klaster DB Amazon Aurora Anda berlaku untuk seluruh klaster, sementara parameter lain hanya berlaku untuk instans DB tertentu dalam klaster DB, sebagai berikut:

- Grup parameter klaster DB — Grup parameter klaster DB berisi serangkaian parameter konfigurasi engine yang berlaku di seluruh klaster DB Aurora. Misalnya, manajemen cache klaster adalah fitur dari klaster DB Aurora yang dikendalikan oleh parameter `apg_ccm_enabled` yang merupakan bagian dari grup parameter klaster DB. Grup parameter klaster DB juga berisi pengaturan default untuk grup parameter DB untuk instans DB yang membentuk klaster.
- Grup parameter DB — Grup parameter DB adalah kumpulan nilai konfigurasi mesin yang berlaku untuk instans DB tertentu dari jenis mesin tersebut. Grup parameter DB untuk mesin DB PostgreSQL digunakan oleh instans RDS for PostgreSQL dan klaster DB Aurora PostgreSQL. Pengaturan konfigurasi ini berlaku untuk properti yang dapat beragam di antara instans DB dalam klaster Aurora, misalnya ukuran untuk buffer memori.

Anda mengelola parameter tingkat klaster dalam grup parameter klaster DB. Anda mengelola parameter tingkat instans dalam grup parameter klaster DB. Anda dapat mengelola parameter menggunakan konsol Amazon RDS, API Amazon RDS AWS CLI, atau Amazon RDS. Ada perintah terpisah untuk mengelola parameter tingkat klaster dan parameter tingkat instans.

- Untuk mengelola parameter tingkat cluster dalam grup parameter cluster DB, gunakan perintah - group. modify-db-cluster-parameter AWS CLI
- Untuk mengelola parameter tingkat instance dalam grup parameter DB untuk instans DB di cluster DB, gunakan perintah. modify-db-parameter-group AWS CLI

Untuk mempelajari selengkapnya AWS CLI, lihat Menggunakan AWS CLI dalam Panduan AWS Command Line Interface Pengguna.

Lihat informasi lebih lanjut tentang grup parameter di Bekerja dengan grup parameter.

Melihat klaster DB dan parameter DB Aurora PostgreSQL

Anda dapat melihat semua grup parameter default yang tersedia untuk instans DB RDS for PostgreSQL dan untuk klaster DB Aurora PostgreSQL di AWS Management Console. Grup parameter default untuk semua mesin DB dan tipe dan versi cluster DB terdaftar untuk setiap AWS Wilayah. Setiap grup parameter kustom juga akan ditampilkan.

Daripada melihat di AWS Management Console, Anda juga dapat mencantumkan parameter yang terdapat dalam grup parameter cluster DB dan grup parameter DB dengan menggunakan AWS CLI atau Amazon RDS API. Misalnya, untuk membuat daftar parameter dalam grup parameter cluster DB Anda menggunakan describe-db-cluster-parameters AWS CLI perintah sebagai berikut:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12
```

Perintah ini akan menghasilkan deskripsi JSON terperinci untuk setiap parameter. Untuk mengurangi jumlah informasi yang dihasilkan, Anda dapat menentukan apa yang Anda inginkan dengan menggunakan opsi `--query`. Misalnya, Anda bisa mendapatkan nama parameter, deskripsinya, dan nilai yang diizinkan untuk grup parameter klaster DB Aurora PostgreSQL 12 default sebagai berikut:

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Untuk Windows:

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name
 default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Grup parameter klaster DB Aurora mencakup grup parameter instans DB dan nilai default untuk mesin Aurora DB tertentu. Anda bisa mendapatkan daftar parameter DB dari grup parameter default

Aurora PostgreSQL default yang sama dengan menggunakan perintah seperti yang ditunjukkan berikut. describe-db-parameters AWS CLI

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 \
    --query 'Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Untuk Windows:

```
aws rds describe-db-parameters --db-parameter-group-name default.aurora-postgresql12 ^
    --query "Parameters[].
[{ParameterName:ParameterName,Description:Description,ApplyType:ApplyType,AllowedValues:Allowed
```

Perintah sebelumnya menghasilkan daftar parameter dari klaster DB atau grup parameter DB dengan deskripsi dan detail lainnya yang ditentukan dalam kueri. Berikut adalah respons contohnya:

```
[
    [
        {
            "ParameterName": "apg_enable_batch_mode_function_execution",
            "ApplyType": "dynamic",
            "Description": "Enables batch-mode functions to process sets of rows at a
 time.",
            "AllowedValues": "0,1"
        }
    ],
    [
        {
            "ParameterName": "apg_enable_correlated_any_transform",
            "ApplyType": "dynamic",
            "Description": "Enables the planner to transform correlated ANY Sublink
 (IN/NOT IN subquery) to JOIN when possible.",
            "AllowedValues": "0,1"
        }
    ],...
```

Berikut ini adalah tabel yang berisi nilai untuk parameter klaster DB default dan parameter DB untuk Aurora PostgreSQL versi 14.

Parameter tingkat klaster Aurora PostgreSQL

Anda dapat melihat parameter tingkat cluster yang tersedia untuk versi PostgreSQL Aurora tertentu menggunakan konsol AWS Manajemen, CLI, atau Amazon RDS API. AWS Untuk informasi tentang cara melihat parameter dalam grup parameter klaster DB Aurora PostgreSQL di konsol RDS, lihat Melihat nilai parameter untuk grup parameter klaster DB.

Beberapa parameter tingkat klaster tidak tersedia di semua versi dan beberapa sedang ditiadakan. Untuk informasi tentang cara melihat parameter versi PostgreSQL Aurora tertentu, lihat Melihat klaster DB dan parameter DB Aurora PostgreSQL.

Misalnya, tabel berikut mencantumkan parameter yang tersedia di grup parameter klaster DB default untuk Aurora PostgreSQL versi 14. Jika Anda membuat klaster DB PostgreSQL Aurora tanpa menentukan grup parameter DB kustom Anda sendiri, klaster DB Anda akan dibuat menggunakan grup parameter klaster DB Aurora default untuk versi yang dipilih, seperti `default.aurora-postgresql14`, `default.aurora-postgresql13`, dan sebagainya.

Untuk daftar parameter instans DB untuk grup parameter klaster DB default yang sama ini, lihat Parameter tingkat instans Aurora PostgreSQL.

Nama parameter	Deskripsi	Default
ansi_constraint_trigger_ordering	Mengubah urutan pengaktifan pemicu pembatasan agar kompatibel dengan standar ANSI SQL.	–
ansi_force_foreign_key_checks	Memastikan tindakan referensial seperti penghapusan kaskade atau pembaruan kaskade akan selalu terjadi terlepas dari berbagai konteks pemicu yang ada untuk tindakan tersebut.	–
ansi_qualified_update_set_target	Mendukung pengkualifikasi tabel dan skema di UPDATE ... Pernyataan SET.	–
apg_ccm_enabled	Mengaktifkan atau menonaktifkan manajemen cache klaster untuk klaster.	–

Nama parameter	Deskripsi	Default
apg_enable_batch_mode_function_execution	Mengaktifkan fungsi mode batch untuk memproses set baris pada suatu waktu.	–
apg_enable_correlated_any_transform	Memungkinkan perencana mentransformasikan Subtautan ANY terkorelasi (subkueri IN/NOT IN) menjadi JOIN jika memungkinkan.	–
apg_enable_function_migration	Memungkinkan perencana memigrasikan fungsi skalar yang memenuhi syarat ke klausa FROM.	–
apg_enable_not_in_transform	Memungkinkan perencana mentransformasikan subkueri NOT IN menjadi ANTI JOIN jika memungkinkan.	–
apg_enable_remove_redundant_inner_joins	Memungkinkan perencana menghapus inner join yang redundan.	–
apg_enable_semijoin_push_down	Memungkinkan penggunaan filter semijoin untuk hash join.	–
apg_plan_mgmt.capture_plan_baselines	Mengambil mode acuan dasar rencana. manual - mengaktifkan pengambilan rencana untuk pernyataan SQL apa pun, off - menonaktifkan pengambilan rencana, automatic - mengaktifkan pengambilan rencana untuk pernyataan dalam pg_stat_statement yang memenuhi kriteria kelayakan.	off
apg_plan_mgmt.max_databases	Menetapkan jumlah maksimum basis data yang dapat mengelola kueri menggunakan apg_plan_mgmt.	10
apg_plan_mgmt.max_plans	Menetapkan jumlah maksimum rencana yang dapat di-cache oleh apg_plan_mgmt.	10000

Nama parameter	Deskripsi	Default
apg_plan_mgmt.plan_retention_period	Jumlah hari maksimum sejak rencana last_used sebelum rencana akan dihapus secara otomatis.	32
apg_plan_mgmt.unapproved_plan_execution_threshold	Perkiraan total biaya rencana yang jika tidak tercapai akan membuat rencana yang Tidak Disetujui dieksekusi.	0
apg_plan_mgmt.use_plan_baselines	Menggunakan hanya rencana yang disetujui atau tetap untuk pernyataan terkelola.	false
application_name	Menetapkan nama aplikasi yang akan dilaporkan dalam statistik dan log.	–
array_nulls	Mengaktifkan input elemen NULL dalam array.	–
aurora_compute_plan_id	Memantau rencana eksekusi kueri untuk mendeteksi rencana eksekusi yang berkontribusi pada pemuatan basis data saat ini dan untuk melacak statistik kinerja rencana eksekusi dari waktu ke waktu. Untuk informasi selengkapnya, lihat Memantau rencana eksekusi kueri untuk Aurora PostgreSQL.	on
authentication_timeout	(s) Menetapkan waktu maksimum yang diizinkan untuk menyelesaikan autentikasi klien.	–
auto_explain.log_analysis	Menggunakan EXPLORE ANALYSIS untuk logging rencana.	–
auto_explain.log_buffers	Mencatat log penggunaan buffer.	–
auto_explain.log_format	Menggunakan EXPLAIN untuk format yang akan digunakan untuk logging rencana.	–

Nama parameter	Deskripsi	Default
auto_explain.log_min_duration	Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat rencana dicatat.	–
auto_explain.log_nested_statement	Mencatat log pernyataan bersarang.	–
auto_explain.log_timing	Mengumpulkan data waktu, bukan hanya hitungan baris.	–
auto_explain.log_trigger	Menyertakan statistik pemicu dalam rencana.	–
auto_explain.log_verbose	Menggunakan EXPLORE VERBOSE untuk logging rencana.	–
auto_explain.sample_rate	Pecahan kueri untuk diproses.	–
autovacuum	Memulai subproses autovacuum.	–
autovacuum_analyze_scale_factor	Jumlah penyisipan, pembaruan, atau penghapusan tuple sebelum analisis sebagai pecahan dari reltuple.	0,05
autovacuum_analyze_threshold	Jumlah minimum penyisipan tuple, pembaruan atau penghapusan sebelum analisis.	–
autovacuum_freeze_max_age	Usia saat tabel akan di-autovacuum untuk mencegah wraparound ID transaksi.	–
autovacuum_max_workers	Menetapkan jumlah maksimum proses pekerja autovacuum yang berjalan secara bersamaan.	TERBESAR (DB InstanceClassMemory /64371566592 ,3)
autovacuum_multixact_freeze_max_age	Usia multixact saat tabel perlu di-autovacuum untuk mencegah wraparound multixact.	–

Nama parameter	Deskripsi	Default
autovacuum_naptime	(s) Waktu untuk tidur di antara pelaksanaan autovacuum.	5
autovacuum_vacuum_cost_delay	(ms) Penundaan biaya vacuum dalam milidetik, untuk autovacuum.	5
autovacuum_vacuum_cost_limit	Jumlah biaya vacuum yang tersedia sebelum napping, untuk autovacuum.	TERBESAR (log (DB InstanceClassMemory /21474836480) * 600,200)
autovacuum_vacuum_insert_scale_factor	Jumlah penyisipan tuple sebelum vacuum sebagai pecahan dari reltuple.	–
autovacuum_vacuum_insert_threshold	Jumlah minimum penyisipan tuple sebelum vacuum, atau -1 untuk menonaktifkan penyisipan vacuum.	–
autovacuum_vacuum_scale_factor	Jumlah pembaruan atau penghapusan tuple sebelum vacuum sebagai pecahan dari reltuple.	0.1
autovacuum_vacuum_threshold	Jumlah minimum pembaruan atau penghapusan tuple sebelum vacuum.	–
autovacuum_work_mem	(kB) Menetapkan memori maksimum yang akan digunakan oleh setiap proses pekerja autovacuum.	TERBESAR (DB InstanceClassMemory /32768 ,131072)
babelfishpg_tds.default_server_name	Nama server Babelfish default	Microsoft SQL Server
babelfishpg_tds.listen_addresses	Menetapkan nama host atau alamat IP untuk mendengarkan TDS.	*
babelfishpg_tds.port	Menetapkan port TCP TDS yang didengarkan server.	1433

Nama parameter	Deskripsi	Default
babelfishpg_tds.tds_debug_log_level	Menetapkan tingkat logging di TDS, 0 menonaktifkan logging	1
babelfishpg_tds.tds_default_numeric_precision	Menetapkan presisi default jenis numerik yang akan dikirim dalam metadata kolom TDS jika mesin tidak menentukannya.	38
babelfishpg_tds.tds_default_numeric_scale	Menetapkan skala default jenis numerik yang akan dikirim dalam metadata kolom TDS jika mesin tidak menentukannya.	8
babelfishpg_tds.tds_default_packet_size	Integer yang menetapkan ukuran paket default untuk semua klien SQL Server yang terhubung.	4096
babelfishpg_tds.tds_default_protocol_version	Integer yang menetapkan versi protokol TDS default untuk semua klien yang terhubung.	DEFAULT
babelfishpg_tds.tds_ssl_encrypt	Menetapkan opsi Enkripsi SSL	0
babelfishpg_tds.tds_ssl_max_protocol_version	Menetapkan versi protokol SSL/TLS maksimum yang akan digunakan untuk sesi tds.	TLSv1.2
babelfishpg_tds.tds_ssl_min_protocol_version	Menetapkan versi protokol SSL/TLS minimum yang akan digunakan untuk sesi tds.	TLSv1
babelfishpg_tsql.default_locale	Lokalitas default yang akan digunakan untuk kolasi yang dibuat oleh CREATE COLLATION.	en-US
babelfishpg_tsql.migration_mode	Mendefinisikan apakah lebih dari satu basis data pengguna dapat didukung	single-db
babelfishpg_tsql.server_collation_name	Nama kolasi server default	sql_latin1_general_cp1_ci_as

Nama parameter	Deskripsi	Default
babelfishpg_tsql.version	Menetapkan output dari variabel @@VERSION	default
backend_flush_after	(8Kb) Jumlah halaman yang jika terpenuhi akan membuat penulisan yang dilakukan sebelumnya dibuang ke disk.	–
backslash_quote	Menetapkan apakah \\ diizinkan dalam string literal.	–
backtrace_functions	Mencatat log pelacakan mundur untuk kesalahan dalam fungsi ini.	–
bytea_output	Menetapkan format output untuk bytea.	–
check_function_bodies	Memeriksa isi fungsi selama CREATE FUNCTION.	–
client_connection_check_interval	Menetapkan interval waktu di antara pemeriksaan pemutusan koneksi saat menjalankan kueri.	–
client_encoding	Menetapkan pengenkodean set karakter klien.	UTF8
client_min_messages	Menetapkan tingkat pesan yang dikirimkan ke klien.	–
compute_query_id	Menghitung pengidentifikasi kueri.	auto
config_file	Menetapkan file konfigurasi utama server.	/rdsdbdata/config/postgresql.conf
constraint_exclusion	Memungkinkan perencana menggunakan pembatasan untuk mengoptimalkan kueri.	–
cpu_index_tuple_cost	Menetapkan perkiraan perencana untuk biaya pemrosesan setiap entri indeks selama pemindaian indeks.	–

Nama parameter	Deskripsi	Default
cpu_operator_cost	Menetapkan perkiraan perencana untuk biaya pemrosesan setiap panggilan operator atau fungsi.	–
cpu_tuple_cost	Menetapkan perkiraan perencana untuk biaya pemrosesan setiap tuple (baris).	–
cron.database_name	Menetapkan basis data untuk menyimpan tabel metadata pg_cron	postgres
cron.log_run	Mencatat log semua pelaksanaan pekerjaan ke tabel job_run_details	on
cron.log_statement	Mencatat log semua pernyataan cron sebelum eksekusi.	off
cron.max_running_jobs	Jumlah maksimum pekerjaan yang dapat dijalankan secara bersamaan.	5
cron.use_background_workers	Mengaktifkan pekerja latar belakang untuk pg_cron	on
cursor_tuple_fraction	Menetapkan perkiraan perencana untuk pecahan dari baris kursor yang akan diambil.	–
data_directory	Menetapkan direktori data server.	/rdsdbdata/db
datestyle	Menetapkan format tampilan untuk nilai tanggal dan waktu.	–
db_user_namespace	Mengaktifkan nama pengguna per basis data.	–
deadlock_timeout	(ms) Menetapkan waktu untuk menunggu lock sebelum memeriksa deadlock.	–
debug_pretty_print	Mengindentasi tampilan penguraian dan hierarki rencana.	–

Nama parameter	Deskripsi	Default
debug_print_parse	Mencatat log hierarki penguraian setiap kueri.	–
debug_print_plan	Mencatat log rencana eksekusi setiap kueri.	–
debug_print_rewritten	Mencatat log hierarki penguraian yang ditulis ulang masing-masing kueri.	–
default_statistics_target	Menetapkan target statistik default.	–
default_tablespace	Menetapkan ruang tabel default untuk membuat tabel dan indeks.	–
default_toast_compression	Menetapkan metode kompresi default untuk nilai kompresibel.	–
default_transaction_deferrable	Menetapkan status default yang dapat ditangguhkan untuk transaksi baru.	–
default_transaction_isolation	Menetapkan tingkat isolasi transaksi untuk setiap transaksi baru.	–
default_transaction_read_only	Menetapkan status hanya baca default untuk transaksi baru.	–
effective_cache_size	(8kB) Menetapkan asumsi perencana tentang ukuran cache disk.	JUMLAH (InstanceClassMemoryDB/12038, -50003)
effective_io_concurrency	Jumlah permintaan serentak yang dapat ditangani secara efisien oleh subsistem disk.	–
enable_async_append	Memungkinkan perencana menggunakan rencana async append.	–
enable_bitmapscan	Memungkinkan perencana menggunakan rencana bitmap-scan.	–

Nama parameter	Deskripsi	Default
enable_gathermerge	Memungkinkan perencana menggunakan rencana gather merge.	–
enable_hashagg	Memungkinkan perencana menggunakan rencana hashed aggregation.	–
enable_hashjoin	Memungkinkan perencana menggunakan rencana hash join.	–
enable_incremental_sort	Memungkinkan perencana menggunakan langkah-langkah pengurutan inkremental.	–
enable_indexonlyscan	Memungkinkan perencana menggunakan index-only-scan rencana.	–
enable_indexscan	Memungkinkan perencana menggunakan rencana index-scan.	–
enable_material	Memungkinkan perencana menggunakan materialization.	–
aktifkan_memoize	Memungkinkan perencana menggunakan memoization	–
enable_mergejoin	Memungkinkan perencana menggunakan rencana merge join.	–
enable_nestloop	Memungkinkan perencana menggunakan rencana nested-loop join.	–
enable_parallel_append	Mengaktifkan penggunaan perencana atas rencana parallel append.	–
aktifkan_parallel_hash	Mengaktifkan penggunaan perencana atas rencana parallel hash.	–
enable_partition_pruning	Mengaktifkan pemangkasan partisi plan-time dan run-time.	–

Nama parameter	Deskripsi	Default
enable_partitionwise_aggregate	Mengaktifkan agregasi dan pengelompokan partitionwise.	–
enable_partitionwise_join	Mengaktifkan partitionwise join.	–
enable_seqscan	Memungkinkan perencana menggunakan rencana sequential-scan.	–
enable_sort	Memungkinkan perencana menggunakan langkah-langkah penyortiran eksplisit.	–
enable_tidscan	Memungkinkan perencana menggunakan rencana TID scan.	–
escape_string_warning	Memperingatkan tentang escape garis miring dalam string literal biasa.	–
exit_on_error	Mengakhiri sesi saat terjadi kesalahan apa pun.	–
extra_float_digits	Menetapkan jumlah digit yang ditampilkan untuk nilai floating-point.	–
force_parallel_mode	Memaksa penggunaan fasilitas kueri paralel.	–
from_collapse_limit	Menetapkan ukuran FROM-list yang jika terlampaui akan membuat subkueri tidak ditutup.	–
geqo	Memungkinkan optimisasi kueri genetik.	–
geqo_effort	GEQO: upaya digunakan untuk mengatur nilai default untuk parameter GEQO lainnya.	–
geqo_generations	GEQO: jumlah iterasi algoritma.	–
geqo_pool_size	GEQO: jumlah individu dalam populasi.	–

Nama parameter	Deskripsi	Default
geqo_seed	GEQO: seed untuk pemilihan jalur acak.	–
geqo_selection_bias	GEQO: tekanan selektif di dalam populasi.	–
geqo_threshold	Menetapkan ambang batas item FROM yang jika terlampaui akan membuat GEQO digunakan.	–
gin_fuzzy_search_limit	Menetapkan hasil maksimum yang diizinkan untuk pencarian persis oleh GIN.	–
gin_pending_list_limit	(kB) Menetapkan ukuran maksimum daftar tertunda untuk indeks GIN.	–
hash_mem_multiplier	Beberapa work_mem yang akan digunakan untuk tabel hash.	–
hba_file	Menetapkan file konfigurasi hba server.	/rdsdbdata/config/pg_hba.conf
hot_standby_feedback	Memungkinkan umpan balik dari hot standby ke primer yang akan menghindari konflik kueri.	on
huge_pages	Mengurangi overhead saat instans DB menangani potongan besar memori yang berdekatan, seperti yang digunakan oleh buffer bersama. Parameter ini diaktifkan secara default untuk semua kelas instans DB selain kelas instans t3.medium, db.t3.large, db.t4g.medium, db.t4g.large.	on
ident_file	Menetapkan file konfigurasi ident server.	/rdsdbdata/config/pg_ident.conf
idle_in_transaction_session_timeout	(ms) Menetapkan durasi maksimum yang diizinkan untuk setiap transaksi idling.	86400000

Nama parameter	Deskripsi	Default
idle_session_timeout	Mengakhiri setiap sesi yang telah idle (yaitu, menunggu kueri klien), tetapi tidak dalam transaksi terbuka, lebih lama dari jumlah waktu yang ditentukan	–
intervalstyle	Menetapkan format tampilan untuk nilai interval.	–
join_collapse_limit	Menetapkan ukuran FORM-list yang jika terlampaui akan membuat konstruk JOIN tidak diratakan.	–
krb_caseins_users	Menetapkan apakah nama pengguna GSSAPI (Generic Security Service API) harus diperlakukan tanpa peka huruf besar/kecil (true) atau tidak. Secara default, parameter ini disetel ke false, sehingga Kerberos mengharapkan nama pengguna yang peka huruf besar/kecil. Untuk informasi selengkapnya, lihat GSSAPI Authentication dalam dokumentasi PostgreSQL.	false
lc_messages	Menetapkan dalam bahasa apa pesan akan ditampilkan.	–
lc_monetary	Menetapkan lokalitas untuk memformat jumlah uang.	–
lc_numeric	Menetapkan lokalitas untuk memformat angka.	–
lc_time	Menetapkan lokalitas untuk memformat nilai tanggal dan waktu.	–
listen_addresses	Menetapkan nama host atau alamat IP untuk didengarkan.	*

Nama parameter	Deskripsi	Default
lo_compat_privileges	Mengaktifkan mode kompatibilitas mundur untuk pemeriksaan hak akses pada objek besar.	0
log_autovacuum_min_duration	(ms) Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat tindakan autovacuum dicatat.	10000
log_connections	Mencatat log setiap koneksi yang berhasil.	–
log_destination	Menetapkan tujuan untuk output log server.	stderr
log_directory	Menetapkan direktori tujuan untuk file log.	/rdsdbdata/log/error
log_disconnections	Mencatat log akhir sebuah sesi, termasuk durasinya.	–
log_duration	Mencatat log durasi setiap pernyataan SQL yang diselesaikan.	–
log_error_verbosity	Menetapkan verbositas pesan yang dicatat.	–
log_executor_stats	Menulis statistik performa pengeksekusi ke log server.	–
log_file_mode	Menetapkan izin file untuk file log.	0644
log_filename	Menetapkan pola nama file untuk file log.	postgresql.log.%Y-%m-%d-%H%M
logging_collector	Mulai subproses untuk menangkap output stderr dan/atau csvlog ke dalam file log.	1
log_hostname	Mencatat log nama host dalam log koneksi.	0
logical_decoding_work_mem	(kB) Memori sebanyak ini dapat digunakan oleh setiap buffer penyusun ulang internal sebelum dialirkan ke disk.	–

Nama parameter	Deskripsi	Default
log_line_prefix	Mengontrol informasi yang ditambahkan ke awal setiap baris log.	%t:%r:%u@%d:%p]:
log_lock_waits	Mencatat log waktu tunggu lock yang panjang.	–
log_min_duration_sample	(ms) Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat sampel pernyataan dicatat. Pengambilan sampel ditentukan oleh log_statement_sample_rate.	–
log_min_duration_statement	(ms) Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat pernyataan dicatat.	–
log_min_error_statement	Menyebabkan semua pernyataan yang menghasilkan kesalahan pada atau di atas tingkat ini dicatat.	–
log_min_messages	Menetapkan tingkat pesan yang dicatat.	–
log_parameter_max_length	(B) Saat mencatat pernyataan, membatasi nilai parameter yang dicatat ke N byte pertama.	–
log_parameter_max_length_on_error	(B) Saat melaporkan kesalahan, membatasi nilai parameter yang dicatat ke N byte pertama.	–
log_parser_stats	Menulis statistik performa pengurai ke log server.	–
log_planner_stats	Menulis statistik performa perencana ke log server.	–
log_replication_commands	Mencatat log setiap perintah replikasi.	–
log_rotation_age	(min) Rotasi file log otomatis akan terjadi setelah N menit.	60

Nama parameter	Deskripsi	Default
log_rotation_size	(kB) Rotasi file log otomatis akan terjadi setelah N kilobyte.	100000
log_statement	Menetapkan jenis pernyataan yang dicatat.	–
log_statement_sample_rate	Pecahan dari pernyataan yang melebihi log_min_duration_sample yang akan dicatat.	–
log_statement_stats	Menulis statistik performa kumulatif ke log server.	–
log_temp_files	(kB) Mencatat log penggunaan file sementara yang lebih besar dari jumlah kilobyte ini.	–
log_timezone	Menetapkan zona waktu yang akan digunakan dalam pesan log.	UTC
log_transaction_sample_rate	Menetapkan pecahan transaksi yang akan dicatat untuk transaksi baru.	–
log_truncate_on_rotation	Memotong file log yang ada dengan nama yang sama selama rotasi log.	0
maintenance_io_concurrency	Varian effective_io_concurrency yang digunakan untuk pekerjaan pemeliharaan.	1
maintenance_work_mem	(kB) Menetapkan memori maksimum yang akan digunakan untuk operasi pemeliharaan.	TERBESAR (DBInstanceClassMemory /63963136 * 1024.65536)
max_connections	Menetapkan jumlah maksimum koneksi serentak.	PALING SEDIKIT (InstanceClassMemoryDB/9531392, 5000)

Nama parameter	Deskripsi	Default
max_files_per_process	Menetapkan jumlah maksimum file yang terbuka secara bersamaan untuk setiap proses server.	–
max_locks_per_transaction	Menetapkan jumlah maksimum lock per transaksi.	64
max_logical_replication_workers	Jumlah maksimum proses pekerja replikasi logis.	–
max_parallel_maintenance_workers	Menetapkan jumlah maksimum proses paralel per operasi pemeliharaan.	–
max_parallel_workers	Menetapkan jumlah maksimum pekerja paralel yang dapat aktif pada satu waktu.	GREATEST($DBInstanceVCPU/2, 8)
max_parallel_workers_per_gather	Menetapkan jumlah maksimum proses paralel per simpul eksekutor.	–
max_pred_locks_per_page	Menetapkan jumlah maksimum tuple predicate-lock per halaman.	–
max_pred_locks_per_relation	Menetapkan jumlah maksimum halaman dan tuple predicate-lock per relasi.	–
max_pred_locks_per_transaction	Menetapkan jumlah maksimum predicate lock per transaksi.	–
max_prepared_transactions	Menetapkan jumlah maksimum transaksi yang disiapkan secara bersamaan.	0
max_replication_slots	Menetapkan jumlah maksimum slot replikasi yang dapat didukung oleh server.	20

Nama parameter	Deskripsi	Default
max_slot_wal_keep_size	(MB) Slot replikasi akan ditandai sebagai gagal, dan segmen akan dirilis untuk dihapus atau didaur ulang, jika ruang sebanyak ini ditempati oleh WAL pada disk.	–
max_stack_depth	(kB) Menetapkan kedalaman tumpukan maksimum, dalam kilobyte.	6144
max_standby_streaming_delay	(ms) Menetapkan penundaan maksimum sebelum membatalkan kueri saat hot standby sedang memproses data WAL yang dialirkan.	14000
max_sync_workers_per_subscription	Jumlah maksimum pekerja sinkronisasi per langganan	2
max_wal_senders	Menetapkan jumlah maksimum proses pengirim WAL yang berjalan secara bersamaan .	10
max_worker_processes	Menetapkan jumlah maksimum proses pekerja serentak.	GREATEST($DBInstanceVCPU*2, 8)
min_dynamic_shared_memory	(MB) Jumlah memori bersama dinamis yang dicadangkan saat pengaktifan.	–
min_parallel_index_scan_size	(8kB) Menetapkan jumlah minimum data indeks untuk pemindaian paralel.	–
min_parallel_table_scan_size	(8kB) Menetapkan jumlah minimum data tabel untuk pemindaian paralel.	–
old_snapshot_threshold	(min) Waktu sebelum snapshot terlalu lama untuk membaca halaman yang diubah setelah snapshot diambil.	–

Nama parameter	Deskripsi	Default
orafce.nls_date_format	Mengemulasi perilaku output tanggal oracle.	–
orafce.timezone	Menentukan zona waktu yang digunakan untuk fungsi sysdate.	–
parallel_leader_participation	Mengontrol apakah Gather and Gather Merge juga menjalankan subrencana.	–
parallel_setup_cost	Menetapkan perkiraan perencana terkait biaya untuk memulai proses pekerja untuk kueri paralel.	–
parallel_tuple_cost	Menetapkan perkiraan perencana untuk biaya pengiriman setiap tuple (baris) dari pekerja ke backend master.	–
password_encryption	Mengenkripsi kata sandi.	–
pgaudit.log	Menentukan kelas pernyataan mana yang akan dicatat oleh logging audit sesi.	–
pgaudit.log_catalog	Menentukan bahwa logging sesi harus diaktifkan jika semua relasi dalam pernyataan berada di pg_catalog.	–
pgaudit.log_level	Menentukan tingkat log yang akan digunakan untuk entri log.	–
pgaudit.log_parameter	Menentukan bahwa audit logging harus mencakup parameter yang diteruskan dengan pernyataan.	–
pgaudit.log_relation	Menentukan apakah logging audit sesi harus membuat entri log terpisah untuk setiap relasi (TABLE, VIEW, dll.) yang direferensikan dalam pernyataan SELECT atau DHTML.	–

Nama parameter	Deskripsi	Default
pgaudit.log_statement_once	Menentukan apakah logging akan mencakup teks pernyataan dan parameter dengan entri log pertama untuk kombinasi pernyataan/subpernyataan atau dengan setiap entri.	–
pgaudit.role	Menentukan peran master yang akan digunakan untuk logging audit objek.	–
pg_bigm.enable_recheck	Menentukan apakah akan melakukan Pemeriksaan Ulang yang merupakan proses internal pencarian teks lengkap.	on
pg_bigm.gin_key_limit	Menentukan jumlah maksimum 2 gram kata kunci pencarian yang akan digunakan untuk pencarian teks lengkap.	0
pg_bigm.last_update	Melaporkan tanggal pembaruan terakhir modul pg_bigm.	2013.11.22
pg_bigm.similarity_limit	Menentukan ambang batas minimum yang digunakan oleh pencarian kesamaan.	0,3
pg_hint_plan.debug_print	Mencatat log hasil dari penguraian petunjuk.	–
pg_hint_plan.enable_hint	Memaksa perencana untuk menggunakan rencana yang ditentukan dalam komentar petunjuk sebelum kueri.	–
pg_hint_plan.enable_hint_table	Memaksa perencana untuk tidak mendapatkan petunjuk dengan menggunakan pencarian tabel.	–
pg_hint_plan.message_level	Tingkat pesan untuk pesan debug.	–

Nama parameter	Deskripsi	Default
pg_hint_plan.parse_messages	Tingkat pesan kesalahan penguraian.	–
pglogical.batch_inserts	Melakukan penyisipan batch jika memungkinkan	–
pglogical.conflict_log_level	Menetapkan tingkat log yang digunakan untuk mencatat konflik yang diselesaikan.	–
pglogical.conflict_resolution	Menetapkan metode yang digunakan untuk penyelesaian konflik untuk konflik yang dapat diselesaikan.	–
pglogical.extra_connection_options	Opsi koneksi untuk ditambahkan ke semua koneksi simpul peer	–
pglogical.synchronous_commit	Nilai komit sinkron spesifik pglogical	–
pglogical.use_spi	Menggunakan SPI alih-alih API tingkat rendah untuk menerapkan perubahan	–
pg_prewarm.autoprewarm	Memulai pekerja autoprewarm.	–
pg_prewarm.autoprewarm_interval	Menetapkan interval di antara dump buffer bersama	–
pg_similarity.block_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.block_threshold	Menetapkan ambang batas yang digunakan oleh fungsi kesamaan Blok.	–
pg_similarity.block_tokenizer	Menetapkan tokenizer untuk fungsi kesamaan Blok.	–

Nama parameter	Deskripsi	Default
pg_similarity.cosine_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.cosine_threshold	Menetapkan ambang batas yang digunakan oleh fungsi kesamaan Kosinus.	–
pg_similarity.cosine_tokenizer	Menetapkan pentokenisasi untuk fungsi kesamaan Kosinus.	–
pg_similarity.dice_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.dice_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Dice.	–
pg_similarity.dice_tokenizer	Menetapkan pentokenisasi untuk ukuran kesamaan Dice.	–
pg_similarity.euclidean_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.euclidean_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Euclidean.	–
pg_similarity.euclidean_tokenizer	Menetapkan pentokenisasi untuk ukuran kesamaan Euclidean.	–
pg_similarity.hamming_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.hamming_threshold	Menetapkan ambang batas yang digunakan oleh metrik kesamaan Blok.	–
pg_similarity.jaccard_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.jaccard_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Jaccard.	–

Nama parameter	Deskripsi	Default
pg_similarity.jaccard_tokenizer	Menetapkan pentokenisasi untuk ukuran kesamaan Jaccard.	–
pg_similarity.jaro_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.jaro_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Jaro.	–
pg_similarity.jarowinkler_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.jarowinkler_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Jarowinkler.	–
pg_similarity.levenshtein_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.levenshtein_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Levenshtein.	–
pg_similarity.matching_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.matching_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Koefisien Pencocokan.	–
pg_similarity.matching_tokenizer	Menetapkan pentokenisasi untuk ukuran kesamaan Koefisien Pencocokan.	–
pg_similarity.mongeelkan_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.mongeelkan_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Monge-Elkan.	–
pg_similarity.mongeelkan_tokenizer	Menetapkan pentokenisasi untuk ukuran kesamaan Monge-Elkan.	–

Nama parameter	Deskripsi	Default
pg_similarity.nw_gap_penalty	Menetapkan penalti kesenjangan yang digunakan oleh ukuran kesamaan Needleman-Wunsch.	–
pg_similarity.nw_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.nw_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Needleman-Wunsch.	–
pg_similarity.overlap_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.overlap_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Koefisien Tumpang-Tindih.	–
pg_similarity.overlap_tokenizer	Menetapkan pentokenisasi untuk ukuran kesamaan Koefisien Tumpang-Tindih.	–
pg_similarity.qgram_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.qgram_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Q-Gram.	–
pg_similarity.qgram_tokenizer	Menetapkan pentokenisasi untuk ukuran Q-Gram.	–
pg_similarity.swg_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.swg_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Smith-Waterman-Gotoh.	–

Nama parameter	Deskripsi	Default
pg_similarity.sw_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.sw_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Smith-Waterman.	–
pg_stat_statements.max	Menetapkan jumlah maksimum pernyataan yang dilacak oleh pg_stat_statements.	–
pg_stat_statements.save	Save pg_stat_statements statistics across server shutdowns.	–
pg_stat_statements.track	Memilih pernyataan mana yang dilacak oleh pg_stat_statements.	–
pg_stat_statements.track_planning	Memilih apakah durasi perencanaan dilacak oleh pg_stat_statements.	–
pg_stat_statements.track_utility	Memilih apakah perintah utilitas dilacak oleh pg_stat_statements.	–
plan_cache_mode	Mengontrol pemilihan perencana untuk rencana kustom atau generik.	–
port	Menetapkan port TCP yang didengarkan server.	EndPointPort
postgis.gdal_enabled_drivers	Mengaktifkan atau menonaktifkan driver GDAL yang digunakan dengan PostGIS di Postgres 9.3.5 dan versi lebih tinggi.	ENABLE_ALL
quote_all_identifiers	Saat membuat fragmen SQL, mengutip semua pengidentifikasi.	–
random_page_cost	Menetapkan perkiraan perencana untuk biaya halaman disk yang diambil secara tidak berurutan.	–

Nama parameter	Deskripsi	Default
rdkit.dice_threshold	Ambang batas bawah kesamaan Dice. Molekul dengan kesamaan lebih rendah dari ambang batas tidak sama berdasarkan operasi #.	–
rdkit.do_chiral_sss	Apakah stereokimia harus diperhitungkan dalam pencocokan substruktur. Jika salah, tidak ada informasi stereokimia yang digunakan dalam kecocokan substruktur.	–
rdkit.tanimoto_threshold	Ambang batas bawah kesamaan Tanimoto. Molekul dengan kesamaan lebih rendah dari ambang batas tidak sama berdasarkan operasi %.	–
rds.accepted_password_auth_method	Memaksa autentikasi untuk koneksi dengan kata sandi yang disimpan secara lokal.	md5+scram
rds.adaptive_autovacuum	Parameter RDS untuk mengaktifkan/menonaktifkan autovacuum adaptif.	1
rds.babelfish_status	Parameter RDS untuk mengaktifkan/menonaktifkan Babelfish for Aurora PostgreSQL.	off
rds.enable_plan_management	Mengaktifkan atau menonaktifkan ekstensi apg_plan_mgmt.	0

Nama parameter	Deskripsi	Default
rds.extensions	Daftar ekstensi yang disediakan oleh RDS	address_standardizer, address_standardizer_data_us, apg_plan_mgmt, aurora_stat_utils, amcheck, autoinc, aws_commons, aws_ml, aws_s3, aws_lambda, bool_plperl, bloom, btree_gin, btree_gist, citext, cube, dblink, dict_int, dict_xsyn, earthdistance, fuzzystrmatch, hll, hstore, hstore_plperl, insert_username, intagg, intarray, ip4r, isn, jsonb_plperl, lo, log_fdw, ltree, moddatetime, old_snapshot, oracle_fdw, orafce, pgaudit, pgcrypto, pglogical, pgrouting, pgrowlocks, pgstattuple, pgtap, pg_bigm, pg_buffercache, pg_cron, pg_freespacemap, pg_hint_plan, pg_partman, pg_prewarm, pg_proctab, pg_repack, pg_simila

Nama parameter	Deskripsi	Default
		rity, pg_stat_s tatements, pg_trgm, pg_visibility, plcoffee, plls, plperl, plpgsql, plprofiler, pltcl, plv8, postgis, postgis_t iger_geocoder, postgis_raster, postgis_topology, postgres_fdw, prefix, rdkit, rds_tools, refint, sslinfo, tablefunc, tds_fdw, test_parser, tsm_system_rows, tsm_system_time, unaccent, uuid-ossp
rds.force_admin_lo gging_level	Melihat pesan log untuk tindakan pengguna admin RDS di basis data pelanggan.	–
rds.force_autovacu um_logging_level	Melihat pesan log yang terkait dengan operasi autovacuum.	WARNING
rds.force_ssl	Memaksa koneksi SSL.	0

Nama parameter	Deskripsi	Default
rds.global_db_rpo	(s) Ambang batas sasaran titik pemulihan, dalam hitungan detik, yang memblokir komit pengguna jika ambang batas ini dilanggar. **Important** Parameter ini dimaksudkan untuk basis data global berbasis Aurora PostgreSQL. Untuk basis data nonglobal, biarkan pada nilai default. Untuk informasi selengkapnya tentang penggunaan parameter ini, lihat the section called “Mengelola RPO untuk basis data global berbasis Aurora PostgreSQL”.	–
rds.logical_replication	Mengaktifkan pendekodean logis.	0
rds.logically_replicate_unlogged_tables	Tabel yang tidak dicatat direplikasi secara logis.	1
rds.log_retention_period	Amazon RDS akan menghapus log PostgreSQL yang lebih lama dari N menit.	4320
rds.pg_stat_ramdisk_size	Ukuran statistik ramdisk dalam MB. Nilai bukan nol akan mengatur ramdisk. Parameter ini hanya tersedia di Aurora PostgreSQL 14 dan versi lebih rendah.	0
rds.rds_superuser_reserved_connections	Menetapkan jumlah slot koneksi yang disediakan untuk rds_superusers.	2
rds.restrict_password_commands	Membatasi perintah terkait kata sandi untuk anggota rds_password	–
rds.superuser_variables	Daftar variabel superuser-only yang pernyataan modifikasi rds_superuser-nya dielevasikan.	session_replication_role

Nama parameter	Deskripsi	Default
recovery_init_sync_method	Menetapkan metode untuk menyinkronkan direktori data sebelum pemulihan crash.	sinkronisasi
remove_temp_files_after_crash	Menghapus file sementara setelah backend crash.	0
restart_after_crash	Menginisialisasi ulang server setelah backend crash.	–
row_security	Mengaktifkan keamanan baris.	–
search_path	Menetapkan urutan pencarian skema untuk nama yang tidak memenuhi syarat skema.	–
seq_page_cost	Menetapkan perkiraan perencana untuk biaya halaman disk yang diambil secara berurutan.	–
session_replication_role	Menetapkan perilaku sesi untuk pemicu dan aturan penulisan ulang.	–
shared_buffers	(8kB) Menetapkan jumlah buffer memori bersama yang digunakan oleh server.	JUMLAH (InstanceClassMemoryDB/12038, -50003)
shared_preload_libraries	Menampilkan daftar pustaka bersama untuk dimuat sebelumnya ke server.	pg_stat_statements
ssl	Mengaktifkan koneksi SSL.	1
ssl_ca_file	Lokasi file otoritas server SSL.	/rdsdbdata/rds-metadata/ca-cert.pem
ssl_cert_file	Lokasi file sertifikat server SSL.	/rdsdbdata/rds-metadata/server-cert.pem
ssl_ciphers	Menetapkan daftar cipher TLS yang diizinkan untuk digunakan pada koneksi aman.	–

Nama parameter	Deskripsi	Default
ssl_crl_dir	Lokasi direktori daftar pencabutan sertifikat SSL.	/rdsdbdata/rds-metadata/ssl_crl_dir/
ssl_key_file	Lokasi file kunci privat server SSL	/rdsdbdata/rds-metadata/server-key.pem
ssl_max_protocol_version	Menetapkan versi protokol SSL/TLS maksimum yang diizinkan	–
ssl_min_protocol_version	Menetapkan versi protokol SSL/TLS minimum yang diizinkan	TLSv1.2
standard_conforming_strings	Menyebabkan string ... memperlakukan garis miring secara literal.	–
statement_timeout	(ms) Menetapkan durasi maksimum yang diizinkan untuk setiap pernyataan.	–
stats_temp_directory	Menulis file statistik sementara ke direktori yang ditentukan.	/rdsdbdata/db/pg_stat_tmp
superuser_reserved_connections	Menetapkan jumlah slot koneksi yang disediakan untuk pengguna super.	3
synchronize_seqscans	Mengaktifkan pemindaian berurutan yang disinkronkan.	–
synchronous_commit	Menetapkan tingkat sinkronisasi transaksi saat ini.	on
tcp_keepalives_count	Jumlah maksimum pengiriman ulang keepalive TCP.	–
tcp_keepalives_idle	(s) Waktu di antara penerbitan keepalive TCP.	–
tcp_keepalives_interval	(s) Waktu di antara pengiriman ulang keepalive TCP.	–

Nama parameter	Deskripsi	Default
temp_buffers	(8kB) Menetapkan jumlah maksimum buffer sementara yang digunakan oleh setiap sesi.	–
temp_file_limit	Membatasi jumlah total ruang disk dalam kilobyte yang dapat digunakan oleh proses PostgreSQL tertentu untuk file sementara, tidak termasuk ruang yang digunakan untuk tabel sementara eksplisit	-1
temp_tablespaces	Menetapkan ruang tabel yang akan digunakan di tabel sementara dan sort file.	–
timezone	Menetapkan zona waktu untuk menampilkan dan menginterpretasikan stempel waktu.	UTC
track_activities	Mengumpulkan informasi tentang eksekusi perintah.	–
track_activity_query_size	Menetapkan ukuran yang disediakan untuk pg_stat_activity.current_query, dalam byte.	4096
track_commit_timestamp	Mengumpulkan waktu komit transaksi.	–
track_counts	Mengumpulkan statistik tentang aktivitas basis data.	–
track_functions	Mengumpulkan statistik tingkat fungsi tentang aktivitas basis data.	pl
track_io_timing	Mengumpulkan statistik waktu tentang aktivitas IO basis data.	1
track_wal_io_timing	Mengumpulkan statistik waktu untuk aktivitas I/O WAL.	–

Nama parameter	Deskripsi	Default
transform_null_equals	Memperlakukan expr=NULL sebagai expr IS NULL.	–
update_process_title	Memperbarui judul proses untuk menampilkan perintah SQL aktif.	–
vacuum_cost_delay	(ms) Penundaan biaya vacuum dalam milidetik.	–
vacuum_cost_limit	Jumlah biaya vacuum yang tersedia sebelum napping.	–
vacuum_cost_page_dirty	Biaya vacuum untuk halaman yang kotor oleh vacuum.	–
vacuum_cost_page_hit	Biaya vacuum untuk halaman yang ditemukan di cache buffer.	–
vacuum_cost_page_miss	Biaya vacuum untuk halaman yang tidak ditemukan dalam cache buffer.	0
vacuum_defer_cleanup_age	Jumlah transaksi yang harus ditangguhkan dengan pembersihan VACUUM dan HOT, jika ada.	–
vacuum_failsafe_age	Usia saat VACUUM harus memicu failsafe untuk menghindari gangguan wraparound.	1200000000
vacuum_freeze_min_age	Usia minimum saat VACUUM harus membekukan baris tabel.	–
vacuum_freeze_table_age	Usia saat VACUUM harus memindai seluruh tabel untuk membekukan tuple.	–
vacuum_multixact_failsafe_age	Usia multixact saat VACUUM harus memicu failsafe untuk menghindari gangguan wraparound.	1200000000

Nama parameter	Deskripsi	Default
vacuum_multixact_freeze_min_age	Usia minimum di mana VACUUM harus membekukan a MultiXactId dalam baris tabel.	–
vacuum_multixact_freeze_table_age	Usia multixact saat VACUUM harus memindai seluruh tabel untuk membekukan tuple.	–
wal_buffers	(8kB) Menetapkan jumlah buffer halaman disk dalam memori bersama untuk WAL.	–
wal_receiver_create_temp_slot	Menetapkan apakah penerima WAL harus membuat slot replikasi sementara jika tidak ada slot permanen yang dikonfigurasi.	0
wal_receiver_status_interval	(s) Menetapkan interval maksimum di antara laporan status penerima WAL ke primer.	–
wal_receiver_timeout	(ms) Menetapkan waktu tunggu maksimum untuk menerima data dari primer.	30000
wal_sender_timeout	(ms) Menetapkan waktu maksimum untuk menunggu replikasi WAL.	–
work_mem	(kB) Menetapkan memori maksimum yang akan digunakan untuk workspace kueri.	–
xmlbinary	Menetapkan cara nilai biner akan dienkode dalam XML.	–
xmloption	Menetapkan apakah data XML dalam operasi penguraian implisit dan serialisasi dianggap sebagai dokumen atau fragmen konten.	–

Parameter tingkat instans Aurora PostgreSQL

Anda dapat melihat parameter tingkat instans yang tersedia untuk versi PostgreSQL Aurora tertentu menggunakan konsol AWS Manajemen, CLI, atau Amazon RDS API. AWS Untuk informasi tentang cara melihat parameter dalam grup parameter DB Aurora PostgreSQL di konsol RDS, lihat Melihat nilai parameter untuk grup parameter DB.

Beberapa parameter tingkat instans tidak tersedia di semua versi dan beberapa sedang ditiadakan. Untuk informasi tentang cara melihat parameter versi PostgreSQL Aurora tertentu, lihat Melihat klaster DB dan parameter DB Aurora PostgreSQL.

Misalnya, tabel berikut mencantumkan parameter yang berlaku untuk instans DB tertentu di klaster DB Aurora PostgreSQL. Daftar ini dihasilkan dengan menjalankan describe-db-parameters AWS CLI perintah dengan `default.aurora-postgresql14` `--db-parameter-group-name` nilai.

Untuk daftar parameter klaster DB untuk grup parameter DB default yang sama ini, lihatParameter tingkat klaster Aurora PostgreSQL.

Nama parameter	Deskripsi	Default
apg_enable_batch_mode_function_execution	Mengaktifkan fungsi mode batch untuk memproses set baris pada suatu waktu.	–
apg_enable_correlated_any_transform	Memungkinkan perencana mentransformasikan Subtautan ANY terkorelasi (subkueri IN/NOT IN) menjadi JOIN jika memungkinkan.	–
apg_enable_function_migration	Memungkinkan perencana memigrasikan fungsi skalar yang memenuhi syarat ke klausa FROM.	–
apg_enable_not_in_transform	Memungkinkan perencana mentransformasikan subkueri NOT IN menjadi ANTI JOIN jika memungkinkan.	–
apg_enable_remove_redundant_inner_joins	Memungkinkan perencana menghapus inner join yang redundan.	–

Nama parameter	Deskripsi	Default
apg_enable_semijoin_push_down	Memungkinkan penggunaan filter semijoin untuk hash join.	–
apg_plan_mgmt.capture_plan_baselines	Mengambil mode acuan dasar rencana. manual - mengaktifkan pengambilan rencana untuk pernyataan SQL apa pun, off - menonaktifkan pengambilan rencana, automatic - mengaktifkan pengambilan rencana untuk pernyataan dalam pg_stat_statement yang memenuhi kriteria kelayakan.	off
apg_plan_mgmt.max_databases	Menetapkan jumlah maksimum basis data yang dapat mengelola kueri menggunakan apg_plan_mgmt.	10
apg_plan_mgmt.max_plans	Menetapkan jumlah maksimum rencana yang dapat di-cache oleh apg_plan_mgmt.	10000
apg_plan_mgmt.plan_retention_period	Jumlah hari maksimum sejak rencana last_used sebelum rencana akan dihapus secara otomatis.	32
apg_plan_mgmt.unapproved_plan_execution_threshold	Perkiraan total biaya rencana yang jika tidak tercapai akan membuat rencana yang Tidak Disetujui dieksekusi.	0
apg_plan_mgmt.use_plan_baselines	Menggunakan hanya rencana yang disetujui atau tetap untuk pernyataan terkelola.	false
application_name	Menetapkan nama aplikasi yang akan dilaporkan dalam statistik dan log.	–

Nama parameter	Deskripsi	Default
aurora_compute_plan_id	Memantau rencana eksekusi kueri untuk mendeteksi rencana eksekusi yang berkontribusi pada pemuatan basis data saat ini dan untuk melacak statistik kinerja rencana eksekusi dari waktu ke waktu. Untuk informasi selengkapnya, lihat Memantau rencana eksekusi kueri untuk Aurora PostgreSQL.	on
authentication_timeout	(s) Menetapkan waktu maksimum yang diizinkan untuk menyelesaikan autentikasi klien.	–
auto_explain.log_analysis	Menggunakan EXPLORE ANALYSIS untuk logging rencana.	–
auto_explain.log_buffers	Mencatat log penggunaan buffer.	–
auto_explain.log_format	Menggunakan EXPLAIN untuk format yang akan digunakan untuk logging rencana.	–
auto_explain.log_min_duration	Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat rencana dicatat.	–
auto_explain.log_nested_statement	Mencatat log pernyataan bersarang.	–
auto_explain.log_timing	Mengumpulkan data waktu, bukan hanya hitungan baris.	–
auto_explain.log_trigger	Menyertakan statistik pemicu dalam rencana.	–
auto_explain.log_verbose	Menggunakan EXPLORE VERBOSE untuk logging rencana.	–

Nama parameter	Deskripsi	Default
auto_explain.sampl e_rate	Pecahan kueri untuk diproses.	–
babelfishpg_tds.li sten_addresses	Menetapkan nama host atau alamat IP untuk mendengarkan TDS.	*
babelfishpg_tds.td s_debug_log_level	Menetapkan tingkat logging di TDS, 0 menonaktifkan logging	1
backend_flush_after	(8Kb) Jumlah halaman yang jika terpenuhi akan membuat penulisan yang dilakukan sebelumny a dibuang ke disk.	–
bytea_output	Menetapkan format output untuk bytea.	–
check_function_bod ies	Memeriksa isi fungsi selama CREATE FUNCTION.	–
client_connection_ check_interval	Menetapkan interval waktu di antara pemeriksa an pemutusan koneksi saat menjalankan kueri.	–
client_min_messages	Menetapkan tingkat pesan yang dikirimkan ke klien.	–
config_file	Menetapkan file konfigurasi utama server.	/rdsdbdata/config/ postgresql.conf
constraint_exclusion	Memungkinkan perencana menggunakan pembatasan untuk mengoptimalkan kueri.	–
cpu_index_tuple_cost	Menetapkan perkiraan perencana untuk biaya pemrosesan setiap entri indeks selama pemindaian indeks.	–
cpu_operator_cost	Menetapkan perkiraan perencana untuk biaya pemrosesan setiap panggilan operator atau fungsi.	–

Nama parameter	Deskripsi	Default
cpu_tuple_cost	Menetapkan perkiraan perencana untuk biaya pemrosesan setiap tuple (baris).	–
cron.database_name	Menetapkan basis data untuk menyimpan tabel metadata pg_cron	postgres
cron.log_run	Mencatat log semua pelaksanaan pekerjaan ke tabel job_run_details	on
cron.log_statement	Mencatat log semua pernyataan cron sebelum eksekusi.	off
cron.max_running_jobs	Jumlah maksimum pekerjaan yang dapat dijalankan secara bersamaan.	5
cron.use_background_workers	Mengaktifkan pekerja latar belakang untuk pg_cron	on
cursor_tuple_fraction	Menetapkan perkiraan perencana untuk pecahan dari baris kursor yang akan diambil.	–
db_user_namespace	Mengaktifkan nama pengguna per basis data.	–
deadlock_timeout	(ms) Menetapkan waktu untuk menunggu lock sebelum memeriksa deadlock.	–
debug_pretty_print	Mengindentasi tampilan penguraian dan hierarki rencana.	–
debug_print_parse	Mencatat log hierarki penguraian setiap kueri.	–
debug_print_plan	Mencatat log rencana eksekusi setiap kueri.	–
debug_print_rewritten	Mencatat log hierarki penguraian yang ditulis ulang masing-masing kueri.	–
default_statistics_target	Menetapkan target statistik default.	–

Nama parameter	Deskripsi	Default
default_transactio n_deferrable	Menetapkan status default yang dapat ditangguhkan untuk transaksi baru.	–
default_transactio n_isolation	Menetapkan tingkat isolasi transaksi untuk setiap transaksi baru.	–
default_transactio n_read_only	Menetapkan status hanya baca default untuk transaksi baru.	–
effective_cache_size	(8kB) Menetapkan asumsi perencana tentang ukuran cache disk.	JUMLAH (Instance ClassMemo ryDB/12038, -50003
effective_io_concu rrency	Jumlah permintaan serentak yang dapat ditangani secara efisien oleh subsistem disk.	–
enable_async_append	Memungkinkan perencana menggunakan rencana async append.	–
enable_bitmapscan	Memungkinkan perencana menggunakan rencana bitmap-scan.	–
enable_gathermerge	Memungkinkan perencana menggunakan rencana gather merge.	–
enable_hashagg	Memungkinkan perencana menggunakan rencana hashed aggregation.	–
enable_hashjoin	Memungkinkan perencana menggunakan rencana hash join.	–
enable_incremental _sort	Memungkinkan perencana menggunakan langkah-langkah pengurutan inkremental.	–
enable_indexonlyscan	Memungkinkan perencana menggunakan index-only-scan rencana.	–

Nama parameter	Deskripsi	Default
enable_indexscan	Memungkinkan perencana menggunakan rencana index-scan.	–
enable_material	Memungkinkan perencana menggunakan materialization.	–
aktifkan_memoize	Memungkinkan perencana menggunakan memoization	–
enable_mergejoin	Memungkinkan perencana menggunakan rencana merge join.	–
enable_nestloop	Memungkinkan perencana menggunakan rencana nested-loop join.	–
enable_parallel_append	Mengaktifkan penggunaan perencana atas rencana parallel append.	–
aktifkan_parallel_hash	Mengaktifkan penggunaan perencana atas rencana parallel hash.	–
enable_partition_pruning	Mengaktifkan pemangkasan partisi plan-time dan run-time.	–
enable_partitionwise_aggregate	Mengaktifkan agregasi dan pengelompokan partitionwise.	–
enable_partitionwise_join	Mengaktifkan partitionwise join.	–
enable_seqscan	Memungkinkan perencana menggunakan rencana sequential-scan.	–
enable_sort	Memungkinkan perencana menggunakan langkah-langkah penyortiran eksplisit.	–
enable_tidscan	Memungkinkan perencana menggunakan rencana TID scan.	–

Nama parameter	Deskripsi	Default
escape_string_warning	Memperingatkan tentang escape garis miring dalam string literal biasa.	–
exit_on_error	Mengakhiri sesi saat terjadi kesalahan apa pun.	–
force_parallel_mode	Memaksa penggunaan fasilitas kueri paralel.	–
from_collapse_limit	Menetapkan ukuran FROM-list yang jika terlampaui akan membuat subkueri tidak ditutup.	–
geqo	Memungkinkan optimisasi kueri genetik.	–
geqo_effort	GEQO: upaya digunakan untuk mengatur nilai default untuk parameter GEQO lainnya.	–
geqo_generations	GEQO: jumlah iterasi algoritma.	–
geqo_pool_size	GEQO: jumlah individu dalam populasi.	–
geqo_seed	GEQO: seed untuk pemilihan jalur acak.	–
geqo_selection_bias	GEQO: tekanan selektif di dalam populasi.	–
geqo_threshold	Menetapkan ambang batas item FROM yang jika terlampaui akan membuat GEQO digunakan.	–
gin_fuzzy_search_limit	Menetapkan hasil maksimum yang diizinkan untuk pencarian persis oleh GIN.	–
gin_pending_list_limit	(kB) Menetapkan ukuran maksimum daftar tertunda untuk indeks GIN.	–
hash_mem_multiplier	Beberapa work_mem yang akan digunakan untuk tabel hash.	–

Nama parameter	Deskripsi	Default
hba_file	Menetapkan file konfigurasi hba server.	/rdsdbdata/config/pg_hba.conf
hot_standby_feedback	Memungkinkan umpan balik dari hot standby ke primer yang akan menghindari konflik kueri.	on
ident_file	Menetapkan file konfigurasi ident server.	/rdsdbdata/config/pg_ident.conf
idle_in_transaction_session_timeout	(ms) Menetapkan durasi maksimum yang diizinkan untuk setiap transaksi idling.	86400000
idle_session_timeout	Mengakhiri setiap sesi yang telah idle (yaitu, menunggu kueri klien), tetapi tidak dalam transaksi terbuka, lebih lama dari jumlah waktu yang ditentukan	–
join_collapse_limit	Menetapkan ukuran FORM-list yang jika terlampaui akan membuat konstruk JOIN tidak diratakan.	–
lc_messages	Menetapkan dalam bahasa apa pesan akan ditampilkan.	–
listen_addresses	Menetapkan nama host atau alamat IP untuk didengarkan.	*
lo_compat_privileges	Mengaktifkan mode kompatibilitas mundur untuk pemeriksaan hak akses pada objek besar.	0
log_connections	Mencatat log setiap koneksi yang berhasil.	–
log_destination	Menetapkan tujuan untuk output log server.	stderr
log_directory	Menetapkan direktori tujuan untuk file log.	/rdsdbdata/log/error

Nama parameter	Deskripsi	Default
log_disconnections	Mencatat log akhir sebuah sesi, termasuk durasinya.	–
log_duration	Mencatat log durasi setiap pernyataan SQL yang diselesaikan.	–
log_error_verbosity	Menetapkan verbositas pesan yang dicatat.	–
log_executor_stats	Menulis statistik performa pengeksekusi ke log server.	–
log_file_mode	Menetapkan izin file untuk file log.	0644
log_filename	Menetapkan pola nama file untuk file log.	postgresql.log.%Y-%m-%d-%H%M
logging_collector	Mulai subproses untuk menangkap output stderr dan/atau csvlog ke dalam file log.	1
log_hostname	Mencatat log nama host dalam log koneksi.	0
logical_decoding_work_mem	(kB) Memori sebanyak ini dapat digunakan oleh setiap buffer penyusun ulang internal sebelum dialirkan ke disk.	–
log_line_prefix	Mengontrol informasi yang ditambahkan ke awal setiap baris log.	%t:%r:%u@%d:%p]:
log_lock_waits	Mencatat log waktu tunggu lock yang panjang.	–
log_min_duration_sample	(ms) Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat sampel pernyataan dicatat. Pengambilan sampel ditentukan oleh log_statement_sample_rate.	–
log_min_duration_statement	(ms) Menetapkan waktu eksekusi minimum yang jika terlampaui akan membuat pernyataan dicatat.	–

Nama parameter	Deskripsi	Default
log_min_error_statement	Menyebabkan semua pernyataan yang menghasilkan kesalahan pada atau di atas tingkat ini dicatat.	–
log_min_messages	Menetapkan tingkat pesan yang dicatat.	–
log_parameter_max_length	(B) Saat mencatat pernyataan, membatasi nilai parameter yang dicatat ke N byte pertama.	–
log_parameter_max_length_on_error	(B) Saat melaporkan kesalahan, membatasi nilai parameter yang dicatat ke N byte pertama.	–
log_parser_stats	Menulis statistik performa pengurai ke log server.	–
log_planner_stats	Menulis statistik performa perencana ke log server.	–
log_replication_commands	Mencatat log setiap perintah replikasi.	–
log_rotation_age	(min) Rotasi file log otomatis akan terjadi setelah N menit.	60
log_rotation_size	(kB) Rotasi file log otomatis akan terjadi setelah N kilobyte.	100000
log_statement	Menetapkan jenis pernyataan yang dicatat.	–
log_statement_sample_rate	Pecahan dari pernyataan yang melebihi log_min_duration_sample yang akan dicatat.	–
log_statement_stats	Menulis statistik performa kumulatif ke log server.	–
log_temp_files	(kB) Mencatat log penggunaan file sementara yang lebih besar dari jumlah kilobyte ini.	–

Nama parameter	Deskripsi	Default
log_timezone	Menetapkan zona waktu yang akan digunakan dalam pesan log.	UTC
log_truncate_on_rotation	Memotong file log yang ada dengan nama yang sama selama rotasi log.	0
maintenance_io_concurrency	Varian effective_io_concurrency yang digunakan untuk pekerjaan pemeliharaan.	1
maintenance_work_mem	(kB) Menetapkan memori maksimum yang akan digunakan untuk operasi pemeliharaan.	TERBESAR (DBInstanceClassMemory /63963136 *1024.65536
max_connections	Menetapkan jumlah maksimum koneksi serentak.	PALING SEDIKIT (InstanceClassMemoryDB/9531392, 5000
max_files_per_process	Menetapkan jumlah maksimum file yang terbuka secara bersamaan untuk setiap proses server.	–
max_locks_per_transaction	Menetapkan jumlah maksimum lock per transaksi.	64
max_parallel_maintenance_workers	Menetapkan jumlah maksimum proses paralel per operasi pemeliharaan.	–
max_parallel_workers	Menetapkan jumlah maksimum pekerja paralel yang dapat aktif pada satu waktu.	GREATEST($DBInstanceVCPU/2, 8
max_parallel_workers_per_gather	Menetapkan jumlah maksimum proses paralel per simpul eksekutor.	–
max_pred_locks_per_page	Menetapkan jumlah maksimum tuple predicate-lock per halaman.	–

Nama parameter	Deskripsi	Default
max_pred_locks_per_relation	Menetapkan jumlah maksimum halaman dan tuple predicate-lock per relasi.	–
max_pred_locks_per_transaction	Menetapkan jumlah maksimum predicate lock per transaksi.	–
max_slot_wal_keep_size	(MB) Slot replikasi akan ditandai sebagai gagal, dan segmen akan dirilis untuk dihapus atau didaur ulang, jika ruang sebanyak ini ditempati oleh WAL pada disk.	–
max_stack_depth	(kB) Menetapkan kedalaman tumpukan maksimum, dalam kilobyte.	6144
max_standby_streaming_delay	(ms) Menetapkan penundaan maksimum sebelum membatalkan kueri saat hot standby sedang memproses data WAL yang dialirkan.	14000
max_worker_processes	Menetapkan jumlah maksimum proses pekerja serentak.	GREATEST($DBInstanceVCPU*2, 8
min_dynamic_shared_memory	(MB) Jumlah memori bersama dinamis yang dicadangkan saat pengaktifan.	–
min_parallel_index_scan_size	(8kB) Menetapkan jumlah minimum data indeks untuk pemindaian paralel.	–
min_parallel_table_scan_size	(8kB) Menetapkan jumlah minimum data tabel untuk pemindaian paralel.	–
old_snapshot_threshold	(min) Waktu sebelum snapshot terlalu lama untuk membaca halaman yang diubah setelah snapshot diambil.	–
parallel_leader_participation	Mengontrol apakah Gather and Gather Merge juga menjalankan subrencana.	–

Nama parameter	Deskripsi	Default
parallel_setup_cost	Menetapkan perkiraan perencana terkait biaya untuk memulai proses pekerja untuk kueri paralel.	–
parallel_tuple_cost	Menetapkan perkiraan perencana untuk biaya pengiriman setiap tuple (baris) dari pekerja ke backend master.	–
pgaudit.log	Menentukan kelas pernyataan mana yang akan dicatat oleh logging audit sesi.	–
pgaudit.log_catalog	Menentukan bahwa logging sesi harus diaktifkan jika semua relasi dalam pernyataan berada di pg_catalog.	–
pgaudit.log_level	Menentukan tingkat log yang akan digunakan untuk entri log.	–
pgaudit.log_parameter	Menentukan bahwa audit logging harus mencakup parameter yang diteruskan dengan pernyataan.	–
pgaudit.log_relation	Menentukan apakah logging audit sesi harus membuat entri log terpisah untuk setiap relasi (TABLE, VIEW, dll.) yang direferensikan dalam pernyataan SELECT atau DHTML.	–
pgaudit.log_statement_once	Menentukan apakah logging akan mencakup teks pernyataan dan parameter dengan entri log pertama untuk kombinasi pernyataan/subpernyataan atau dengan setiap entri.	–
pgaudit.role	Menentukan peran master yang akan digunakan untuk logging audit objek.	–

Nama parameter	Deskripsi	Default
pg_bigm.enable_recheck	Menentukan apakah akan melakukan Pemeriksaan Ulang yang merupakan proses internal pencarian teks lengkap.	on
pg_bigm.gin_key_limit	Menentukan jumlah maksimum 2 gram kata kunci pencarian yang akan digunakan untuk pencarian teks lengkap.	0
pg_bigm.last_update	Melaporkan tanggal pembaruan terakhir modul pg_bigm.	2013.11.22
pg_bigm.similarity_limit	Menentukan ambang batas minimum yang digunakan oleh pencarian kesamaan.	0,3
pg_hint_plan.debug_print	Mencatat log hasil dari penguraian petunjuk.	–
pg_hint_plan.enable_hint	Memaksa perencana untuk menggunakan rencana yang ditentukan dalam komentar petunjuk sebelum kueri.	–
pg_hint_plan.enable_hint_table	Memaksa perencana untuk tidak mendapatkan petunjuk dengan menggunakan pencarian tabel.	–
pg_hint_plan.message_level	Tingkat pesan untuk pesan debug.	–
pg_hint_plan.parse_messages	Tingkat pesan kesalahan penguraian.	–
pglogical.batch_inserts	Melakukan penyisipan batch jika memungkinkan	–
pglogical.conflict_log_level	Menetapkan tingkat log yang digunakan untuk mencatat konflik yang diselesaikan.	–

Nama parameter	Deskripsi	Default
pglogical.conflict_resolution	Menetapkan metode yang digunakan untuk penyelesaian konflik untuk konflik yang dapat diselesaikan.	–
pglogical.extra_connection_options	Opsi koneksi untuk ditambahkan ke semua koneksi simpul peer	–
pglogical.synchronous_commit	Nilai komit sinkron spesifik pglogical	–
pglogical.use_spi	Menggunakan SPI alih-alih API tingkat rendah untuk menerapkan perubahan	–
pg_similarity.block_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.block_threshold	Menetapkan ambang batas yang digunakan oleh fungsi kesamaan Blok.	–
pg_similarity.block_tokenizer	Menetapkan tokenizer untuk fungsi kesamaan Blok.	–
pg_similarity.cosine_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.cosine_threshold	Menetapkan ambang batas yang digunakan oleh fungsi kesamaan Kosinus.	–
pg_similarity.cosine_tokenizer	Menetapkan pentokenisasi untuk fungsi kesamaan Kosinus.	–
pg_similarity.dice_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.dice_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Dice.	–

Nama parameter	Deskripsi	Default
pg_similarity.dice_tokenizer	Menetapkan pentokenisasi untuk ukuran kesamaan Dice.	–
pg_similarity.euclidean_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.euclidean_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Euclidean.	–
pg_similarity.euclidean_tokenizer	Menetapkan pentokenisasi untuk ukuran kesamaan Euclidean.	–
pg_similarity.hamming_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.hamming_threshold	Menetapkan ambang batas yang digunakan oleh metrik kesamaan Blok.	–
pg_similarity.jaccard_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.jaccard_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Jaccard.	–
pg_similarity.jaccard_tokenizer	Menetapkan pentokenisasi untuk ukuran kesamaan Jaccard.	–
pg_similarity.jaro_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.jaro_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Jaro.	–
pg_similarity.jarowinkler_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.jarowinkler_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Jarowinkler.	–

Nama parameter	Deskripsi	Default
pg_similarity.levenshtein_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.levenshtein_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Levenshtein.	–
pg_similarity.matching_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.matching_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Koefisien Pencocokan.	–
pg_similarity.matching_tokenizer	Menetapkan pentokenisasi untuk ukuran kesamaan Koefisien Pencocokan.	–
pg_similarity.mongeelkan_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.mongeelkan_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Monge-Elkan.	–
pg_similarity.mongeelkan_tokenizer	Menetapkan pentokenisasi untuk ukuran kesamaan Monge-Elkan.	–
pg_similarity.nw_gap_penalty	Menetapkan penalti kesenjangan yang digunakan oleh ukuran kesamaan Needleman-Wunsch.	–
pg_similarity.nw_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.nw_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Needleman-Wunsch.	–
pg_similarity.overlap_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–

Nama parameter	Deskripsi	Default
pg_similarity.overlap_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Koefisien Tumpang-Tindih.	–
pg_similarity.overlap_tokenizer	Menetapkan pentokenisasi untuk ukuran kesamaan Koefisien Tumpang-Tindih.	–
pg_similarity.qgram_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.qgram_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Q-Gram.	–
pg_similarity.qgram_tokenizer	Menetapkan pentokenisasi untuk ukuran Q-Gram.	–
pg_similarity.swg_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.swg_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Smith-Waterman-Gotoh.	–
pg_similarity.sw_is_normalized	Menetapkan apakah nilai hasil dinormalisasi atau tidak.	–
pg_similarity.sw_threshold	Menetapkan ambang batas yang digunakan oleh ukuran kesamaan Smith-Waterman.	–
pg_stat_statements.max	Menetapkan jumlah maksimum pernyataan yang dilacak oleh pg_stat_statements.	–
pg_stat_statements.save	Save pg_stat_statements statistics across server shutdowns.	–
pg_stat_statements.track	Memilih pernyataan mana yang dilacak oleh pg_stat_statements.	–

Nama parameter	Deskripsi	Default
pg_stat_statements.track_planning	Memilih apakah durasi perencanaan dilacak oleh pg_stat_statements.	–
pg_stat_statements.track_utility	Memilih apakah perintah utilitas dilacak oleh pg_stat_statements.	–
postgis.gdal_enabled_drivers	Mengaktifkan atau menonaktifkan driver GDAL yang digunakan dengan PostGIS di Postgres 9.3.5 dan versi lebih tinggi.	ENABLE_ALL
quote_all_identifiers	Saat membuat fragmen SQL, mengutip semua pengidentifikasi.	–
random_page_cost	Menetapkan perkiraan perencana untuk biaya halaman disk yang diambil secara tidak berurutan.	–
rds.enable_memory_management	Meningkatkan kemampuan manajemen memori di Aurora PostgreSQL 12.17, 13.13, 14.10, 15.5, dan versi yang lebih tinggi yang mencegah masalah stabilitas dan restart database yang disebabkan oleh memori bebas yang tidak mencukupi. Untuk informasi selengkapnya, lihat Manajemen memori yang ditingkatkan dalam Aurora PostgreSQL.	Benar
rds.force_admin_logging_level	Melihat pesan log untuk tindakan pengguna admin RDS di basis data pelanggan.	–
rds.log_retention_period	Amazon RDS akan menghapus log PostgreSQL yang lebih lama dari N menit.	4320

Nama parameter	Deskripsi	Default
rds.memory_allocation_guard	Meningkatkan kemampuan manajemen memori di Aurora PostgreSQL 11.21, 12.16, 13.12, 14.9, 15.4, dan versi lama yang mencegah masalah stabilitas dan restart database yang disebabkan oleh memori bebas yang tidak mencukupi. Untuk informasi selengkapnya, lihat Manajemen memori yang ditingkatkan dalam Aurora PostgreSQL.	False
rds.pg_stat_ramdisk_size	Ukuran statistik ramdisk dalam MB. Nilai bukan nol akan mengatur ramdisk.	0
rds.rds_superuser_reserved_connections	Menetapkan jumlah slot koneksi yang disediakan untuk rds_superusers.	2
rds.superuser_variables	Daftar variabel superuser-only yang pernyataan modifikasi rds_superuser-nya dielevasikan.	session_replication_role
remove_temp_files_after_crash	Menghapus file sementara setelah backend crash.	0
restart_after_crash	Menginisialisasi ulang server setelah backend crash.	–
row_security	Mengaktifkan keamanan baris.	–
search_path	Menetapkan urutan pencarian skema untuk nama yang tidak memenuhi syarat skema.	–
seq_page_cost	Menetapkan perkiraan perencana untuk biaya halaman disk yang diambil secara berurutan.	–
session_replication_role	Menetapkan perilaku sesi untuk pemicu dan aturan penulisan ulang.	–

Nama parameter	Deskripsi	Default
shared_buffers	(8kB) Menetapkan jumlah buffer memori bersama yang digunakan oleh server.	JUMLAH (InstanceClassMemoryDB/12038, -50003
shared_preload_libraries	Menampilkan daftar pustaka bersama untuk dimuat sebelumnya ke server.	pg_stat_statements
ssl_ca_file	Lokasi file otoritas server SSL.	/rdsdbdata/rds-metadata/ca-cert.pem
ssl_cert_file	Lokasi file sertifikat server SSL.	/rdsdbdata/rds-metadata/server-cert.pem
ssl_crl_dir	Lokasi direktori daftar pencabutan sertifikat SSL.	/rdsdbdata/rds-metadata/ssl_crl_dir/
ssl_key_file	Lokasi file kunci privat server SSL	/rdsdbdata/rds-metadata/server-key.pem
standard_conforming_strings	Menyebabkan string ... memperlakukan garis miring secara literal.	–
statement_timeout	(ms) Menetapkan durasi maksimum yang diizinkan untuk setiap pernyataan.	–
stats_temp_directory	Menulis file statistik sementara ke direktori yang ditentukan.	/rdsdbdata/db/pg_stat_tmp
superuser_reserved_connections	Menetapkan jumlah slot koneksi yang disediakan untuk pengguna super.	3
synchronize_seqscans	Mengaktifkan pemindaian berurutan yang disinkronkan.	–
tcp_keepalives_count	Jumlah maksimum pengiriman ulang keepalive TCP.	–
tcp_keepalives_idle	(s) Waktu di antara penerbitan keepalive TCP.	–

Nama parameter	Deskripsi	Default
tcp_keepalives_interval	(s) Waktu di antara pengiriman ulang keepalive TCP.	–
temp_buffers	(8kB) Menetapkan jumlah maksimum buffer sementara yang digunakan oleh setiap sesi.	–
temp_file_limit	Membatasi jumlah total ruang disk dalam kilobyte yang dapat digunakan oleh proses PostgreSQL tertentu untuk file sementara, tidak termasuk ruang yang digunakan untuk tabel sementara eksplisit	-1
temp_tablespaces	Menetapkan ruang tabel yang akan digunakan di tabel sementara dan sort file.	–
track_activities	Mengumpulkan informasi tentang eksekusi perintah.	–
track_activity_query_size	Menetapkan ukuran yang disediakan untuk pg_stat_activity.current_query, dalam byte.	4096
track_counts	Mengumpulkan statistik tentang aktivitas basis data.	–
track_functions	Mengumpulkan statistik tingkat fungsi tentang aktivitas basis data.	pl
track_io_timing	Mengumpulkan statistik waktu tentang aktivitas IO basis data.	1
transform_–_equals	Memperlakukan expr=– sebagai expr IS –.	–
update_process_title	Memperbarui judul proses untuk menampilkan perintah SQL aktif.	–
wal_receiver_status_interval	(s) Menetapkan interval maksimum di antara laporan status penerima WAL ke primer.	–

Nama parameter	Deskripsi	Default
work_mem	(kB) Menetapkan memori maksimum yang akan digunakan untuk workspace kueri.	–
xmlbinary	Menetapkan cara nilai biner akan dienkode dalam XML.	–
xmloption	Menetapkan apakah data XML dalam operasi penguraian implisit dan serialisasi dianggap sebagai dokumen atau fragmen konten.	–

Peristiwa tunggu Amazon Aurora PostgreSQL

Berikut ini adalah peristiwa tunggu yang umum terjadi di Aurora PostgreSQL. Untuk mempelajari selengkapnya tentang peristiwa tunggu dan menyetel klaster DB PostgreSQL, lihat Menyetel dengan peristiwa tunggu di Aurora PostgreSQL.

Aktivitas: ArchiverMain

Proses pengarsip menunggu aktivitas.

Aktivitas: AutoVacuumMain

Proses peluncur autovacuum menunggu aktivitas.

Aktivitas: BgWriterHibernate

Proses penulis latar belakang berhibernasi sambil menunggu aktivitas.

Aktivitas: BgWriterMain

Proses penulis latar belakang menunggu aktivitas.

Aktivitas: CheckpointerMain

Proses checkpointer menunggu aktivitas.

Aktivitas: LogicalApplyMain

Proses penerapan replikasi logis menunggu aktivitas.

Aktivitas: LogicalLauncherMain

Proses peluncur replikasi logis menunggu aktivitas.

Aktivitas: PgStatMain

Proses pengumpul statistik menunggu aktivitas.

Aktivitas: RecoveryWalAll

Sebuah proses menunggu write-ahead log (WAL) dari aliran saat pemulihan.

Aktivitas: RecoveryWalStream

Proses pengaktifan menunggu write-ahead log (WAL) untuk tiba selama pemulihan aliran.

Aktivitas: SysLoggerMain

Proses syslogger menunggu aktivitas.

Aktivitas: WalReceiverMain

Proses penerima write-ahead log (WAL) menunggu aktivitas.

Aktivitas: WalSenderMain

Proses pengirim write-ahead log (WAL) menunggu aktivitas.

Aktivitas: WalWriterMain

Proses penulis write-ahead log (WAL) menunggu aktivitas.

BufferPin:BufferPin

Sebuah proses menunggu untuk mendapatkan pin eksklusif pada buffer.

Klien:GSS OpenServer

Sebuah proses menunggu untuk membaca data dari klien sambil membuat sesi Generic Security Service Application Program Interface (GSSAPI).

Klien: ClientRead

Sebuah proses backend menunggu untuk menerima data dari klien PostgreSQL. Untuk informasi selengkapnya, lihat Client:ClientRead.

Klien: ClientWrite

Sebuah proses backend menunggu untuk mengirim lebih banyak data ke klien PostgreSQL. Untuk informasi selengkapnya, lihat Client:ClientWrite.

Klien:libpq WalReceiverConnect

Sebuah proses menunggu di penerima write-ahead log (WAL) untuk membuat koneksi ke server jarak jauh.

Klien:libpq WalReceiverReceive

Sebuah proses menunggu di penerima write-ahead log (WAL) untuk menerima data dari server jarak jauh.

Klien:SSL OpenServer

Sebuah proses menunggu Lapisan Soket Aman (SSL) saat mencoba koneksi.

Klien: WalReceiverWaitStart

Sebuah proses menunggu proses pengaktifan untuk mengirim data awal untuk replikasi aliran.

Klien: WalSenderWaitFor WAL

Sebuah proses menunggu write-ahead log (WAL) untuk dibuang dalam proses pengirim WAL.

Klien: WalSenderWriteData

Sebuah proses menunggu aktivitas saat memproses balasan dari penerima write-ahead log (WAL) dalam proses pengirim WAL.

CPU

Sebuah proses backend aktif atau menunggu CPU. Untuk informasi selengkapnya, lihat CPU.

Extension:extension

Sebuah proses backend menunggu kondisi yang ditentukan oleh ekstensi atau modul.

IO: AuroraOptimizedReadsCacheRead

Sebuah proses menunggu pembacaan dari cache bertingkat Optimized Reads karena halaman tidak tersedia dalam memori bersama.

IO: AuroraOptimizedReads CacheSegmentMemotong

Sebuah proses menunggu file segmen cache bertingkat Optimized Reads untuk dipotong.

IO: AuroraOptimizedReadsCacheWrite

Proses penulis latar belakang menunggu untuk menulis di cache bertingkat Optimized Reads.

IO: AuroraStorageLogAllocate

Sebuah sesi mengalokasikan metadata dan mempersiapkan penulisan log transaksi.

IO: BufFileRead

Ketika operasi membutuhkan lebih banyak memori daripada jumlah yang ditentukan oleh parameter memori kerja, mesin akan membuat file sementara pada disk. Peristiwa tunggu

ini terjadi ketika operasi membaca dari file sementara. Untuk informasi selengkapnya, lihat IO:BufFileRead dan IO:BufFileWrite.

IO: BufFileWrite

Ketika operasi membutuhkan lebih banyak memori daripada jumlah yang ditentukan oleh parameter memori kerja, mesin akan membuat file sementara pada disk. Peristiwa tunggu ini terjadi ketika operasi menulis ke file sementara. Untuk informasi selengkapnya, lihat IO:BufFileRead dan IO:BufFileWrite.

IO: ControlFileRead

Sebuah proses menunggu pembacaan dari file `pg_control`.

IO: ControlFileSync

Sebuah proses menunggu file `pg_control` untuk mencapai penyimpanan durabel.

IO: ControlFileSyncUpdate

Sebuah proses menunggu pembaruan pada file `pg_control` untuk mencapai penyimpanan durabel.

IO: ControlFileWrite

Sebuah proses menunggu penulisan pada file `pg_control`.

IO: ControlFileWriteUpdate

Sebuah proses menunggu penulisan untuk memperbarui file `pg_control`.

IO: CopyFileRead

Sebuah proses menunggu pembacaan selama operasi penyalinan file.

IO: CopyFileWrite

Sebuah proses menunggu penulisan selama operasi penyalinan file.

IO: DataFileExtend

Sebuah proses menunggu file data relasi untuk diperluas.

IO: DataFileFlush

Sebuah proses menunggu file data relasi untuk mencapai penyimpanan durabel.

IO: DataFileImmediateSync

Sebuah proses menunggu sinkronisasi langsung file data relasi ke penyimpanan durabel.

IO: DataFilePrefetch

Sebuah proses menunggu prefetch asinkron dari file data relasi.

IO: DataFileSync

Sebuah proses menunggu perubahan pada file data relasi untuk mencapai penyimpanan durabel.

IO: DataFileRead

Sebuah proses backend mencoba menemukan halaman di buffer bersama, tidak menemukannya, sehingga membacanya dari penyimpanan. Untuk informasi selengkapnya, lihat IO:DataFileRead.

IO: DataFileTruncate

Sebuah proses menunggu file data relasi untuk dipotong.

IO: DataFileWrite

Sebuah proses menunggu penulisan pada file data relasi.

IO:DSM FillZeroWrite

Sebuah proses menunggu untuk menulis nol byte pada backing file memori bersama dinamis.

IO: LockFileAddToDataDirRead

Sebuah proses menunggu pembacaan sambil menambahkan baris pada file kunci direktori data.

IO: LockFileAddToDataDirSync

Sebuah proses menunggu data untuk mencapai penyimpanan durabel sambil menambahkan baris pada file kunci direktori data.

IO: LockFileAddToDataDirWrite

Sebuah proses menunggu penulisan sambil menambahkan baris pada file kunci direktori data.

IO: LockFileCreateRead

Sebuah proses menunggu untuk membaca saat membuat file kunci direktori data.

IO: LockFileCreateSync

Sebuah proses menunggu data untuk mencapai penyimpanan durabel sambil membuat file kunci direktori data.

IO: LockFileCreateWrite

Sebuah proses menunggu penulisan saat membuat file kunci direktori data.

IO: LockFileReCheckDataDirRead

Sebuah proses menunggu pembacaan selama pemeriksaan ulang file kunci direktori data.

IO: LogicalRewriteCheckpointSync

Sebuah proses menunggu pemetaan penulisan ulang logis untuk mencapai penyimpanan durabel selama checkpoint.

IO: LogicalRewriteMappingSync

Sebuah proses menunggu pemetaan data untuk mencapai penyimpanan durabel selama penulisan ulang logis.

IO: LogicalRewriteMappingWrite

Sebuah proses menunggu penulisan data pemetaan selama penulisan ulang logis.

IO: LogicalRewriteSync

Sebuah proses menunggu pemetaan penulisan ulang logis untuk mencapai penyimpanan durabel.

IO: LogicalRewriteTruncate

Sebuah proses menunggu pemotongan data pemetaan selama penulisan ulang logis.

IO: LogicalRewriteWrite

Sebuah proses menunggu penulisan pemetaan penulisan ulang logis.

IO: RelationMapRead

Sebuah proses menunggu pembacaan file peta relasi.

IO: RelationMapSync

Sebuah proses menunggu file peta relasi untuk mencapai penyimpanan durabel.

IO: RelationMapWrite

Sebuah proses menunggu penulisan pada file peta relasi.

IO: ReorderBufferRead

Sebuah proses menunggu pembacaan selama manajemen buffer penyusunan ulang.

IO: ReorderBufferWrite

Sebuah proses menunggu penulisan selama manajemen buffer penyusunan ulang.

IO: ReorderLogicalMappingRead

Sebuah proses menunggu pembacaan pemetaan logis selama manajemen buffer penyusunan ulang.

IO: ReplicationSlotRead

Sebuah proses menunggu pembacaan dari file kontrol slot replikasi.

IO: ReplicationSlotRestoreSync

Sebuah proses menunggu file kontrol slot replikasi untuk mencapai penyimpanan durabel sambil memulihkannya ke memori.

IO: ReplicationSlotSync

Sebuah proses menunggu file kontrol slot replikasi untuk mencapai penyimpanan durabel.

IO: ReplicationSlotWrite

Sebuah proses menunggu penulisan pada file kontrol slot replikasi.

IO:SLRU FlushSync

Sebuah proses menunggu data simple least-recently used (SLRU) untuk mencapai penyimpanan durabel selama checkpoint atau penonaktifan basis data.

IO:SLRURead

Sebuah proses menunggu pembacaan halaman simple least-recently used (SLRU).

IO:SLRUSync

Sebuah proses menunggu data simple least-recently used (SLRU) untuk mencapai penyimpanan durabel setelah penulisan halaman.

IO:SLRUWrite

Sebuah proses menunggu penulisan halaman simple least-recently used (SLRU).

IO: SnapbuildRead

Sebuah proses menunggu pembacaan snapshot katalog historis terserialisasi.

IO: SnapbuildSync

Sebuah proses menunggu snapshot katalog historis terserialisasi untuk mencapai penyimpanan durabel.

IO: SnapbuildWrite

Sebuah proses menunggu penulisan snapshot katalog historis terserialisasi.

IO: TimelineHistoryFileSync

Sebuah proses menunggu file riwayat lini masa yang diterima melalui replikasi aliran untuk mencapai penyimpanan durabel.

IO: TimelineHistoryFileWrite

Sebuah proses menunggu penulisan file riwayat lini masa yang diterima melalui replikasi aliran.

IO: TimelineHistoryRead

Sebuah proses menunggu pembacaan file riwayat lini masa.

IO: TimelineHistorySync

Sebuah proses menunggu file riwayat lini masa yang baru dibuat untuk mencapai penyimpanan durabel.

IO: TimelineHistoryWrite

Sebuah proses menunggu penulisan file riwayat lini masa yang baru dibuat.

IO: TwophaseFileRead

Sebuah proses menunggu pembacaan file status dua fase.

IO: TwophaseFileSync

Sebuah proses menunggu file status dua fase untuk mencapai penyimpanan durabel.

IO: TwophaseFileWrite

Sebuah proses menunggu penulisan file status dua fase.

IO:WAL BootstrapSync

Sebuah proses menunggu write-ahead log (WAL) untuk mencapai penyimpanan durabel selama bootstrapping.

IO:WAL BootstrapWrite

Sebuah proses menunggu penulisan halaman write-ahead log (WAL) selama bootstrapping.

IO:WAL CopyRead

Sebuah proses menunggu pembacaan saat membuat segmen write-ahead log (WAL) baru dengan menyalin yang sudah ada.

IO:WAL CopySync

Sebuah proses menunggu segmen write-ahead log (WAL) baru yang dibuat dengan menyalin yang sudah ada untuk mencapai penyimpanan durabel.

IO:WAL CopyWrite

Sebuah proses menunggu penulisan saat membuat segmen write-ahead log (WAL) baru dengan menyalin yang sudah ada.

IO:WAL InitSync

Sebuah proses menunggu file write-ahead log (WAL) yang baru diinisialisasi untuk mencapai penyimpanan durabel.

IO:WAL InitWrite

Sebuah proses menunggu penulisan saat menginisialisasi file write-ahead log (WAL).

IO:WALRead

Sebuah proses menunggu pembacaan dari file write-ahead log (WAL).

IO:WAL SenderTimelineHistoryRead

Sebuah proses menunggu pembacaan dari file riwayat lini masa selama perintah lini masa pengirim WAL.

IO:WALSync

Sebuah proses menunggu file write-ahead log (WAL) untuk mencapai penyimpanan durabel.

IO:WAL SyncMethodAssign

Sebuah proses menunggu data untuk mencapai penyimpanan durabel sambil menetapkan metode sinkronisasi write-ahead log (WAL) baru.

IO:WALWrite

Sebuah proses menunggu penulisan pada file write-ahead log (WAL).

IO: XactSync

Sebuah proses backend menunggu subsistem penyimpanan Aurora untuk mengonfirmasi komit transaksi reguler, atau komit atau rollback transaksi yang disiapkan. Untuk informasi selengkapnya, lihat IO:XactSync.

IPC: BackupWaitWalArchive

Sebuah proses menunggu file write-ahead log (WAL) yang diperlukan agar cadangan berhasil diarsipkan.

IPC: AuroraOptimizedReadsCacheWriteStop

Sebuah proses sedang menunggu penulis latar belakang berhenti menulis ke cache berjenjang Optimized Reads.

IPC: BgWorkerShutdown

Sebuah proses menunggu pekerja latar belakang untuk dinonaktifkan.

IPC: BgWorkerStartup

Sebuah proses menunggu pekerja latar belakang untuk dimulai.

IPC: BtreePage

Sebuah proses menunggu nomor halaman yang diperlukan untuk melanjutkan pemindaian B-tree paralel menjadi tersedia.

IPC: CheckpointDone

Sebuah proses menunggu checkpoint selesai.

IPC: CheckpointStart

Sebuah proses menunggu checkpoint untuk dimulai.

IPC: ClogGroupUpdate

Sebuah proses menunggu pemimpin grup untuk memperbarui status transaksi di akhir transaksi.

IPC: DamRecordTxAck

Sebuah proses backend telah menghasilkan peristiwa aliran aktivitas basis data dan menunggu peristiwa menjadi durabel. Untuk informasi selengkapnya, lihat IPC:DamRecordTxAck.

IPC: ExecuteGather

Sebuah proses menunggu aktivitas dari proses turunan saat menjalankan simpul rencana Gather.

IPC:Hash/Batch/Allocating

Sebuah proses menunggu peserta hash paralel terpilih untuk mengalokasikan tabel hash.

IPC:Hash/Batch/Electing

Sebuah proses memilih peserta hash paralel untuk mengalokasikan tabel hash.

IPC:Hash/Batch/Loading

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan pemuatan tabel hash.

IPC:Hash/Build/Allocating

Sebuah proses menunggu peserta hash paralel terpilih untuk mengalokasikan tabel hash awal.

IPC:Hash/Build/Electing

Sebuah proses memilih peserta hash paralel untuk mengalokasikan tabel hash awal.

IPC: Hash/Build/ HashingInner

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan hashing relasi dalam.

IPC: Hash/Build/ HashingOuter

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan partisi relasi luar.

IPC: Hash//Mengalokasikan GrowBatches

Sebuah proses menunggu peserta hash paralel terpilih untuk mengalokasikan lebih banyak batch.

IPC: GrowBatches Hash//Memutuskan

Sebuah proses memilih peserta hash paralel untuk memutuskan pertumbuhan batch mendatang.

IPC: Hash// GrowBatches Memilih

Sebuah proses memilih peserta hash paralel untuk mengalokasikan lebih banyak batch.

IPC: Hash//Menyelesaikan GrowBatches

Sebuah proses menunggu peserta hash paralel terpilih untuk memutuskan pertumbuhan batch mendatang.

IPC: Hash//Repartisi GrowBatches

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan partisi ulang.

IPC: Hash//Mengalokasikan GrowBuckets

Sebuah proses menunggu peserta hash paralel terpilih untuk menyelesaikan alokasi lebih banyak bucket.

IPC: Hash// GrowBuckets Memilih

Sebuah proses memilih peserta hash paralel untuk mengalokasikan lebih banyak bucket.

IPC: GrowBuckets Hash//Memasukkan kembali

Sebuah proses menunggu peserta hash paralel lainnya selesai menyisipkan tuple ke dalam bucket baru.

IPC: HashBatchAllocate

Sebuah proses menunggu peserta hash paralel terpilih untuk mengalokasikan tabel hash.

IPC: HashBatchElect

Sebuah proses menunggu untuk memilih peserta hash paralel untuk mengalokasikan tabel hash.

IPC: HashBatchLoad

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan pemuatan tabel hash.

IPC: HashBuildAllocate

Sebuah proses menunggu peserta hash paralel terpilih untuk mengalokasikan tabel hash awal.

IPC: HashBuildElect

Sebuah proses menunggu untuk memilih peserta hash paralel untuk mengalokasikan tabel hash awal.

IPC: HashBuildHashInner

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan hashing relasi dalam.

IPC: 'HashBuildHashOuter

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan partisi relasi luar.

IPC: HashGrowBatchesAllocate

Sebuah proses menunggu peserta hash paralel terpilih untuk mengalokasikan lebih banyak batch.

IPC: 'HashGrowBatchesDecide

Sebuah proses menunggu untuk memilih peserta hash paralel untuk memutuskan pertumbuhan batch mendatang.

IPC: HashGrowBatchesElect

Sebuah proses menunggu untuk memilih peserta hash paralel untuk mengalokasikan lebih banyak batch.

IPC: HashGrowBatchesFinish

Sebuah proses menunggu peserta hash paralel terpilih untuk memutuskan pertumbuhan batch mendatang.

IPC: HashGrowBatchesRepartition

Sebuah proses menunggu peserta hash paralel lainnya untuk menyelesaikan partisi ulang.

IPC: HashGrowBucketsAllocate

Sebuah proses menunggu peserta hash paralel terpilih untuk menyelesaikan alokasi lebih banyak bucket.

IPC: HashGrowBucketsElect

Sebuah proses menunggu untuk memilih peserta hash paralel untuk mengalokasikan lebih banyak bucket.

IPC: HashGrowBucketsReinsert

Sebuah proses menunggu peserta hash paralel lainnya selesai menyisipkan tuple ke dalam bucket baru.

IPC: LogicalSyncData

Sebuah proses menunggu server jarak jauh replikasi logis untuk mengirim data untuk sinkronisasi tabel awal.

IPC: LogicalSyncStateChange

Sebuah proses menunggu server jarak jauh replikasi logis untuk mengubah status.

IPC: MessageQueueInternal

Sebuah proses menunggu proses lain untuk dilampirkan ke antrean pesan bersama.

IPC: MessageQueuePutMessage

Sebuah proses menunggu untuk menulis pesan protokol ke antrean pesan bersama.

IPC: MessageQueueReceive

Sebuah proses menunggu untuk menerima byte dari antrean pesan bersama.

IPC: MessageQueueSend

Sebuah proses menunggu untuk mengirim byte ke antrean pesan bersama.

IPC: ParallelBitmapScan

Sebuah proses menunggu pemindaian bitmap paralel untuk diinisialisasi.

IPC: ParallelCreateIndexScan

Sebuah proses menunggu pekerja CREATE INDEX paralel untuk menyelesaikan pemindaian heap.

IPC: ParallelFinish

Sebuah proses menunggu pekerja paralel untuk menyelesaikan penghitungan.

IPC: ProcArrayGroupUpdate

Sebuah proses menunggu pemimpin grup untuk menghapus ID transaksi di akhir operasi paralel.

IPC: ProcSignalBarrier

Sebuah proses menunggu peristiwa penghalang untuk diproses oleh semua backend.

IPC:Promote

Sebuah proses menunggu promosi siaga.

IPC: RecoveryConflictSnapshot

Sebuah proses menunggu penyelesaian pertentangan pemulihan untuk pembersihan vacuum.

IPC: RecoveryConflictTablespace

Sebuah proses menunggu penyelesaian pertentangan pemulihan untuk menghapus ruang tabel.

IPC: RecoveryPause

Sebuah proses menunggu pemulihan untuk dilanjutkan.

IPC: ReplicationOriginDrop

Sebuah proses menunggu asal replikasi menjadi tidak aktif agar dapat dihapus.

IPC: ReplicationSlotDrop

Sebuah proses menunggu slot replikasi menjadi tidak aktif agar dapat dihapus.

IPC: SafeSnapshot

Sebuah proses menunggu untuk mendapatkan snapshot yang valid untuk transaksi READ ONLY DEFERRABLE.

IPC: SyncRep

Sebuah proses menunggu konfirmasi dari server jarak jauh selama replikasi sinkron.

IPC: XactGroupUpdate

Sebuah proses menunggu pemimpin grup untuk memperbarui status transaksi pada akhir operasi paralel.

Lock:advisory

Sebuah proses backend meminta kunci advisory dan menunggunya. Untuk informasi selengkapnya, lihat Lock:advisory.

Lock:extend

Sebuah proses backend menunggu kunci dirilis agar dapat memperluas relasi. Kunci ini diperlukan karena hanya satu proses backend yang dapat memperluas relasi pada satu waktu. Untuk informasi selengkapnya, lihat Lock:extend.

Lock:frozenid

Sebuah proses menunggu untuk memperbarui `pg_database.datfrozenxid` dan `pg_database.datminmxid`.

Lock:object

Sebuah proses menunggu untuk mendapatkan kunci pada objek basis data nonrelasi.

Lock:page

Sebuah proses menunggu untuk mendapatkan kunci pada halaman relasi.

Lock:Relation

Sebuah proses backend menunggu untuk mendapatkan kunci pada relasi yang dikunci oleh transaksi lain. Untuk informasi selengkapnya, lihat Lock:Relation.

Lock:spectoken

Sebuah proses menunggu untuk mendapatkan kunci penyisipan spekulatif.

Lock:speculative token

Sebuah proses menunggu untuk mendapatkan kunci penyisipan spekulatif.

Lock:transactionid

Sebuah transaksi menunggu kunci tingkat baris. Untuk informasi selengkapnya, lihat Lock:transactionid.

Lock:tuple

Sebuah proses backend menunggu untuk mendapatkan kunci pada tuple sementara proses backend lain memegang kunci yang bertentangan pada tuple yang sama. Untuk informasi selengkapnya, lihat Lock:tuple.

Lock:userlock

Sebuah proses menunggu untuk mendapatkan kunci pengguna.

Lock:virtualxid

Sebuah proses menunggu untuk mendapatkan kunci ID transaksi virtual.

LWLock: AddinShmemInit

Sebuah proses menunggu untuk mengelola alokasi ruang ekstensi dalam memori bersama.

LWLock: AddinShmemInitLock

Sebuah proses menunggu untuk mengelola alokasi ruang dalam memori bersama.

LWLock:async

Sebuah proses menunggu I/O pada buffer asinkron (notify).

LWLock: AsyncCtlLock

Sebuah proses menunggu untuk membaca atau memperbarui status notifikasi bersama.

LWLock: AsyncQueueLock

Sebuah proses menunggu untuk membaca atau memperbarui pesan notifikasi.

LWLock: AuroraOptimizedReadsCacheMapping

Sebuah proses menunggu untuk mengaitkan blok data dengan halaman di cache bertingkat Optimized Reads.

LWLock: AutoFile

Sebuah proses menunggu untuk memperbarui file `postgresql.auto.conf`.

LWLock: AutoFileLock

Sebuah proses menunggu untuk memperbarui file `postgresql.auto.conf`.

LWLock:Autovacuum

Sebuah proses menunggu untuk membaca atau memperbarui status pekerja autovacuum saat ini.

LWLock: AutovacuumLock

Sebuah pekerja atau peluncur autovacuum menunggu untuk memperbarui atau membaca status pekerja autovacuum saat ini.

LWLock: AutovacuumSchedule

Sebuah proses menunggu untuk memastikan bahwa tabel yang dipilih untuk autovacuum masih memerlukan vacuuming.

LWLock: AutovacuumScheduleLock

Sebuah proses menunggu untuk memastikan bahwa tabel yang telah dipilihnya untuk vacuum masih memerlukan vacuuming.

LWLock: BackendRandomLock

Sebuah proses menunggu untuk menghasilkan angka acak.

LWLock: BackgroundWorker

Sebuah proses menunggu untuk membaca atau memperbarui status pekerja latar belakang.

LWLock: BackgroundWorkerLock

Sebuah proses menunggu untuk membaca atau memperbarui status pekerja latar belakang.

LWLock: BtreeVacuum

Sebuah proses menunggu untuk membaca atau memperbarui informasi terkait vacuum untuk indeks B-tree.

LWLock: BtreeVacuumLock

Sebuah proses menunggu untuk membaca atau memperbarui informasi terkait vacuum untuk indeks B-tree.

LWLock:buffer_content

Sebuah proses backend menunggu untuk memperoleh kunci ringan pada konten buffer memori bersama. Untuk informasi selengkapnya, lihat LWLock:buffer_content (BufferContent).

LWLock:buffer_mapping

Sebuah proses backend menunggu untuk mengaitkan blok data dengan buffer di kumpulan buffer bersama. Untuk informasi selengkapnya, lihat LWLock:buffer_mapping.

LWLock:BufferIO

Sebuah proses backend ingin membacakan halaman ke memori bersama. Proses ini menunggu proses lain untuk menyelesaikan I/O-nya untuk halaman. Untuk informasi selengkapnya, lihat LWLock:BufferIO (IPC:BufferIO).

LWLock:Checkpoint

Sebuah proses menunggu untuk memulai checkpoint.

LWLock: CheckpointLock

Sebuah proses menunggu untuk melakukan checkpoint.

LWLock: CheckpointerComm

Sebuah proses menunggu untuk mengelola permintaan `fsync`.

LWLock: CheckpointerCommLock

Sebuah proses menunggu untuk mengelola permintaan `fsync`.

LWLock:clog

Sebuah proses menunggu I/O pada buffer clog (status transaksi).

LWLOCK: C LogControlLock

Sebuah proses menunggu untuk membaca atau memperbarui status transaksi.

LWLOCK: C LogTruncationLock

Sebuah proses menunggu untuk menjalankan `txid_status` atau memperbarui ID transaksi paling lama yang tersedia untuknya.

LWLock:commit_timestamp

Sebuah proses menunggu I/O pada buffer stempel waktu komit.

LWLock: CommitTs

Sebuah proses menunggu untuk membaca atau memperbarui nilai terakhir yang ditetapkan untuk stempel waktu komit transaksi.

LWLock: CommitTsBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk stempel waktu komit.

LWLock: CommitTsControlLock

Sebuah proses menunggu untuk membaca atau memperbarui stempel waktu komit transaksi.

LWLock: CommitTsLock

Sebuah proses menunggu untuk membaca atau memperbarui nilai terakhir yang ditetapkan untuk stempel waktu transaksi.

LWLock: SLRU CommitTs

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk stempel waktu komit.

LWLock: ControlFile

Sebuah proses menunggu untuk membaca atau memperbarui file `pg_control` atau membuat file write-ahead log (WAL) baru.

LWLock: ControlFileLock

Sebuah proses menunggu untuk membaca atau memperbarui file kontrol atau pembuatan file write-ahead log (WAL) baru.

LWLock: DynamicSharedMemoryControl

Sebuah proses menunggu untuk membaca atau memperbarui informasi alokasi memori bersama dinamis.

LWLock: DynamicSharedMemoryControlLock

Sebuah proses menunggu untuk membaca atau memperbarui status memori bersama dinamis.

LWLock:lock_manager

Sebuah proses backend menunggu untuk menambahkan atau memeriksa kunci untuk proses backend. Atau, proses backend ini menunggu untuk bergabung atau keluar dari grup penguncian yang digunakan oleh kueri paralel. Untuk informasi selengkapnya, lihat LWLock:lock_manager.

LWLock: LockFastPath

Sebuah proses menunggu untuk membaca atau memperbarui informasi kunci jalur cepat proses.

LWLock: LogicalRepWorker

Sebuah proses menunggu untuk membaca atau memperbarui status pekerja replikasi logis.

LWLock: LogicalRepWorkerLock

Sebuah proses menunggu tindakan pada pekerja replikasi logis untuk diselesaikan.

LWLock:multixact_member

Sebuah proses menunggu I/O pada buffer multixact_member.

LWLock:multixact_offset

Sebuah proses menunggu I/O pada buffer offset multixact.

LWLock: MultiXactGen

Sebuah proses menunggu untuk membaca atau memperbarui status multixact bersama.

LWLock: MultiXactGenLock

Sebuah proses menunggu untuk membaca atau memperbarui status multixact bersama.

LWLock: MultiXactMemberBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk anggota multixact. Untuk informasi selengkapnya, lihat LWLock: MultiXact.

LWLock: MultiXactMemberControlLock

Sebuah proses menunggu untuk membaca atau memperbarui pemetaan anggota multixact.

LWLock: SLRU MultiXactMember

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk anggota multixact. Untuk informasi selengkapnya, lihat LWLock: MultiXact.

LWLock: MultiXactOffsetBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk offset multixact. Untuk informasi selengkapnya, lihat LWLock: MultiXact.

LWLock: MultiXactOffsetControlLock

Sebuah proses menunggu untuk membaca atau memperbarui pemetaan offset multixact.

LWLock: SLRU MultiXactOffset

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk offset multixact. Untuk informasi selengkapnya, lihat LWLock: MultiXact.

LWLock: MultiXactTruncation

Sebuah proses menunggu untuk membaca atau memotong informasi multixact.

LWLock: MultiXactTruncationLock

Sebuah proses menunggu untuk membaca atau memotong informasi multixact.

LWLock: NotifyBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk pesan NOTIFY.

LWLock: NotifyQueue

Sebuah proses menunggu untuk membaca atau memperbarui pesan NOTIFY.

LWLock: NotifyQueueTail

Sebuah proses menunggu untuk memperbarui batas penyimpanan pesan NOTIFY.

LWLock: NotifyQueueTailLock

Sebuah proses menunggu untuk memperbarui batas penyimpanan pesan notifikasi.

LWLock:NotifySLRU

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk pesan NOTIFY.

LWLock: OidGen

Sebuah proses menunggu untuk mengalokasikan ID objek (OID) baru.

LWLock: OidGenLock

Sebuah proses menunggu untuk mengalokasikan atau menetapkan objek ID (OID).

LWLock:oldserxid

Sebuah proses menunggu I/O pada buffer oldserxid.

LWLock: OldSerXidLock

Sebuah proses menunggu untuk membaca atau mencatat transaksi serialisasi yang bertentangan.

LWLock: OldSnapshotTimeMap

Sebuah proses menunggu untuk membaca atau memperbarui informasi kontrol snapshot lama.

LWLock: OldSnapshotTimeMapLock

Sebuah proses menunggu untuk membaca atau memperbarui informasi kontrol snapshot lama.

LWLock:parallel_append

Sebuah proses menunggu untuk memilih subrencana berikutnya selama eksekusi rencana append paralel.

LWLock:parallel_hash_join

Sebuah proses menunggu untuk mengalokasikan atau menukar potongan memori atau memperbarui penghitung selama eksekusi rencana hash paralel.

LWLock:parallel_query_dsa

Sebuah proses menunggu kunci pada alokasi memori bersama dinamis untuk kueri paralel.

LWLock: ParallelAppend

Sebuah proses menunggu untuk memilih subrencana berikutnya selama eksekusi rencana append paralel.

LWLock: ParallelHashJoin

Sebuah proses menunggu untuk menyinkronkan pekerja selama eksekusi rencana untuk hash join paralel.

Lwlock: DSA ParallelQuery

Sebuah proses menunggu alokasi memori bersama dinamis untuk kueri paralel.

Lwlock: DSA PerSession

Sebuah proses menunggu alokasi memori bersama dinamis untuk kueri paralel.

Lwlock: PerSessionRecordType

Sebuah proses menunggu untuk mengakses informasi kueri paralel tentang tipe komposit.

Lwlock: PerSessionRecordTypmod

Sebuah proses menunggu untuk mengakses informasi kueri paralel tentang pengubah tipe yang mengidentifikasi jenis catatan anonim.

Lwlock: PerXactPredicateList

Sebuah proses menunggu untuk mengakses daftar kunci predikat yang dipegang oleh transaksi yang dapat diserialisasi saat ini selama kueri paralel.

Lwlock:predicate_lock_manager

Sebuah proses menunggu untuk menambahkan atau memeriksa informasi kunci predikat.

Lwlock: PredicateLockManager

Sebuah proses menunggu untuk mengakses informasi kunci predikat yang digunakan oleh transaksi yang dapat diserialisasi.

Lwlock:proc

Sebuah proses menunggu untuk membaca atau memperbarui informasi kunci jalur cepat.

LWLock: ProcArray

Sebuah proses menunggu untuk mengakses struktur data per proses bersama (biasanya, untuk mendapatkan snapshot atau melaporkan ID transaksi sesi).

LWLock: ProcArrayLock

Sebuah proses menunggu untuk mendapatkan snapshot atau penghapusan ID transaksi di akhir transaksi.

LWLock: RelationMapping

Sebuah proses menunggu untuk membaca atau memperbarui file `pg_filenode.map` (digunakan untuk melacak penetapan simpul file untuk katalog sistem tertentu).

LWLock: RelationMappingLock

Sebuah proses sedang menunggu untuk memperbarui file peta relasi yang digunakan untuk menyimpan catalog-to-file-node pemetaan.

LWLock: RelCacheInit

Sebuah proses menunggu untuk membaca atau memperbarui file `pg_internal.init` (file inisialisasi cache relasi).

LWLock: RelCacheInitLock

Sebuah proses menunggu untuk membaca atau menulis file inisialisasi cache relasi.

LWLock:replication_origin

Sebuah proses menunggu untuk membaca atau memperbarui progres replikasi.

LWLock:replication_slot_io

Sebuah proses menunggu I/O pada slot replikasi.

LWLock: ReplicationOrigin

Sebuah proses menunggu untuk membuat, menghapus, atau menggunakan asal replikasi.

LWLock: ReplicationOriginLock

Sebuah proses menunggu untuk mengatur, menghapus, atau menggunakan asal replikasi.

LWLock: ReplicationOriginState

Sebuah proses menunggu untuk membaca atau memperbarui progres satu asal replikasi.

LWLock: ReplicationSlotAllocation

Sebuah proses menunggu untuk mengalokasikan atau membebaskan slot replikasi.

LWLock: ReplicationSlotAllocationLock

Sebuah proses menunggu untuk mengalokasikan atau membebaskan slot replikasi.

LWLock: ReplicationSlotControl

Sebuah proses menunggu untuk membaca atau memperbarui status slot replikasi.

LWLock: ReplicationSlotControlLock

Sebuah proses menunggu untuk membaca atau memperbarui status slot replikasi.

LWLock: IO ReplicationSlot

Sebuah proses menunggu I/O pada slot replikasi.

LWLock: SerialBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk pertentangan transaksi yang dapat diserialisasi.

LWLock: SerializableFinishedList

Sebuah proses menunggu untuk mengakses daftar transaksi yang dapat diserialisasi yang selesai.

LWLock: SerializableFinishedListLock

Sebuah proses menunggu untuk mengakses daftar transaksi yang dapat diserialisasi yang selesai.

LWLock: SerializablePredicateList

Sebuah proses menunggu untuk mengakses daftar kunci predikat yang dipegang oleh transaksi yang dapat diserialisasi.

LWLock: SerializablePredicateLockListLock

Sebuah proses menunggu untuk melakukan operasi pada daftar kunci yang dipegang oleh transaksi yang dapat diserialisasi.

LWLock: SerializableXactHash

Sebuah proses menunggu untuk membaca atau memperbarui informasi tentang transaksi serialisasi.

LWLock: SerializableXactHashLock

Sebuah proses menunggu untuk mengambil atau menyimpan informasi tentang transaksi yang dapat diserialisasi.

LWLock:SerialSLRU

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk pertentangan transaksi yang dapat diserialisasi.

LWLock: SharedTidBitmap

Sebuah proses menunggu untuk mengakses bitmap pengidentifikasi tuple bersama (TID) selama pemindaian indeks bitmap paralel.

LWLock: SharedTupleStore

Sebuah proses menunggu untuk mengakses penyimpanan tuple bersama selama kueri paralel.

LWLock: ShmemIndex

Sebuah proses menunggu untuk menemukan atau mengalokasikan ruang dalam memori bersama.

LWLock: ShmemIndexLock

Sebuah proses menunggu untuk menemukan atau mengalokasikan ruang dalam memori bersama.

LWLOCK: S InvalRead

Sebuah proses menunggu untuk mengambil pesan dari antrean invalidasi katalog bersama.

LWLOCK: S InvalReadLock

Sebuah proses menunggu untuk mengambil atau menghapus pesan dari antrean invalidasi bersama.

LWLOCK: S InvalWrite

Sebuah proses menunggu untuk menambahkan pesan ke antrean invalidasi katalog bersama.

LWLOCK: S InvalWriteLock

Sebuah proses menunggu untuk menambahkan pesan dalam antrean invalidasi bersama.

LWLock:subtrans

Sebuah proses menunggu I/O pada buffer subtransaksi.

LWLock: SubtransBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk subtransaksi.

LWLock: SubtransControlLock

Sebuah proses menunggu untuk membaca atau memperbarui informasi subtransaksi.

LWLock:SubtransSLRU

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk subtransaksi.

LWLock: SyncRep

Sebuah proses menunggu untuk membaca atau memperbarui informasi tentang status replikasi sinkron.

LWLock: SyncRepLock

Sebuah proses menunggu untuk membaca atau memperbarui informasi tentang replika sinkron.

LWLock: SyncScan

Sebuah proses menunggu untuk memilih lokasi awal pemindaian tabel yang disinkronkan.

LWLock: SyncScanLock

Sebuah proses menunggu untuk mendapatkan lokasi awal pemindaian pada tabel untuk pemindaian yang disinkronkan.

LWLock: TablespaceCreate

Sebuah proses menunggu untuk membuat atau menghapus ruang tabel.

LWLock: TablespaceCreateLock

Sebuah proses menunggu untuk membuat atau menghapus ruang tabel.

LWLock:tbm

Sebuah proses menunggu kunci iterator bersama pada tree bitmap (TBM).

LWLock: TwoPhaseState

Sebuah proses menunggu untuk membaca atau memperbarui status transaksi yang disiapkan.

LWLock: TwoPhaseStateLock

Sebuah proses menunggu untuk membaca atau memperbarui status transaksi yang disiapkan.

LWLock:wal_insert

Sebuah proses menunggu untuk menyisipkan write-ahead log (WAL) ke dalam buffer memori.

LWLOCK: wal BufMapping

Sebuah proses menunggu untuk mengganti halaman dalam buffer write-ahead log (WAL).

LWLOCK: wal BufMappingLock

Sebuah proses menunggu untuk mengganti halaman dalam buffer write-ahead log (WAL).

LWLock:WALInsert

Sebuah proses menunggu untuk menyisipkan data write-ahead log (WAL) ke dalam buffer memori.

LWLock:WALWrite

Sebuah proses menunggu buffer write-ahead log (WAL) untuk ditulis ke disk.

LWLOCK: wal WriteLock

Sebuah proses menunggu buffer write-ahead log (WAL) untuk ditulis ke disk.

LWLock: WrapLimitsVacuum

Sebuah proses menunggu untuk memperbarui batas ID transaksi dan konsumsi multixact.

LWLock: WrapLimitsVacuumLock

Sebuah proses menunggu untuk memperbarui batas ID transaksi dan konsumsi multixact.

LWLock: XactBuffer

Sebuah proses menunggu I/O pada buffer simple least-recently used (SLRU) untuk status transaksi.

LWLock:XactSLRU

Sebuah proses menunggu untuk mengakses cache simple least-recently used (SLRU) untuk status transaksi.

LWLock: XactTruncation

Sebuah proses menunggu untuk menjalankan pg_xact_status atau memperbarui ID transaksi paling lama yang tersedia untuknya.

LWLock: XidGen

Sebuah proses menunggu untuk mengalokasikan ID transaksi baru.

LWLock: XidGenLock

Sebuah proses menunggu untuk mengalokasikan atau menetapkan ID transaksi.

Batas waktu: BaseBackupThrottle

Sebuah proses menunggu selama pencadangan dasar saat aktivitas throttling.

Batas waktu: PgSleep

Sebuah proses backend telah memanggil fungsi pg_sleep dan menunggu batas waktu tidur berakhir. Untuk informasi selengkapnya, lihat Timeout:PgSleep.

Batas waktu: RecoveryApplyDelay

Sebuah proses menunggu untuk menerapkan write-ahead log (WAL) selama pemulihan karena pengaturan penundaan.

Batas waktu: RecoveryRetrieveRetryInterval

Sebuah proses menunggu selama pemulihan ketika data write-ahead log (WAL) tidak tersedia dari sumber apa pun (pg_wal, arsip, atau aliran).

Batas waktu: VacuumDelay

Sebuah proses menunggu di titik penundaan vacuum berbasis biaya.

Untuk daftar lengkap peristiwa tunggu PostgreSQL, lihat The Statistics Collector > Wait Event tables dalam dokumentasi PostgreSQL.

Pembaruan Amazon Aurora PostgreSQL

Setelah itu, Anda dapat menemukan informasi tentang rilis dan pembaruan versi mesin Amazon Aurora PostgreSQL. Anda juga dapat menemukan informasi tentang cara meningkatkan mesin Aurora PostgreSQL Anda. Untuk informasi selengkapnya tentang rilis Aurora secara umum, lihat Versi Amazon Aurora.

> **Tip**
>
> Anda dapat meminimalkan waktu henti yang diperlukan untuk meningkatkan klaster DB dengan menggunakan deployment blue/green. Untuk informasi selengkapnya, lihat Menggunakan Deployment Blue/Green untuk pembaruan basis data.

Topik

- Mengidentifikasi versi Amazon Aurora PostgreSQL
- Versi rilis dan mesin Amazon Aurora PostgreSQL
- Versi ekstensi untuk Amazon Aurora PostgreSQL
- Meningkatkan klaster DB Amazon Aurora PostgreSQL
- Rilis dukungan jangka panjang (LTS) Aurora PostgreSQL

Mengidentifikasi versi Amazon Aurora PostgreSQL

Amazon Aurora menyertakan fitur tertentu yang bersifat umum bagi Aurora dan tersedia bagi semua klaster DB Aurora. Aurora mencakup fitur lain yang khusus untuk mesin basis data tertentu yang didukung Aurora. Fitur-fitur ini tersedia hanya untuk klaster DB Aurora yang menggunakan mesin basis data tersebut, seperti Aurora PostgreSQL.

Basis data Aurora biasanya memiliki dua nomor versi, nomor versi mesin basis data dan nomor versi Aurora. Jika rilis Aurora PostgreSQL memiliki nomor versi Aurora, nomor tersebut disertakan setelah nomor versi mesin dalam daftar Versi rilis dan mesin Amazon Aurora PostgreSQL.

Nomor versi Aurora

Nomor versi Aurora menggunakan skema penamaan *mayor*.*minor*.*patch*. Versi patch Aurora mencakup perbaikan bug penting yang ditambahkan ke versi minor setelah dirilis. Untuk mempelajari

selengkapnya tentang rilis Amazon Aurora mayor, minor, dan patch, lihat Versi mayor Amazon Aurora, Versi minor Amazon Aurora, dan Versi patch Amazon Aurora.

Anda dapat mengetahui nomor versi Aurora dari instans DB Aurora PostgreSQL dengan kueri SQL berikut:

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

Mulai dengan rilis PostgreSQL versi 13.3, 12.8, 11.13, 10.18, dan untuk semua versi lain yang lebih baru, nomor versi Aurora lebih diselaraskan ke versi mesin PostgreSQL. Misalnya, mengueri klaster DB Aurora PostgreSQL 13.3 mengembalikan hal berikut:

```
aurora_version
----------------
 13.3.1
(1 row)
```

Rilis sebelumnya, seperti klaster DB Aurora PostgreSQL 10.14, mengembalikan nomor versi yang mirip dengan berikut ini:

```
aurora_version
----------------
 2.7.3
(1 row)
```

Nomor versi mesin PostgreSQL

Dimulai dengan PostgreSQL 10, versi mesin basis data PostgreSQL menggunakan skema penomoran *mayor*.*minor* untuk semua rilis. Beberapa contoh termasuk PostgreSQL 10.18, PostgreSQL 12.7, dan PostgreSQL 13.3.

Rilis sebelum PostgreSQL 10 menggunakan skema penomoran *mayor*.*mayor*.*minor* di mana dua digit pertama membentuk nomor versi utama dan digit ketiga menunjukkan versi minor. Misalnya, PostgreSQL 9.6 adalah versi mayor, dengan versi minor 9.6.21 atau 9.6.22 ditunjukkan oleh digit ketiga.

> **Note**
>
> PostgreSQL versi mesin 9.6 tidak lagi didukung. Untuk meningkatkan, lihat Meningkatkan klaster DB Amazon Aurora PostgreSQL. Untuk kebijakan dan lini masa rilis versi, lihatBerapa lama versi mayor Amazon Aurora tetap tersedia.

Anda dapat mengetahui nomor versi mesin basis data PostgreSQL dengan kueri SQL berikut:

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

Untuk klaster DB Aurora PostgreSQL 13.3, hasilnya adalah sebagai berikut:

```
version
-----------------------------------------------------------------------------------------------------
 PostgreSQL 13.3 on x86_64-pc-linux-gnu, compiled by x86_64-pc-linux-gnu-gcc (GCC)
 7.4.0, 64-bit
(1 row)
```

Versi rilis dan mesin Amazon Aurora PostgreSQL

Amazon Aurora Edisi Kompatibel PostgreSQL diperbarui secara berkala. Pembaruan diterapkan pada klaster DB Aurora PostgreSQL selama jadwal pemeliharaan sistem. Saat pembaruan diterapkan bergantung pada jenis pembaruan, Wilayah AWS, dan pengaturan periode pemeliharaan untuk klaster DB. Banyak rilis yang tercantum mencakup baik nomor versi PostgreSQL maupun nomor versi Amazon Aurora. Namun, sejak rilis PostgreSQL versi 13.3, 12.8, 11.13, 10.18, dan untuk semua versi lain yang lebih baru, nomor versi Aurora tidak digunakan. Untuk mengidentifikasi nomor versi basis data Aurora PostgreSQL Anda, lihat. Mengidentifikasi versi Amazon Aurora PostgreSQL

Lihat informasi tentang ekstensi dan modul di Versi ekstensi untuk Amazon Aurora PostgreSQL.

> **Note**
>
> Untuk informasi tentang kebijakandan lini masa rilis versi Amazon Aurora, lihat Berapa lama versi mayor Amazon Aurora tetap tersedia.

Lihat informasi tentang dukungan untuk Amazon Aurora di FAQ Amazon RDS.

Untuk menentukan versi mesin basis data Aurora PostgreSQL yang tersedia di sebuah Wilayah AWS, gunakan perintah AWS CLI describe-db-engine-versions. Sebagai contoh:

```
aws rds describe-db-engine-versions --engine aurora-postgresql --query '*[].
[EngineVersion]' --output text --region aws-region
```

Untuk daftar Wilayah AWS, lihat Ketersediaan Wilayah Aurora PostgreSQL.

Untuk detail tentang versi PostgreSQL yang tersedia di Aurora PostgreSQL, lihat Catatan Rilis untuk Aurora PostgreSQL.

Versi ekstensi untuk Amazon Aurora PostgreSQL

Anda dapat menginstal dan mengonfigurasi berbagai ekstensi PostgreSQL untuk digunakan dengan klaster DB Aurora PostgreSQL. Misalnya, Anda dapat menggunakan ekstensi `pg_partman` PostgreSQL untuk mengotomatisasi pembuatan dan pemeliharaan partisi tabel. Untuk mempelajari selengkapnya tentang ekstensi ini dan ekstensi lain yang tersedia untuk Aurora PostgreSQL, lihat Menangani ekstensi dan pembungkus data asing.

Untuk detail tentang ekstensi PostgreSQL yang didukung di Aurora PostgreSQL, lihat Versi ekstensi untuk Amazon Aurora PostgreSQL dalam Catatan Rilis untuk Aurora PostgreSQL.

Meningkatkan klaster DB Amazon Aurora PostgreSQL

Amazon Aurora membuat mesin basis data PostgreSQL versi baru tersedia di Wilayah AWS hanya setelah pengujian ekstensif. Anda dapat meningkatkan klaster DB Aurora PostgreSQL ke versi baru jika tersedia di Wilayah Anda.

Bergantung pada versi Aurora PostgreSQL yang sedang dijalankan oleh klaster DB Anda, tingkatkan ke rilis baru berupa peningkatan minor atau peningkatan mayor. Misalnya, tingkatkan klaster DB Aurora PostgreSQL 11.15 ke Aurora PostgreSQL 13.6 berupa peningkatan versi mayor. Peningkatan klaster DB Aurora PostgreSQL 13.3 ke Aurora PostgreSQL 13.7 berupa peningkatan versi minor. Pada topik berikut, Anda dapat menemukan informasi tentang cara melakukan kedua jenis peningkatan.

Daftar Isi

- Ikhtisar proses peningkatan PostgreSQL Aurora
- Mendapatkan daftar versi yang tersedia di Wilayah AWS
- Cara melakukan peningkatan versi mayor

- Menguji peningkatan klaster DB produksi ke versi mayor baru
- Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru
  - Peningkatan mayor untuk basis data global
- Sebelum melakukan upgrade versi minor
- Cara melakukan peningkatan versi minor dan menerapkan patch
  - Peningkatan rilis minor dan patching nol-waktu henti
  - Meningkatkan mesin Aurora PostgreSQL ke versi minor baru
- Meningkatkan ekstensi PostgreSQL
- Teknik peningkatan blue/green alternatif

Ikhtisar proses peningkatan PostgreSQL Aurora

Berikut adalah perbedaan antara peningkatan versi mayor dan minor:

Peningkatan dan patch versi minor

Peningkatan dan patch versi minor hanya mencakup perubahan yang kompatibel mundur dengan aplikasi yang ada. Peningkatan dan patch versi minor akan tersedia bagi Anda hanya setelah Aurora PostgreSQL menguji dan menyetujuinya.

Peningkatan versi minor dapat diterapkan untuk Anda secara otomatis oleh Aurora. Jika Anda membuat klaster DB Aurora PostgreSQL baru, berarti opsi Aktifkan peningkatan versi minor telah dipilih sebelumnya. Kecuali Anda mematikan opsi ini, tingkatkan versi minor diterapkan secara otomatis selama periode pemeliharaan terjadwal Anda. Untuk informasi selengkapnya tentang opsi peningkatan versi minor otomatis (AmVU) dan cara mengubah klaster DB Aurora untuk menggunakannya, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora.

Jika opsi peningkatan versi minor otomatis tidak diatur untuk klaster DB Aurora PostgreSQL, Aurora PostgreSQL Anda tidak akan ditingkatkan ke versi minor baru secara otomatis. Sebaliknya, ketika versi minor baru dirilis di cluster DB PostgreSQL Anda Wilayah AWS dan Aurora Anda menjalankan versi minor yang lebih lama, Aurora meminta Anda untuk meningkatkan. Ini dilakukan dengan menambahkan rekomendasi ke tugas pemeliharaan untuk klaster Anda.

Patch tidak dianggap sebagai peningkatan, dan tidak diterapkan secara otomatis. Aurora PostgreSQL meminta Anda untuk menerapkan patch apa pun dengan menambahkan

rekomendasi ke tugas pemeliharaan untuk klaster DB Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Cara melakukan peningkatan versi minor dan menerapkan patch.

> **Note**
>
> Patch yang menyelesaikan masalah keamanan atau masalah penting lainnya juga ditambahkan sebagai tugas pemeliharaan. Namun, patch ini diperlukan. Pastikan untuk menerapkan patch keamanan ke klaster DB Aurora PostgreSQL Anda jika tersedia dalam tugas pemeliharaan Anda yang tertunda.

Proses peningkatan melibatkan kemungkinan pemadaman singkat karena setiap instans dalam klaster ditingkatkan ke versi baru. Namun, setelah Aurora PostgreSQL versi 14.3.3, 13.7.3, 12.11.3, 11.16.3, 10.21.3 dan rilis lain yang lebih tinggi dari versi minor ini dan versi mayor yang lebih baru, proses peningkatan menggunakan fitur patching nol-waktu henti (ZDP). Fitur ini meminimalkan pemadaman, dan dalam banyak kasus sepenuhnya menghilangkannya. Untuk informasi selengkapnya, lihat Peningkatan rilis minor dan patching nol-waktu henti.

> **Note**
>
> ZDP tidak didukung dalam kasus berikut:
>
> - Saat klaster DB Aurora PostgreSQL dikonfigurasi sebagai Aurora Serverless v1.
> - Ketika Aurora PostgreSQL DB cluster dikonfigurasi sebagai database global Aurora di sekunder. Wilayah AWS
> - Selama peningkatan instans pembaca di basis data global Aurora.
> - Selama patch OS dan peningkatan OS.
>
> ZDP didukung untuk cluster Aurora PostgreSQL DB yang dikonfigurasi sebagai. Aurora Serverless v2

Peningkatan versi mayor

Tidak seperti peningkatan dan patch versi minor, Aurora PostgreSQL tidak memiliki opsi peningkatan versi mayor utama otomatis. Versi PostgreSQL mayor baru mungkin berisi perubahan basis data yang tidak kompatibel mundur dengan aplikasi yang ada. Fungsionalitas baru dapat menyebabkan aplikasi yang ada tidak berfungsi dengan benar.

Untuk mencegah masalah apa pun, sebaiknya Anda mengikuti proses yang diuraikan dalam Menguji peningkatan klaster DB produksi ke versi mayor baru sebelum meningkatkan instans DB di klaster DB Aurora PostgreSQL Anda. Pertama pastikan aplikasi Anda dapat berjalan di versi baru dengan mengikuti prosedur tersebut. Kemudian, Anda dapat meningkatkan klaster DB Aurora PostgreSQL ke versi baru secara manual.

Proses upgrade melibatkan kemungkinan pemadaman singkat ketika semua instance di cluster ditingkatkan ke versi baru. Proses perencanaan awal juga membutuhkan waktu. Sebaiknya Anda selalu melakukan tugas peningkatan selama periode pemeliharaan klaster atau saat pengoperasiannya minimum. Untuk informasi selengkapnya, lihat Cara melakukan peningkatan versi mayor.

> **Note**
>
> Peningkatan versi minor dan peningkatan versi mayor mungkin melibatkan pemadaman singkat. Oleh karena itu, sebaiknya Anda melakukan atau menjadwalkan peningkatan selama periode pemeliharaan atau selama periode penggunaan rendah lainnya.

Klaster DB Aurora PostgreSQL terkadang memerlukan pembaruan sistem operasi. Pembaruan ini mungkin menyertakan pustaka glibc versi yang lebih baru. Selama pembaruan tersebut, sebaiknya Anda mengikuti pedoman seperti yang dijelaskan dalam Kolasi yang didukung di Aurora PostgreSQL.

Mendapatkan daftar versi yang tersedia di Wilayah AWS

Anda bisa mendapatkan daftar semua versi mesin yang tersedia sebagai target peningkatan untuk cluster Aurora PostgreSQL DB Anda dengan menanyakan Anda menggunakan perintah, sebagai berikut. Wilayah AWS describe-db-engine-versions AWS CLI

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version version-number \
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

Untuk Windows:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

Misalnya, untuk mengidentifikasi target pemutakhiran yang valid untuk klaster Aurora PostgreSQL versi 12.10 DB, jalankan perintah berikut: AWS CLI

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version 12.10 \
  --query 'DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}' \
  --output text
```

Untuk Windows:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version 12.10 ^
  --query "DBEngineVersions[*].ValidUpgradeTarget[*].{EngineVersion:EngineVersion}" ^
  --output text
```

Dalam tabel ini, Anda dapat menemukan target peningkatan versi mayor dan minor yang tersedia untuk berbagai versi DB Aurora PostgreSQL.

Versi sumbe saat ini	Target peningkatan
15.5	16
15.4	16 15
15.3	16 15 15
15.2	16 15 15 15

Versi sumbe saat ini	Target peningkatan
14.10	16 15
14.9	16 15 15 14
14.8	16 15 15 15 15 14 14
14.7	16 15 15 15 15 14 14 14
14.6	16 15 15 15 15 14 14 14 14
14.5	16 15 15 15 15 14 14 14 14 14
14.4	16 15 15 15 15 14 14 14 14 14 14
14.3	16 15 15 15 15 14 14 14 14 14 14 14
13.13	16 15 14
13.12	16 15 15 14 14
13.11	16 15 15 15 14 14 14
13.10	16 15 15 15 15 14 14 14 14 13 13 13
13.9	16 15 15 15 15 14 14 14 14 14 13 13
13.8	16 15 15 15 15 14 14 14 14 14 14 13 13 13 13 13
13.7	16 15 15 15 15 14 14 14 14 14 14 14 14 13 13 13 13 13 13
12.17	16 15 14 13
12.16	16 15 15 14 14 13 13
12.15	16 15 15 15 14 14 14 13 13 13
12.14	16 15 15 15 15 14 14 14 14 13 13 13 13 12

Versi sumbe saat ini	Target peningkatan
12.13	16 15 15 15 15 14 14 14 14 14 13 13 13 13 13 12 12 12 12
12.12	16 15 15 15 15 14 14 14 14 14 14 13 13 13 13 13 12 12 12 13 12 12 12
12.11	16 15 15 15 15 14 14 14 14 14 14 14 13 13 13 13 13 13 13 12 12 12 12
12.9	16 15 15 15 15 14 14 14 14 13 13 13 13 13 13 13 12 12 12 12 12 12 12
11.21	16 15 15 14 14 13 13 12 12
11.9	16 15 15 15 15 14 14 14 14 14 13 13 13 13 13 12 12 12 12 12 12 12 12 11.21

Untuk versi apa pun yang Anda pertimbangkan, selalu periksa ketersediaan kelas instans DB klaster Anda. Misalnya, `db.r4` tidak didukung untuk Aurora PostgreSQL 13. Jika klaster DB Aurora PostgreSQL Anda saat ini menggunakan kelas instans db.r4, Anda harus migrasi ke db.r5 sebelum mencoba peningkatan. Untuk informasi selengkapnya tentang kelas instans DB, termasuk kelas yang berbasis Graviton2 dan kelas yang berbasis Intel, lihat Kelas instans DB Aurora.

Cara melakukan peningkatan versi mayor

Peningkatan versi mayor mungkin berisi perubahan basis data yang tidak kompatibel mundur dengan basis data versi sebelumnya. Fungsionalitas baru dalam versi baru dapat menyebabkan aplikasi yang ada tidak berfungsi dengan benar. Untuk menghindari masalah, Amazon Aurora tidak menerapkan peningkatan versi mayor secara otomatis. Sebaliknya, sebaiknya Anda merencanakan peningkatan versi utama secara cermat dengan mengikuti langkah-langkah berikut:

1. Pilih versi mayor yang diinginkan dari daftar target yang tersedia dari yang tercantum untuk versi Anda dalam tabel. Anda bisa mendapatkan daftar versi yang tepat yang tersedia di versi Anda saat ini dengan menggunakan AWS CLI. Wilayah AWS Untuk detailnya, lihat Mendapatkan daftar versi yang tersedia di Wilayah AWS.
2. Verifikasi apakah aplikasi Anda berfungsi seperti yang diharapkan saat deployment uji coba versi baru. Untuk informasi tentang proses lengkapnya, lihat Menguji peningkatan klaster DB produksi ke versi mayor baru.

3. Setelah memverifikasi bahwa aplikasi Anda berfungsi seperti yang diharapkan saat deployment uji coba, Anda dapat meningkatkan klaster. Untuk detailnya, lihat Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru.

> **Note**
>
> Anda dapat melakukan peningkatan versi mayor dari versi berbasis Babelfish for Aurora PostgreSQL 13 mulai dari versi 13.6 ke versi berbasis Aurora PostgreSQL 14 mulai dari versi 14.6. Babelfish for Aurora PostgreSQL 13.4 dan 13.5 tidak mendukung peningkatan versi mayor.

Anda bisa mendapatkan daftar versi mesin yang tersedia sebagai target peningkatan versi utama untuk cluster DB PostgreSQL Aurora Anda dengan menanyakan Anda menggunakan perintah, sebagai berikut. Wilayah AWS describe-db-engine-versions AWS CLI

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-engine-versions \
  --engine aurora-postgresql \
  --engine-version version-number \
  --query 'DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}' \
  --output text
```

Untuk Windows:

```
aws rds describe-db-engine-versions ^
  --engine aurora-postgresql ^
  --engine-version version-number ^
  --query "DBEngineVersions[].ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`].
{EngineVersion:EngineVersion}" ^
  --output text
```

Dalam beberapa kasus, versi yang ingin ditingkatkan bukanlah target untuk versi Anda saat ini. Dalam kasus seperti itu, gunakan informasi dalam versions table untuk melakukan peningkatan versi minor hingga klaster Anda memiliki versi dengan target yang Anda pilih di baris targetnya.

Menguji peningkatan klaster DB produksi ke versi mayor baru

Setiap versi mayor baru mencakup penyempurnaan pada pengoptimal kueri yang dirancang untuk meningkatkan performa. Namun, beban kerja Anda mungkin mencakup kueri yang menyebabkan rencana berperforma lebih buruk di versi baru. Itulah mengapa kami menyarankan Anda menguji dan meninjau performa sebelum melakukan peningkatan dalam produksi. Anda dapat mengelola stabilitas rencana kueri di seluruh versi dengan menggunakan ekstensi Manajemen Rencana Kueri (QPM), seperti yang dijelaskan dalam Memastikan stabilitas rencana setelah upgrade versi mayor.

Sebelum meningkatkan klaster DB Aurora PostgreSQL produksi ke versi mayor baru, sebaiknya Anda menguji peningkatan untuk memverifikasi bahwa semua aplikasi Anda berfungsi dengan benar:

1. Siapkan grup parameter yang kompatibel dengan versi.

   Jika menggunakan instans DB kustom atau grup parameter klaster DB, Anda dapat memilih dari dua opsi:

   a. Tentukan instans DB default, grup parameter klaster DB, atau keduanya untuk versi mesin DB baru.

   b. Buat grup parameter kustom untuk versi mesin DB baru.

   Jika Anda menautkan instans DB baru atau grup parameter klaster DB sebagai bagian dari permintaan peningkatan, pastikan untuk mem-boot ulang basis data setelah proses peningkatan selesai untuk menerapkan parameter. Jika instans DB perlu di-boot ulang untuk menerapkan perubahan grup parameter, status grup parameter instans akan menampilkan `pending-reboot`. Anda dapat melihat status grup parameter instance di konsol atau dengan menggunakan perintah CLI seperti describe-db-instancesatau. describe-db-clusters
2. Periksa penggunaan yang tidak didukung:
   - Lakukan atau putar kembali semua transaksi terbuka yang disiapkan sebelum mencoba peningkatan. Anda dapat menggunakan kueri berikut untuk memverifikasi bahwa tidak ada transaksi terbuka yang disiapkan di instans Anda.

     ```
     SELECT count(*) FROM pg_catalog.pg_prepared_xacts;
     ```

   - Hapus semua penggunaan jenis data reg* sebelum mencoba peningkatan. Kecuali untuk `regtype` dan `regclass`, Anda tidak dapat meningkatkan jenis data reg*. Utilitas pg_upgrade (yang digunakan oleh Amazon Aurora untuk melakukan peningkatan) tidak dapat mempertahankan jenis data ini. Untuk mempelajari selengkapnya tentang utilitas ini, lihat pg_upgrade dalam dokumentasi PostgreSQL.

Untuk memverifikasi bahwa jenis data reg* yang tidak didukung tidak digunakan, gunakan kueri berikut untuk setiap basis data.

```
SELECT count(*) FROM pg_catalog.pg_class c, pg_catalog.pg_namespace n,
 pg_catalog.pg_attribute a
  WHERE c.oid = a.attrelid
      AND NOT a.attisdropped
      AND a.atttypid IN ('pg_catalog.regproc'::pg_catalog.regtype,
                         'pg_catalog.regprocedure'::pg_catalog.regtype,
                         'pg_catalog.regoper'::pg_catalog.regtype,
                         'pg_catalog.regoperator'::pg_catalog.regtype,
                         'pg_catalog.regconfig'::pg_catalog.regtype,
                         'pg_catalog.regdictionary'::pg_catalog.regtype)
      AND c.relnamespace = n.oid
      AND n.nspname NOT IN ('pg_catalog', 'information_schema');
```

- Jika Anda meningkatkan Aurora PostgreSQL versi 10.18 atau klaster DB yang lebih tinggi yang memiliki `pgRouting` ekstensi terinstal, lepaskan ekstensi sebelum ditingkatkan ke versi 12.4 atau lebih tinggi.

  Jika Anda meningkatkan Aurora PostgreSQL versi 10.x yang memiliki ekstensi `pg_repack` versi 1.4.3 terinstal, lepaskan ekstensi sebelum ditingkatkan ke versi yang lebih tinggi.

3. Periksa basis data template1 dan template0.

   Agar peningkatan berhasil, basis data template 1 dan template 0 harus ada dan harus tercantum sebagai template. Untuk melakukannya, gunakan perintah berikut:

```
SELECT datname, datistemplate FROM pg_database;

datname    | datistemplate
-----------+---------------
template0  | t
rdsadmin   | f
template1  | t
postgres   | f
```

   Dalam output perintah, nilai `datistemplate` untuk basis data template1 dan template0 seharusnya `t`.

4. Hapus slot replikasi logis.

Proses peningkatan tidak dapat dilanjutkan jika klaster DB Aurora PostgreSQL menggunakan slot replikasi logis apa pun. Slot replikasi logis biasanya digunakan untuk tugas migrasi data jangka pendek, seperti migrasi data menggunakan AWS DMS atau untuk mereplikasi tabel dari database ke danau data, alat BI, atau target lainnya. Sebelum meningkatkan, pastikan Anda mengetahui tujuan dari setiap slot replikasi logis yang ada, dan konfirmasikan bahwa menghapusnya tidak akan jadi masalah. Anda dapat memeriksa slot replikasi logis menggunakan kueri berikut:

```
SELECT * FROM pg_replication_slots;
```

Jika slot replikasi logis masih digunakan, Anda tidak boleh menghapusnya, dan Anda tidak dapat melanjutkan dengan peningkatan. Namun, jika slot replikasi logis tidak diperlukan, Anda dapat menghapusnya menggunakan SQL berikut:

```
SELECT pg_drop_replication_slot(slot_name);
```

Skenario replikasi logis yang menggunakan ekstensi `pglogical` juga harus memiliki slot yang dijatuhkan dari simpul penerbit untuk peningkatan versi utama yang berhasil pada simpul tersebut. Namun, Anda dapat memulai ulang proses replikasi dari simpul pelanggan setelah peningkatan. Untuk informasi selengkapnya, lihat Membangun kembali replikasi logis setelah peningkatan besar.

5. Lakukan pencadangan.

   Proses peningkatan membuat snapshot klaster DB dari klaster DB Anda selama peningkatan. Jika Anda juga ingin melakukan pencadangan manual sebelum proses peningkatan, lihat Membuat snapshot klaster DB untuk informasi selengkapnya.

6. Tingkatkan ekstensi tertentu ke versi terbaru yang tersedia sebelum melakukan peningkatan versi utama. Ekstensi yang akan diperbarui meliputi:

   - `pgRouting`
   - `postgis_raster`
   - `postgis_tiger_geocoder`
   - `postgis_topology`
   - `address_standardizer`
   - `address_standardizer_data_us`

   Jalankan perintah berikut untuk setiap ekstensi yang baru diinstal.

```
ALTER EXTENSION PostgreSQL-extension UPDATE TO 'new-version';
```

Untuk informasi selengkapnya, lihat Meningkatkan ekstensi PostgreSQL. Untuk mempelajari selengkapnya tentang cara meningkatkan PostGIS, lihat Langkah 6: Meningkatkan ekstensi PostGIS.

7. Jika Anda meningkatkan ke versi 11.x, buang ekstensi yang tidak didukung sebelum melakukan peningkatan versi utama. Ekstensi yang perlu dibuang meliputi:
   - `chkpass`
   - `tsearch2`
8. Buang jenis data `unknown`, bergantung pada versi target Anda.

   PostgreSQL versi 10 tidak mendukung jenis data `unknown`. Jika basis data versi 9.6 menggunakan jenis data `unknown`, tingkatkan ke versi 10 akan menampilkan pesan kesalahan seperti berikut.

   ```
   Database instance is in a state that cannot be upgraded: PreUpgrade checks failed:
   The instance could not be upgraded because the 'unknown' data type is used in user
    tables.
   Please remove all usages of the 'unknown' data type and try again."
   ```

   Untuk menemukan jenis data `unknown` di basis data Anda sehingga Anda dapat menghapus kolom tersebut atau mengubahnya ke jenis data yang didukung, gunakan kode SQL berikut untuk setiap basis data.

   ```
   SELECT n.nspname, c.relname, a.attname
       FROM pg_catalog.pg_class c,
       pg_catalog.pg_namespace n,
       pg_catalog.pg_attribute a
       WHERE c.oid = a.attrelid AND NOT a.attisdropped AND
       a.atttypid = 'pg_catalog.unknown'::pg_catalog.regtype AND
       c.relkind IN ('r','m','c') AND
       c.relnamespace = n.oid AND
       n.nspname !~ '^pg_temp_' AND
       n.nspname !~ '^pg_toast_temp_' AND n.nspname NOT IN ('pg_catalog',
    'information_schema');
   ```

9. Lakukan peningkatan dry-run.

Sebaiknya Anda menguji peningkatan versi utama pada duplikat basis data produksi sebelum mencoba peningkatan pada basis data produksi Anda. Anda dapat memantau rencana eksekusi pada instans pengujian duplikat untuk setiap kemungkinan regresi rencana eksekusi dan untuk mengevaluasi performanya. Untuk membuat instans pengujian duplikat, Anda dapat memulihkan basis data Anda dari snapshot terakhir atau mengkloning basis data Anda. Untuk informasi selengkapnya, lihat Memulihkan dari snapshot atau Mengkloning volume untuk klaster DB Amazon Aurora.

Untuk informasi selengkapnya, lihat Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru.

10. Tingkatkan instans produksi Anda.

    Jika peningkatan versi utama operasi dry-run berhasil, Anda seharusnya dapat meningkatkan basis data produksi dengan percaya diri. Untuk informasi selengkapnya, lihat Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru.

    > **Note**
    >
    > Selama proses peningkatan, Aurora PostgreSQL mengambil snapshot klaster DB jika periode retensi cadangan klaster lebih besar dari 0. Anda tidak dapat melakukan point-in-time pemulihan cluster Anda selama proses ini. Kemudian, Anda dapat melakukan point-in-time pemulihan ke waktu sebelum pemutakhiran dimulai dan setelah snapshot otomatis instans Anda selesai. Namun, Anda tidak dapat melakukan point-in-time pemulihan ke versi minor sebelumnya.

    Untuk informasi tentang peningkatan yang sedang berlangsung, Anda dapat menggunakan Amazon RDS untuk melihat dua log yang dibuat oleh utilitas pg_upgrade. Log tersebut adalah `pg_upgrade_internal.log` dan `pg_upgrade_server.log`. Amazon Aurora menambahkan stempel waktu ke nama file untuk log ini. Anda dapat melihat log ini sebagaimana Anda dapat melihat log lainnya. Untuk informasi selengkapnya, lihat Memantau file log Amazon Aurora.

11. Meningkatkan ekstensi PostgreSQL. Proses peningkatan PostgreSQL tidak meningkatkan ekstensi PostgreSQL apa pun. Untuk informasi selengkapnya, lihat Meningkatkan ekstensi PostgreSQL.

Setelah Anda menyelesaikan peningkatan versi utama, sebaiknya ikuti langkah berikut:

- Jalankan operasi ANALYZE untuk menyegarkan tabel `pg_statistic`. Anda harus melakukannya untuk setiap basis data pada semua instans DB PostgreSQL Anda. Statistik pengoptimal tidak ditransfer selama peningkatan versi utama, jadi Anda perlu membuat ulang semua statistik untuk menghindari masalah performa. Jalankan perintah tanpa parameter apa pun untuk menghasilkan statistik untuk semua tabel reguler dalam basis data saat ini, sebagai berikut:

  ```
  ANALYZE VERBOSE;
  ```

  Bendera VERBOSE bersifat opsional, tetapi kemajuannya akan ditampilkan jika digunakan. Untuk informasi selengkapnya, lihat ANALYZE di dokumentasi PostgreSQL.

  > **Note**
  >
  > Jalankan ANALYZE pada sistem Anda setelah peningkatan untuk menghindari masalah performa.

- Jika Anda melakukan peningkatan ke PostgreSQL versi 10, jalankan `REINDEX` pada setiap indeks hash yang dimiliki. Indeks hash diubah di versi 10 dan harus dibuat ulang. Untuk menemukan indeks hash yang tidak valid, jalankan SQL berikut untuk setiap basis data yang berisi indeks hash.

  ```
  SELECT idx.indrelid::regclass AS table_name,
     idx.indexrelid::regclass AS index_name
  FROM pg_catalog.pg_index idx
     JOIN pg_catalog.pg_class cls ON cls.oid = idx.indexrelid
     JOIN pg_catalog.pg_am am ON am.oid = cls.relam
  WHERE am.amname = 'hash'
  AND NOT idx.indisvalid;
  ```

- Sebaiknya Anda menguji aplikasi di basis data yang ditingkatkan dengan beban kerja serupa untuk memverifikasi bahwa semuanya berfungsi sesuai harapan. Setelah peningkatan diverifikasi, Anda dapat menghapus instans pengujian ini.

Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru

Saat Anda memulai proses peningkatan ke versi mayor baru, Aurora PostgreSQL mengambil snapshot dari klaster DB Aurora sebelum membuat perubahan apa pun pada klaster Anda. Snapshot ini dibuat hanya untuk peningkatan versi mayor, bukan peningkatan versi minor. Saat proses peningkatan selesai, Anda dapat menemukan snapshot ini di antara snapshot manual yang tercantum di bagian Snapshot di konsol RDS. Nama snapshot mencakup `preupgrade` sebagai

awalan, nama klaster DB Aurora PostgreSQL Anda, versi sumber, versi target, dan tanggal serta stempel waktu, seperti yang ditunjukkan pada contoh berikut.

```
preupgrade-docs-lab-apg-global-db-12-8-to-13-6-2022-05-19-00-19
```

Setelah peningkatan selesai, Anda dapat menggunakan snapshot yang dibuat dan disimpan Aurora dalam daftar snapshot manual untuk memulihkan klaster DB ke versi sebelumnya, jika perlu.

> **Tip**
>
> Secara umum, snapshot menyediakan banyak cara untuk memulihkan klaster DB Aurora ke berbagai titik waktu. Untuk mempelajari lebih lanjut, lihat Memulihkan dari snapshot klaster DB dan Memulihkan klaster DB ke waktu tertentu. Namun, Aurora PostgreSQL tidak mendukung penggunaan snapshot untuk memulihkan ke versi minor sebelumnya.

Selama proses peningkatan versi mayor, Aurora mengalokasikan volume dan mengkloning klaster DB Aurora PostgreSQL sumber. Jika peningkatan gagal karena alasan apa pun, Aurora PostgreSQL menggunakan klon ini untuk memutar kembali peningkatan. Setelah lebih dari 15 klon dari volume sumber dialokasikan, klon berikutnya menjadi salinan lengkap dan memakan waktu lebih lama. Hal ini juga dapat menyebabkan proses peningkatan memakan waktu lebih lama. Jika Aurora PostgreSQL membatalkan peningkatan, perhatikan hal berikut:

- Anda dapat melihat entri dan metrik penagihan untuk volume awal dan volume kloning yang dialokasikan selama peningkatan. Aurora PostgreSQL membersihkan volume ekstra setelah periode retensi cadangan klaster melampaui waktu peningkatan .
- Salinan snapshot lintas-wilayah berikutnya dari klaster ini akan menjadi salinan lengkap, bukan salinan tambahan.

Untuk meningkatkan instans DB yang membentuk klaster Anda dengan aman, Aurora PostgreSQL menggunakan utilitas pg_upgrade. Setelah peningkatan penulis selesai, setiap instans pembaca mengalami pemadaman singkat saat ditingkatkan ke versi mayor baru. Untuk mempelajari selengkapnya tentang utilitas PostgreSQL ini, lihat pg_upgrade di dokumentasi PostgreSQL.

Anda dapat memutakhirkan cluster Aurora PostgreSQL DB Anda ke versi baru dengan menggunakan, API, atau RDS. AWS Management Console AWS CLI

Konsol

Untuk meningkatkan versi mesin klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih klaster DB yang ingin Anda tingkatkan.
3. Pilih Ubah. Halaman Ubah klaster DB akan muncul.
4. Untuk Versi mesin, pilih versi baru.
5. Pilih Lanjutkan dan periksa ringkasan perubahan.
6. Untuk segera menerapkan perubahan, pilih Terapkan segera. Dalam beberapa kasus, pemadaman dapat terjadi jika opsi ini dipilih. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora.
7. Di halaman konfirmasi, tinjau perubahan Anda. Jika sudah benar, pilih Ubah Klaster untuk menyimpan perubahan.

   Atau pilih Kembali untuk mengedit perubahan atau Batal untuk membatalkan perubahan.

AWS CLI

Untuk memutakhirkan versi mesin cluster DB, gunakan modify-db-cluster AWS CLI perintah. Tentukan parameter berikut:

- `--db-cluster-identifier` – Nama klaster DB.
- `--engine-version` – Nomor versi peningkatan mesin basis data. Untuk informasi tentang versi mesin yang valid, gunakan AWS CLI describe-db-engine-versionsperintah.
- `--allow-major-version-upgrade` – Tanda yang diperlukan saat parameter `--engine-version` merupakan versi utama yang berbeda dengan versi utama saat ini dari klaster DB.
- `--no-apply-immediately` – Terapkan perubahan selama periode pemeliharaan berikutnya. Untuk segera menerapkan perubahan, gunakan `--apply-immediately`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
```

```
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --allow-major-version-upgrade \
    --no-apply-immediately
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine-version new_version ^
    --allow-major-version-upgrade ^
    --no-apply-immediately
```

RDS API

Untuk meningkatkan versi mesin klaster DB, gunakan operasi ModifyDBCluster. Tentukan parameter berikut:

- `DBClusterIdentifier` – Nama klaster DB, misalnya *`mydbcluster`*.
- `EngineVersion` – Nomor versi peningkatan mesin basis data. Untuk informasi tentang versi mesin yang valid, gunakan operasi DescribeDB EngineVersions.
- `AllowMajorVersionUpgrade` – Tanda yang diperlukan saat parameter `EngineVersion` merupakan versi utama yang berbeda dengan versi utama saat ini dari klaster DB.
- `ApplyImmediately` – Apakah akan menerapkan perubahan secara langsung atau selama periode pemeliharaan berikutnya. Untuk segera menerapkan perubahan, tetapkan nilai ke `true`. Untuk menerapkan perubahan selama periode pemeliharaan berikutnya, tetapkan nilai ke `false`.

Peningkatan mayor untuk basis data global

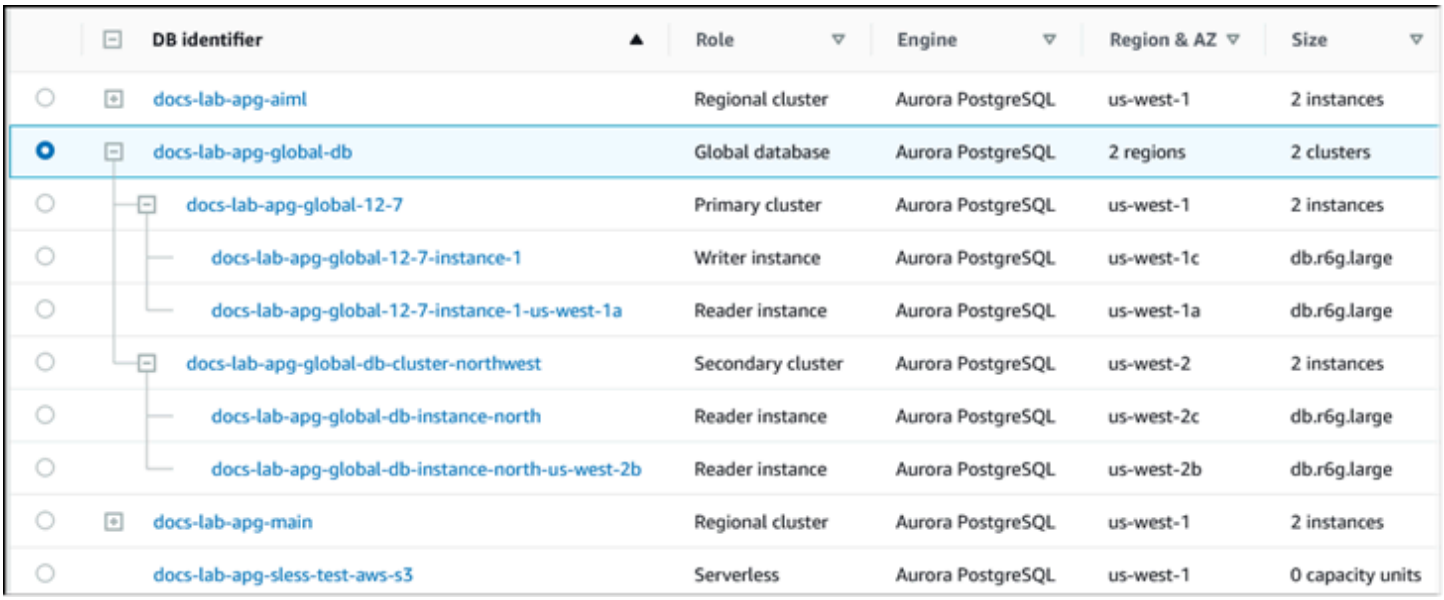
Untuk klaster basis data global Aurora, proses peningkatan akan meningkatkan semua klaster DB yang membentuk basis data global Aurora pada saat yang bersamaan. Itu dilakukan untuk memastikan bahwa masing-masing menjalankan versi Aurora PostgreSQL yang sama. Ini juga memastikan bahwa setiap perubahan pada tabel sistem, format file data, dan sebagainya secara otomatis direplikasi ke semua klaster sekunder.

Untuk meningkatkan klaster basis data global ke Aurora PostgreSQL versi mayor baru, sebaiknya Anda menguji aplikasi Anda pada versi yang ditingkatkan, seperti yang dijelaskan dalam Menguji peningkatan klaster DB produksi ke versi mayor baru. Pastikan untuk menyiapkan grup parameter

cluster DB dan pengaturan grup parameter DB Anda untuk masing-masing Wilayah AWS di database global Aurora Anda sebelum peningkatan sebagaimana dirinci. step 1. Menguji peningkatan klaster DB produksi ke versi mayor baru

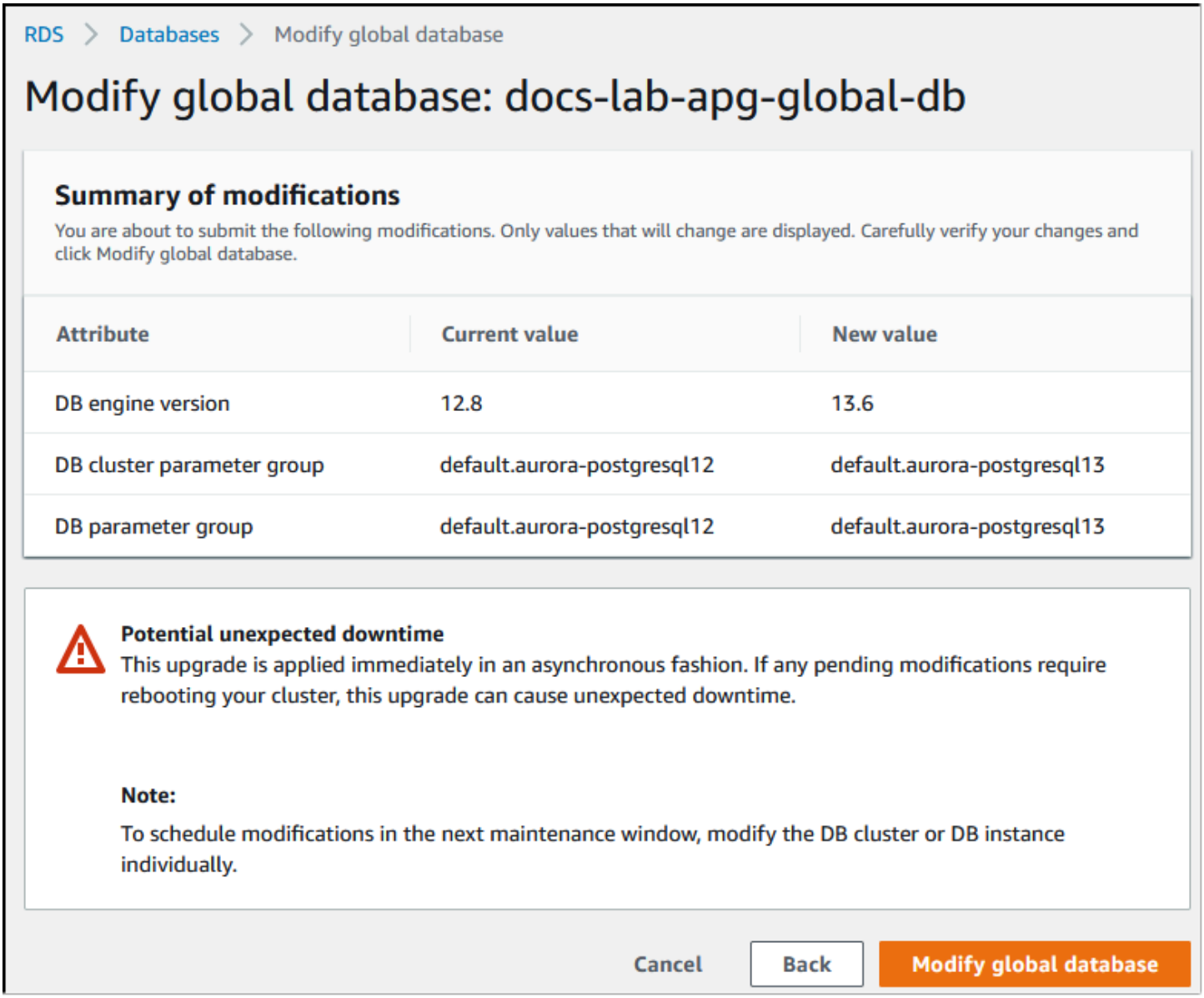
Jika klaster basis data global Aurora PostgreSQL Anda memiliki sasaran titik pemulihan (RPO) yang ditetapkan untuk parameter `rds.global_db_rpo`-nya, pastikan untuk mengatur ulang parameter sebelum melakukan peningkatan. Proses peningkatan versi mayor tidak berfungsi jika RPO diaktifkan. Secara default, parameter ini dinonaktifkan. Untuk informasi lebih lanjut tentang basis data global Aurora PostgreSQL dan RPO, lihat Mengelola RPO untuk basis data global berbasis Aurora PostgreSQL.

Jika Anda memverifikasi bahwa aplikasi Anda dapat berjalan sesuai harapan saat deployment uji coba versi baru, Anda dapat memulai proses peningkatan. Untuk melakukannya, lihat Meningkatkan mesin Aurora PostgreSQL ke versi mayor baru. Pastikan untuk memilih item tingkat atas dari daftar Basis Data di konsol RDS, Basis data global, seperti yang ditunjukkan pada gambar berikut.



DB identifier	Role	Engine	Region & AZ	Size
docs-lab-apg-aiml	Regional cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-global-db	Global database	Aurora PostgreSQL	2 regions	2 clusters
docs-lab-apg-global-12-7	Primary cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-global-12-7-instance-1	Writer instance	Aurora PostgreSQL	us-west-1c	db.r6g.large
docs-lab-apg-global-12-7-instance-1-us-west-1a	Reader instance	Aurora PostgreSQL	us-west-1a	db.r6g.large
docs-lab-apg-global-db-cluster-northwest	Secondary cluster	Aurora PostgreSQL	us-west-2	2 instances
docs-lab-apg-global-db-instance-north	Reader instance	Aurora PostgreSQL	us-west-2c	db.r6g.large
docs-lab-apg-global-db-instance-north-us-west-2b	Reader instance	Aurora PostgreSQL	us-west-2b	db.r6g.large
docs-lab-apg-main	Regional cluster	Aurora PostgreSQL	us-west-1	2 instances
docs-lab-apg-sless-test-aws-s3	Serverless	Aurora PostgreSQL	us-west-1	0 capacity units

Seperti halnya modifikasi apa pun, Anda dapat mengonfirmasi bahwa Anda ingin melanjutkan proses saat diminta.



Alih-alih menggunakan konsol, Anda dapat memulai proses peningkatan dengan menggunakan AWS CLI atau RDS API. Seperti halnya konsol, Anda beroperasi pada klaster basis data global Aurora, bukan konstituennya, sebagai berikut:

- Gunakan modify-global-cluster AWS CLI perintah untuk memulai upgrade untuk database global Aurora Anda dengan menggunakan. AWS CLI
- Gunakan ModifyGlobalClusterAPI untuk memulai upgrade.

Sebelum melakukan upgrade versi minor

Kami menyarankan Anda melakukan tindakan berikut untuk mengurangi waktu henti selama upgrade versi minor:

- Pemeliharaan cluster Aurora DB harus dilakukan selama periode lalu lintas rendah. Gunakan Performance Insights untuk mengidentifikasi periode waktu ini agar dapat mengonfigurasi jendela pemeliharaan dengan benar. Untuk informasi selengkapnya tentang Performance Insights, lihat Memantau pemuatan DB dengan Performance Insights di Amazon RDS. Untuk informasi lebih lanjut tentang jendela pemeliharaan cluster DB,Menyesuaikan periode pemeliharaan klaster DB yang dinginkan.
- Gunakan AWS SDK yang mendukung backoff eksponensial dan jitter sebagai praktik terbaik. Untuk informasi lebih lanjut, lihat Exponential Backoff And Jitter.

Cara melakukan peningkatan versi minor dan menerapkan patch

Peningkatan dan tambalan versi minor Wilayah AWS hanya tersedia setelah pengujian yang ketat. Sebelum merilis peningkatan dan patch, Aurora PostgreSQL menguji untuk memastikan bahwa masalah keamanan yang diketahui, bug, dan masalah lain yang muncul setelah rilis versi komunitas minor tidak mengganggu stabilitas armada Aurora PostgreSQL secara keseluruhan.

Karena Aurora PostgreSQL membuat versi minor baru tersedia, instans yang membentuk klaster DB Aurora PostgreSQL Anda dapat ditingkatkan secara otomatis selama periode pemeliharaan yang Anda tentukan. Agar hal ini terjadi, opsi Aktifkan peningkatan versi minor otomatis pada klaster DB Aurora PostgreSQL Anda harus diaktifkan. Semua instans DB yang membentuk klaster DB Aurora PostgreSQL Anda harus mengaktifkan opsi peningkatan versi minor otomatis (AmVU) sehingga peningkatan minor diterapkan di seluruh klaster.

> **Tip**
>
> Pastikan opsi Aktifkan peningkatan versi minor otomatis diaktifkan untuk semua instans DB PostgreSQL yang membentuk klaster DB Aurora PostgreSQL. Opsi ini harus diaktifkan agar setiap instans di klaster DB berfungsi. Untuk informasi tentang cara mengatur Peningkatan versi minor otomatis, dan cara kerja pengaturan saat diterapkan di tingkat klaster dan instans, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora.

Anda dapat memeriksa nilai opsi Enable auto minor version upgrade untuk semua cluster Aurora PostgreSQL DB Anda dengan menggunakan perintah dengan kueri berikut. describe-db-instances AWS CLI

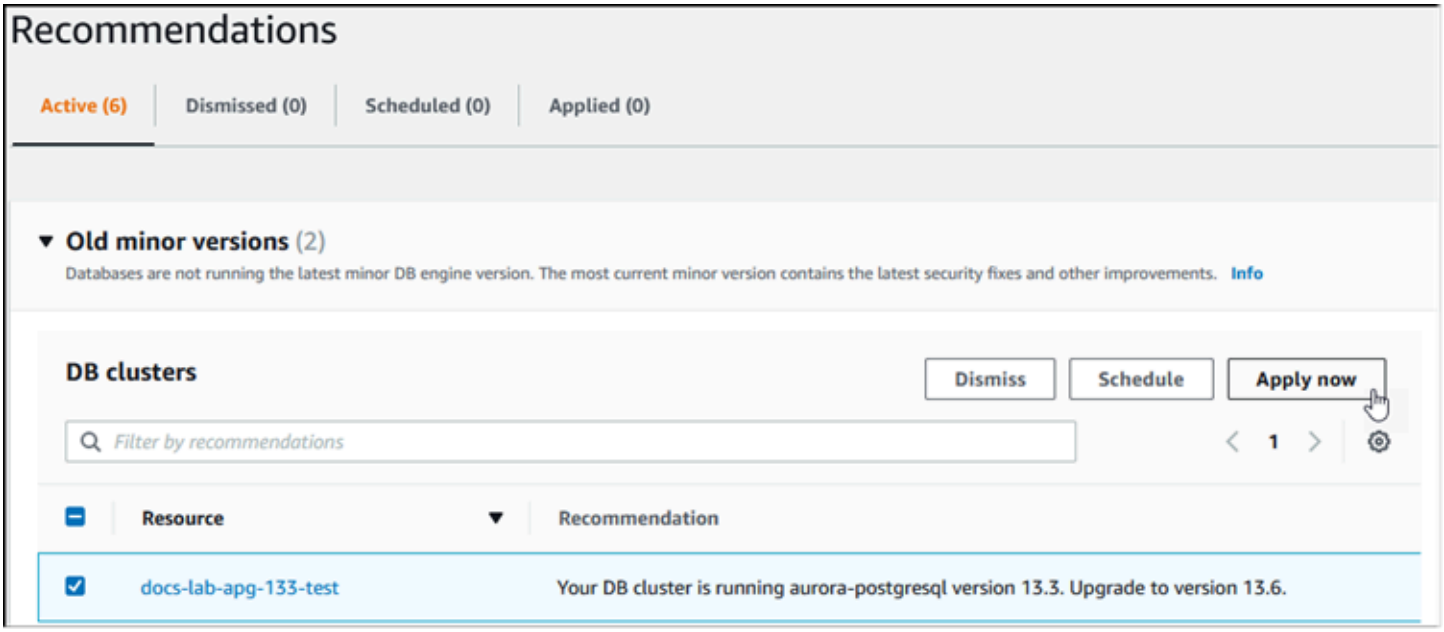
```
aws rds describe-db-instances \
  --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,DBInstanceIdentifier:DBInstanceIdentifier,AutoMinorVer
```

Kueri ini menampilkan daftar semua klaster DB Aurora dan instans--nya dengan nilai `false` atau `true` untuk status pengaturan `AutoMinorVersionUpgrade`. Perintah seperti yang ditunjukkan mengasumsikan bahwa Anda telah AWS CLI dikonfigurasi untuk menargetkan default Wilayah AWS Anda.

Untuk informasi selengkapnya tentang opsi AmVU dan cara memodifikasi klaster DB Aurora untuk menggunakannya, lihat Peningkatan versi minor otomatis untuk klaster DB Aurora.

Anda dapat meningkatkan klaster DB Aurora PostgreSQL ke versi minor baru baik dengan menanggapi tugas pemeliharaan, atau memodifikasi klaster untuk menggunakan versi baru.

Anda dapat mengidentifikasi peningkatan atau patch yang tersedia untuk klaster DB Aurora PostgreSQL dengan menggunakan konsol RDS dan membuka menu Rekomendasi. Di sana, Anda dapat menemukan daftar berbagai masalah pemeliharaan seperti Versi minor lama. Bergantung pada lingkungan produksi Anda, Anda dapat memilih untuk Menjadwalkan peningkatan atau mengambil tindakan segera, dengan memilih Terapkan sekarang, seperti yang ditunjukkan di bawah.



Untuk mempelajari lebih lanjut tentang cara memelihara klaster DB Aurora, termasuk cara menerapkan patch dan peningkatan versi minor secara manual, lihat Memelihara klaster DB Amazon Aurora.

Peningkatan rilis minor dan patching nol-waktu henti

Meningkatkan klaster DB Aurora PostgreSQL melibatkan kemungkinan pemadaman. Selama proses peningkatan, basis data dimatikan karena sedang ditingkatkan. Jika Anda memulai peningkatan saat basis data sibuk, Anda akan kehilangan semua koneksi dan transaksi yang sedang diproses oleh klaster DB. Jika Anda menunggu hingga basis data idle untuk melakukan peningkatan, Anda mungkin harus menunggu lama.

Fitur patching nol-waktu henti (ZDP) meningkatkan proses peningkatan. Dengan ZDP, baik peningkatan versi minor maupun patch dapat diterapkan dengan dampak minimal ke klaster DB Aurora PostgreSQL Anda. ZDP digunakan saat menerapkan patch atau peningkatan versi minor yang lebih baru ke versi Aurora PostgreSQL dan rilis lain yang lebih tinggi dengan versi minor ini dan versi mayor yang lebih baru. Artinya, tingkatkan ke versi minor baru dari salah satu rilis ini dan seterusnya akan menggunakan ZDP.

Tabel berikut menunjukkan versi Aurora PostgreSQL dan kelas instans DB tempat ZDP tersedia:

Versi	Kelas instans db.r*	Kelas instans db.t*	Kelas instans db.x*	Kelas instans db.serverless
Versi 10.21.0 dan versi 10.21 yang lebih tinggi	Ya	Ya	Ya	N/A
Versi 11.16.0 dan versi 11.16 yang lebih tinggi	Ya	Ya	Ya	N/A
Versi 11.17 dan versi yang lebih tinggi	Ya	Ya	Ya	N/A
Versi 12.11.0 dan versi 12.11 yang lebih tinggi	Ya	Ya	Ya	N/A

Versi	Kelas instans db.r*	Kelas instans db.t*	Kelas instans db.x*	Kelas instans db.serverless
Versi 12.12 dan versi yang lebih tinggi	Ya	Ya	Ya	N/A
Versi 13.7.0 dan versi 13.7 yang lebih tinggi	Ya	Ya	Ya	N/A
Versi 13.8 dan versi yang lebih tinggi	Ya	Ya	Ya	Ya
Versi 14.3.1 dan versi 14.3 yang lebih tinggi	Ya	Ya	Ya	N/A
Versi 14.4.0 dan versi 14.4 yang lebih tinggi	Ya	Ya	Ya	N/A
Versi 14.5 dan versi yang lebih tinggi	Ya	Ya	Ya	Ya
Versi 15.3 dan versi yang lebih tinggi	Ya	Ya	Ya	Ya

ZDP bekerja dengan mempertahankan koneksi klien saat ini ke klaster DB Aurora PostgreSQL Anda selama proses peningkatan Aurora PostgreSQL. Namun, dalam kasus berikut, koneksi akan dibatalkan agar ZDP selesai:

- Kueri atau transaksi berjalan lama sedang berlangsung.
- Pernyataan bahasa definisi data (DDL) sedang berlangsung.
- Tabel sementara atau kunci tabel sedang digunakan.

- Semua sesi mendengarkan di saluran pemberitahuan.
- Kursor dalam status 'WITH HOLD' sedang digunakan.
- Koneksi TLSv1.3 atau TLSv1.1 sedang digunakan.

Selama proses peningkatan menggunakan ZDP, mesin basis data mencari titik tenang untuk menjeda semua transaksi baru. Tindakan ini melindungi basis data selama patch dan peningkatan. Untuk memastikan bahwa aplikasi Anda berjalan lancar dengan transaksi yang dijeda, sebaiknya Anda mengintegrasikan logika coba lagi ke dalam kode Anda. Pendekatan ini memastikan bahwa sistem dapat mengelola waktu henti singkat tanpa gagal dan dapat mencoba kembali transaksi baru setelah peningkatan.

Saat ZDP berhasil diselesaikan, sesi aplikasi dipertahankan kecuali untuk sesi koneksinya hilang, dan mesin basis data dimulai ulang saat peningkatan masih berlangsung. Meskipun memulai ulang mesin basis data dapat menyebabkan penurunan throughput sementara, hal ini biasanya hanya berlangsung selama beberapa detik atau paling lama kira-kira satu menit.

Dalam beberapa kasus, patching nol-waktu henti (ZDP) mungkin tidak berhasil. Misalnya, perubahan parameter yang berada dalam status `pending` pada klaster DB Aurora PostgreSQL Anda atau instansnya mengganggu ZDP.

Anda dapat menemukan metrik dan peristiwa untuk operasi ZDP di halaman Peristiwa di konsol. Peristiwa ini termasuk dimulainya peningkatan ZDP dan penyelesaian peningkatan. Dalam hal ini Anda dapat menemukan durasi pemrosesannya, dan jumlah koneksi yang dipertahankan dan hilang yang terjadi selama mulai ulang. Anda dapat menemukan detailnya di log kesalahan basis data Anda.

Meningkatkan mesin Aurora PostgreSQL ke versi minor baru

Anda dapat memutakhirkan cluster Aurora PostgreSQL DB Anda ke versi minor baru dengan menggunakan konsol, API, atau RDS. AWS CLI Sebelum melakukan peningkatan, sebaiknya Anda mengikuti praktik terbaik yang sama yang kami sarankan untuk peningkatan versi mayor. Seperti halnya versi mayor baru, versi minor baru juga dapat memiliki peningkatan pengoptimal, seperti perbaikan, yang dapat menyebabkan regresi rencana kueri. Untuk memastikan stabilitas rencana, sebaiknya Anda menggunakan ekstensi Manajemen Rencana Kueri (QPM) seperti yang dijelaskan dalam Memastikan stabilitas rencana setelah upgrade versi mayor.

Konsol

Untuk meningkatkan versi mesin klaster DB Aurora PostgreSQL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data, lalu pilih klaster DB yang ingin Anda tingkatkan.
3. Pilih Ubah. Halaman Ubah klaster DB akan muncul.
4. Untuk Versi mesin, pilih versi baru.
5. Pilih Lanjutkan dan periksa ringkasan perubahan.
6. Untuk segera menerapkan perubahan, pilih Terapkan segera. Dalam beberapa kasus, pemadaman dapat terjadi jika opsi ini dipilih. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora.
7. Di halaman konfirmasi, tinjau perubahan Anda. Jika sudah benar, pilih Ubah Klaster untuk menyimpan perubahan.

   Atau pilih Kembali untuk mengedit perubahan atau Batal untuk membatalkan perubahan.

AWS CLI

Untuk memutakhirkan versi mesin cluster DB, gunakan modify-db-cluster AWS CLI perintah dengan parameter berikut:

- `--db-cluster-identifier` – Nama klaster DB Aurora PostgreSQL Anda.
- `--engine-version` – Nomor versi peningkatan mesin basis data. Untuk informasi tentang versi mesin yang valid, gunakan AWS CLI describe-db-engine-versionsperintah.
- `--no-apply-immediately` – Terapkan perubahan selama periode pemeliharaan berikutnya. Untuk menerapkan perubahan secara langsung, gunakan `--apply-immediately`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier mydbcluster \
    --engine-version new_version \
    --no-apply-immediately
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --engine-version new_version ^
    --no-apply-immediately
```

RDS API

Untuk meningkatkan versi mesin klaster DB, gunakan operasi ModifyDBCluster. Tentukan parameter berikut:

- `DBClusterIdentifier` – Nama klaster DB, misalnya *`mydbcluster`*.
- `EngineVersion` – Nomor versi peningkatan mesin basis data. Untuk informasi tentang versi mesin yang valid, gunakan operasi DescribeDB EngineVersions.
- `ApplyImmediately` – Apakah akan menerapkan perubahan secara langsung atau selama periode pemeliharaan berikutnya. Untuk segera menerapkan perubahan, tetapkan nilai ke `true`. Untuk menerapkan perubahan selama periode pemeliharaan berikutnya, tetapkan nilai ke `false`.

Meningkatkan ekstensi PostgreSQL

Meningkatkan klaster DB Aurora PostgreSQL Anda ke versi mayor atau minor baru tidak akan meningkatkan ekstensi PostgreSQL secara bersamaan. Untuk sebagian besar ekstensi, Anda meningkatkan ekstensi setelah peningkatan versi mayor atau minor selesai. Namun, dalam beberapa kasus, Anda meningkatkan ekstensi sebelum meningkatkan mesin DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat list of extensions to update di Menguji peningkatan klaster DB produksi ke versi mayor baru.

Menginstal ekstensi PostgreSQL membutuhkan hak istimewa `rds_superuser`. Biasanya, `rds_superuser` mendelegasikan izin atas ekstensi tertentu untuk pengguna yang relevan (peran), untuk memfasilitasi pengelolaan ekstensi yang ditentukan. Itu berarti bahwa tugas meningkatkan semua ekstensi di klaster DB Aurora PostgreSQL Anda mungkin melibatkan banyak pengguna (peran) yang berbeda. Perhatikan hal ini terutama jika Anda ingin mengotomatiskan proses peningkatan dengan menggunakan skrip. Untuk informasi selengkapnya tentang hak istimewa dan peran PostgreSQL, lihat Keamanan dengan Amazon Aurora PostgreSQL.

> **Note**
>
> Untuk informasi selengkapnya tentang ekstensi PostGIS, lihat Mengelola data spasial dengan ekstensi PostGIS (Langkah 6: Meningkatkan ekstensi PostGIS).

Untuk memperbarui ekstensi `pg_repack`, hapus ekstensi dan kemudian buat versi baru di instans DB yang ditingkatkan. Untuk informasi selengkapnya, lihat pg_repack installation di dokumentasi `pg_repack`.

Untuk memperbarui ekstensi setelah peningkatan mesin, gunakan perintah `ALTER EXTENSION UPDATE`.

```
ALTER EXTENSION extension_name UPDATE TO 'new_version';
```

Untuk mencantumkan ekstensi yang saat ini terinstal, gunakan katalog pg_extension PostgreSQL dalam perintah berikut.

```
SELECT * FROM pg_extension;
```

Untuk melihat daftar versi ekstensi tertentu yang tersedia untuk penginstalan Anda, gunakan tampilan pg_available_extension_versions PostgreSQL dalam perintah berikut.

```
SELECT * FROM pg_available_extension_versions;
```

Teknik peningkatan blue/green alternatif

Dalam situasi tertentu, prioritas utama Anda adalah melakukan switchover langsung dari klaster lama ke klaster yang ditingkatkan. Dalam situasi seperti itu, Anda dapat menggunakan proses multistep yang menjalankan cluster side-by-side lama dan baru. Di sini, Anda mereplikasi data dari klaster lama ke klaster baru hingga Anda siap untuk mengambil alih klaster baru. Untuk detailnya, lihat Menggunakan Deployment Blue/Green untuk pembaruan basis data.

Rilis dukungan jangka panjang (LTS) Aurora PostgreSQL

Setiap Aurora PostgreSQL versi baru tetap tersedia sampai waktu tertentu untuk Anda gunakan saat membuat atau meningkatkan klaster DB. Setelah periode ini, Anda harus meningkatkan klaster yang menggunakan versi tersebut. Anda dapat meningkatkan klaster Anda secara manual sebelum periode dukungan berakhir, atau Aurora dapat secara otomatis meningkatkan versi untuk Anda ketika versi Aurora PostgreSQL tidak lagi didukung.

Aurora menetapkan versi Aurora PostgreSQL tertentu sebagai rilis dukungan jangka panjang (LTS). Klaster basis data yang menggunakan rilis LTS dapat bertahan pada versi yang sama lebih lama

dan menjalani siklus peningkatan yang lebih sedikit dibandingkan klaster yang menggunakan rilis non-LTS. Versi minor LTS hanya mencakup perbaikan bug (melalui versi patch); versi LTS tidak menyertakan fitur baru yang dirilis setelah diperkenalkan.

Setahun sekali, klaster DB yang berjalan pada versi minor LTS di-patch ke versi patch terbaru dari rilis LTS. Kami melakukan patching ini untuk membantu memastikan bahwa Anda mendapat manfaat dari perbaikan keamanan dan stabilitas kumulatif. Kami mungkin memberi patch versi minor LTS lebih sering jika ada perbaikan penting, seperti untuk keamanan, yang perlu diterapkan.

> **Note**
>
> Agar tetap menggunakan versi minor LTS selama siklus hidupnya, pastikan untuk menonaktifkan Peningkatan versi minor otomatis untuk instans DB Anda. Untuk menghindari peningkatan klaster DB Anda secara otomatis dari versi minor LTS, atur Peningkatan versi minor otomatis ke Tidak pada semua instans DB di klaster Aurora Anda.

Kami menyarankan Anda meningkatkan ke rilis terbaru, daripada menggunakan rilis LTS, untuk sebagian besar klaster Aurora PostgreSQL Anda. Melakukan hal itu memanfaatkan Aurora sebagai layanan terkelola dan memberi Anda akses ke fitur terbaru dan perbaikan bug. Rilis LTS ditujukan untuk klaster yang memiliki karakteristik berikut:

- Anda tidak dapat memberikan waktu henti pada aplikasi Aurora PostgreSQL untuk peningkatan di luar kejadian langka untuk patch penting.
- Siklus pengujian untuk klaster dan aplikasi terkait membutuhkan waktu lama untuk setiap pembaruan ke mesin basis data Aurora PostgreSQL.
- Versi basis data untuk klaster Aurora PostgreSQL Anda memiliki semua fitur mesin DB dan perbaikan bug yang dibutuhkan aplikasi Anda.

Rilis LTS saat ini untuk Aurora PostgreSQL adalah sebagai berikut:

- PostgreSQL 14.6. Versi ini dirilis pada 20 Januari 2023. Untuk informasi selengkapnya, lihat PostgreSQL 14.6 di Catatan Rilis untuk Aurora PostgreSQL.
- PostgreSQL 13.9. Versi ini dirilis pada 20 Januari 2023. Untuk informasi selengkapnya, lihat PostgreSQL 13.9 di Catatan Rilis untuk Aurora PostgreSQL.
- PostgreSQL 12.9. Versi ini dirilis pada 25 Februari 2022. Untuk informasi selengkapnya, lihat PostgreSQL 12.9 di Catatan Rilis untuk Aurora PostgreSQL.

- PostgreSQL 11.9 (Aurora PostgreSQL 3.4.) Versi ini dirilis pada 11 Desember 2020. Untuk informasi selengkapnya tentang versi ini, lihat PostgreSQL 11.9, Aurora PostgreSQL rilis 3.4 di Catatan Rilis untuk Aurora PostgreSQL.

Untuk informasi tentang cara mengidentifikasi versi mesin basis data dan Aurora, lihat Mengidentifikasi versi Amazon Aurora PostgreSQL.

Menggunakan basis data global Amazon Aurora

Basis data global Amazon Aurora mencakup beberapa Wilayah AWS, memungkinkan pembacaan global latensi rendah dan memberikan pemulihan cepat dari pemadaman langka yang mungkin memengaruhi keseluruhan. Wilayah AWS Basis data global Aurora memiliki satu klaster DB primer di satu Wilayah dan hingga lima klaster DB sekunder di Wilayah yang berbeda-beda.

Topik

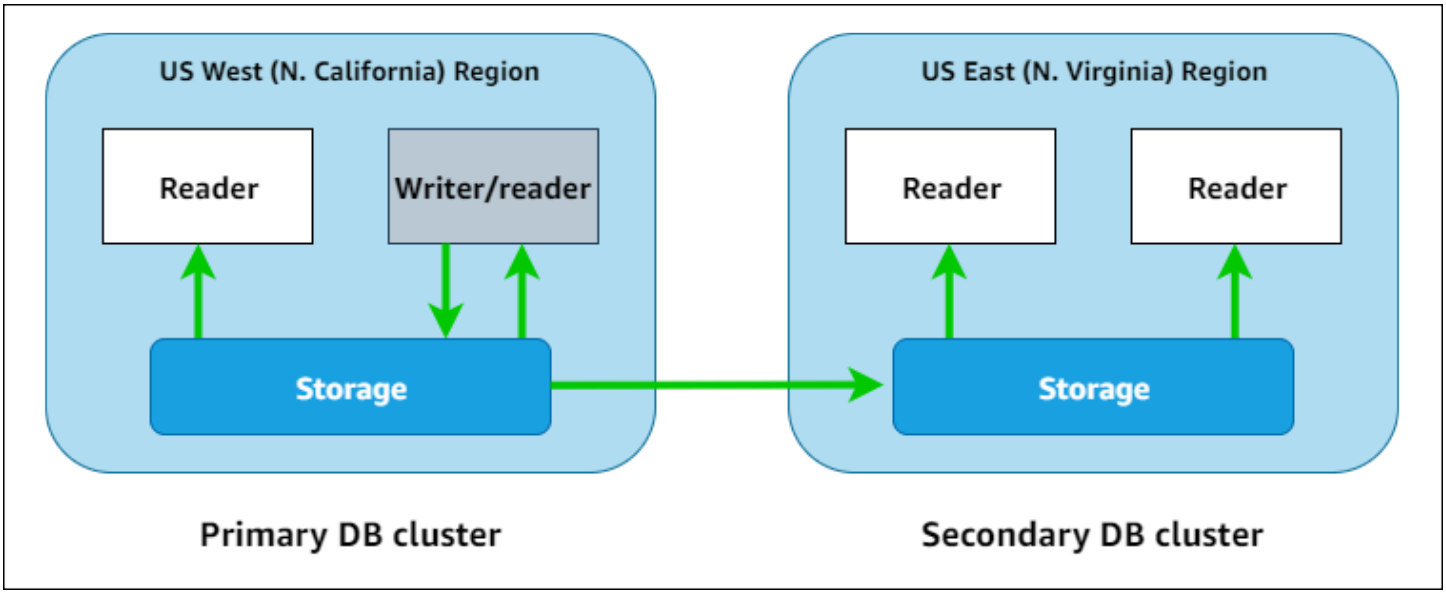
- Gambaran umum basis data global Amazon Aurora
- Keuntungan basis data global Amazon Aurora
- Ketersediaan Wilayah dan versi
- Keterbatasan basis data global Amazon Aurora
- Memulai basis data global Amazon Aurora
- Mengelola basis data global Amazon Aurora
- Terhubung ke basis data global Amazon Aurora
- Menggunakan penerusan menulis dalam basis data global Amazon Aurora
- Menggunakan switchover atau failover dalam basis data global Amazon Aurora
- Memantau basis data global Amazon Aurora
- Menggunakan basis data global Amazon Aurora dengan layanan AWS lainnya
- Meningkatkan basis data global Amazon Aurora

Gambaran umum basis data global Amazon Aurora

Dengan menggunakan basis data global Amazon Aurora, Anda dapat menjalankan aplikasi yang terdistribusi secara global menggunakan satu basis data Aurora yang mencakup beberapa Wilayah AWS.

Database global Aurora terdiri dari satu primer Wilayah AWS tempat data Anda ditulis, dan hingga lima sekunder hanya-baca. Wilayah AWS Anda mengeluarkan operasi penulisan secara langsung ke klaster DB primer di Wilayah AWS primer. Aurora mereplikasi data ke sekunder Wilayah AWS menggunakan infrastruktur khusus, dengan latensi biasanya di bawah satu detik.

Dalam diagram berikut, Anda dapat menemukan contoh database global Aurora yang mencakup dua. Wilayah AWS



Anda dapat menaikkan skala masing-masing klaster sekunder secara independen dengan menambahkan satu atau beberapa Aurora Replicas (instans DB Aurora hanya-baca) untuk melayani beban kerja hanya-baca.

Hanya klaster primer yang melakukan operasi penulisan. Klien yang melakukan operasi penulisan terhubung ke titik akhir klaster DB dari klaster DB primer. Seperti yang ditunjukkan dalam diagram, basis data global Aurora menggunakan volume penyimpanan klaster, bukan mesin basis data untuk replikasi. Untuk mempelajari selengkapnya, lihat Gambaran umum penyimpanan Amazon Aurora.

Basis data global Aurora dirancang untuk aplikasi yang memiliki jejak global. Klaster DB sekunder hanya-baca (Wilayah AWS) memungkinkan Anda mendukung operasi pembacaan lebih dekat dengan pengguna aplikasi. Dengan menggunakan fitur penerusan tulis, Anda juga dapat mengonfigurasi basis data global Aurora agar klaster sekunder mengirim data ke primer. Untuk informasi selengkapnya, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.

Basis data global Aurora mendukung dua operasi yang berbeda dalam mengubah Wilayah klaster DB primer Anda, tergantung pada skenarionya: switchover basis data global dan failover basis data global.

- Untuk prosedur operasional yang direncanakan seperti rotasi Regional, gunakan switchover basis data global (sebelumnya disebut "failover terencana yang dikelola"). Dengan fitur ini, Anda dapat merelokasi klaster primer dari basis data global Aurora yang sehat ke salah satu Wilayah sekundernya tanpa kehilangan data. Untuk mempelajari selengkapnya, lihat Melakukan switchover untuk basis data global Amazon Aurora.

- Untuk memulihkan basis data global Aurora Anda setelah terjadi pemadaman di Wilayah primer, gunakan failover basis data global. Dengan fitur ini, Anda akan melakukan failover klaster DB primer ke Wilayah (failover lintas Wilayah). Untuk mempelajari selengkapnya, lihat Melakukan failover terkelola untuk basis data global Aurora.

Keuntungan basis data global Amazon Aurora

Dengan menggunakan basis data global Aurora, Anda dapat memperoleh keuntungan berikut:

- Pembacaan global dengan latensi lokal—Jika memiliki kantor di seluruh dunia, Anda dapat menggunakan basis data global Aurora agar sumber informasi utama Anda tetap diperbarui di Wilayah AWS primer. Kantor yang berada di Wilayah lain dapat mengakses informasi di Wilayah masing-masing, dengan latensi lokal.
- Klaster DB Aurora sekunder yang dapat diskalakan—Anda dapat menskalakan klaster sekunder dengan menambahkan lebih banyak instans hanya-baca (Aurora Replicas) ke Wilayah AWS sekunder. Klaster sekunder bersifat hanya-baca, sehingga dapat mendukung hingga 16 instans Aurora Replica hanya-baca, bukan batas biasa sebanyak 15 untuk satu klaster Aurora.
- Replikasi dari klaster DB Aurora primer ke sekunder yang cepat—Replikasi yang dilakukan oleh basis data global Aurora memberikan dampak performa yang kecil pada klaster DB primer. Sumber daya instans DB dikhususkan sepenuhnya untuk melayani beban kerja baca dan tulis aplikasi.
- Pemulihan dari pemadaman tingkat Wilayah—Klaster sekunder memungkinkan Anda menyediakan basis data global Aurora di Wilayah AWS primer baru dengan lebih cepat (RTO lebih rendah) dan dengan dampak kehilangan data yang lebih sedikit (RPO lebih rendah) daripada solusi replikasi tradisional.

Ketersediaan Wilayah dan versi

Ketersediaan dan dukungan fitur bervariasi di seluruh versi spesifik dari setiap mesin basis data Aurora, dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang ketersediaan Wilayah dan versi dengan Aurora serta basis data global, lihat Basis data global Aurora.

Keterbatasan basis data global Amazon Aurora

Pembatasan berikut saat ini berlaku untuk basis data global Aurora:

- Database global Aurora tersedia dalam versi tertentu Wilayah AWS dan untuk Aurora MySQL dan Aurora PostgreSQL tertentu saja. Untuk informasi selengkapnya, lihat Basis data global Aurora.
- Basis data global Aurora memiliki persyaratan konfigurasi khusus untuk kelas instans DB Aurora yang didukung, jumlah maksimum Wilayah AWS, dan sebagainya. Untuk informasi selengkapnya, lihat Persyaratan konfigurasi basis data global Amazon Aurora.
- Untuk Aurora MySQL dengan kompatibilitas MySQL 5.7, pengalih basis data global Aurora memerlukan versi 2.09.1 atau versi minor yang lebih tinggi.
- Anda dapat melakukan switchover atau failover lintas wilayah terkelola pada database global Aurora hanya jika cluster DB primer dan sekunder memiliki versi mesin tingkat mayor, minor, dan patch yang sama. Namun, tingkat patch dapat berbeda jika versi mesin minor merupakan salah satu versi berikut:

Mesin basis data	Versi mesin minor
Aurora PostgreSQL	• Versi 14.5 atau versi minor yang lebih tinggi • Versi 13.8 atau versi minor yang lebih tinggi • Versi 12.12 atau versi minor yang lebih tinggi • Versi 11.17 atau versi minor yang lebih tinggi

  Untuk informasi selengkapnya, lihat Kompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola.
- Basis data global Aurora saat ini tidak mendukung fitur Aurora berikut:
  - Aurora Serverless v1
  - Backtracking di Aurora
- Untuk batasan penggunaan fitur Proksi RDS dengan basis data global, lihat Batasan untuk Proksi RDS dengan basis data global.
- Peningkatan versi minor otomatis tidak berlaku untuk klaster Aurora MySQL dan Aurora PostgreSQL yang merupakan bagian dari basis data global Aurora. Perlu diketahui bahwa Anda dapat menentukan pengaturan ini untuk instans DB yang merupakan bagian dari klaster basis data global, tetapi pengaturan tersebut tidak akan berpengaruh.
- Basis data global Aurora saat ini tidak mendukung Aurora Auto Scaling untuk klaster DB sekunder.

- Untuk menggunakan aliran aktivitas database pada database global Aurora yang menjalankan Aurora MySQL 5.7, versi mesin harus versi 2.08 atau lebih tinggi. Untuk informasi tentang streaming aktivitas basis data, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.
- Berikut adalah batasan yang saat ini berlaku untuk peningkatan basis data global Aurora:
  - Anda tidak dapat menerapkan grup parameter kustom ke klaster basis data global saat melakukan peningkatan versi utama dari basis data global Aurora tersebut. Anda membuat grup parameter kustom di setiap Wilayah klaster global dan menerapkannya secara manual ke klaster Regional setelah melakukan peningkatan.
  - Dengan basis data global Aurora berdasarkan Aurora MySQL, Anda tidak dapat melakukan peningkatan in-place dari Aurora MySQL versi 2 ke versi 3 jika parameter `lower_case_table_names` diaktifkan. Untuk informasi selengkapnya tentang metode yang dapat Anda gunakan, lihat Peningkatan versi utama.
  - Dengan basis data global Aurora berdasarkan Aurora PostgreSQL, Anda tidak dapat melakukan peningkatan versi utama dari mesin DB Aurora jika fitur sasaran titik pemulihan (RPO) diaktifkan. Untuk informasi tentang fitur RPO, lihat Mengelola RPO untuk basis data global berbasis Aurora PostgreSQL.
  - Dengan basis data global Aurora berdasarkan Aurora MySQL, Anda tidak dapat melakukan peningkatan versi minor dari versi 3.01 atau 3.02 ke 3.03 atau yang lebih tinggi dengan menggunakan proses standar. Untuk detail tentang proses yang akan digunakan, lihat Meningkatkan Aurora MySQL dengan mengubah versi mesin.

  Untuk informasi tentang peningkatan basis data global Aurora, lihat Meningkatkan basis data global Amazon Aurora.
- Anda tidak dapat menghentikan atau memulai klaster DB Aurora dalam basis data global Aurora secara individual. Untuk mempelajari informasi lebih lanjut, lihat Menghentikan dan memulai klaster DB Amazon Aurora.
- Aurora Replicas yang melekat pada klaster DB Aurora sekunder dapat dimulai ulang dalam keadaan tertentu. Jika instans DB Wilayah AWS penulis primer dimulai ulang atau gagal, Replika Aurora di Wilayah sekunder juga dimulai ulang. Klaster sekunder kemudian tidak akan tersedia hingga semua replika kembali tersinkronisasi dengan instans penulis klaster DB primer. Perilaku klaster primer saat melakukan reboot atau failover sama seperti klaster DB tunggal nonglobal. Untuk informasi selengkapnya, lihat Replikasi dengan Amazon Aurora.

  Pastikan Anda memahami dampaknya terhadap basis data global Aurora sebelum membuat perubahan pada klaster DB primer. Untuk mempelajari informasi lebih lanjut, lihat Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan.

- Database global Aurora saat ini tidak mendukung `inaccessible-encryption-credentials-recoverable` status ketika Amazon Aurora kehilangan akses ke AWS KMS kunci untuk cluster DB. Dalam kasus ini, klaster DB yang terenkripsi akan langsung berstatus `inaccessible-encryption-credentials` terminal. Untuk informasi selengkapnya tentang status ini, lihat Melihat status klaster DB.
- Klaster DB berbasis Aurora PostgreSQL yang berjalan pada basis data global Aurora memiliki batasan berikut:
  - Manajemen cache klaster tidak didukung untuk klaster DB Aurora PostgreSQL yang merupakan bagian dari basis data global Aurora.
  - Jika klaster DB primer dari basis data global Aurora didasarkan pada replika instans Amazon RDS PostgreSQL, Anda tidak dapat membuat klaster sekunder. Jangan mencoba membuat sekunder dari cluster itu menggunakan operasi AWS Management Console, the AWS CLI, atau `CreateDBCluster` API. Upaya tersebut akan kehabisan waktu, dan klaster sekunder tidak akan dibuat.

Sebaiknya Anda membuat klaster DB sekunder untuk basis data global Aurora Anda dengan menggunakan versi mesin DB Aurora yang sama seperti klaster primer. Lihat informasi yang lebih lengkap di Membuat basis data global Amazon Aurora.

Memulai basis data global Amazon Aurora

Untuk memulai dengan basis data global Aurora, pertama putuskan mesin DB Aurora apa yang ingin Anda gunakan dan di Wilayah AWS apa. Hanya versi mesin basis data Aurora MySQL dan Aurora PostgreSQL tertentu saja di beberapa Wilayah Wilayah AWS yang mendukung basis data global Aurora. Untuk daftar lengkapnya, lihat Basis data global Aurora.

Anda dapat membuat basis data global Aurora dengan salah satu cara berikut:

- Buat basis data global Aurora baru dengan klaster DB Aurora dan instans DB Aurora baru — Anda dapat melakukannya dengan mengikuti langkah-langkah di Membuat basis data global Amazon Aurora. Setelah Anda membuat klaster DB Aurora primer, Anda lalu menambahkan Wilayah AWS sekunder dengan mengikuti langkah-langkah di Menambahkan Wilayah AWS ke basis data global Amazon Aurora.
- Gunakan klaster DB Aurora yang sudah ada yang mendukung fitur basis data global Aurora dan tambahkan Wilayah AWS ke sana — Anda dapat melakukannya hanya jika klaster DB Aurora yang

sudah ada menggunakan versi mesin DB yang mendukung mode global Aurora atau kompatibel secara global. Untuk beberapa versi mesin DB, mode ini eksplisit, tapi tidak untuk versi lainnya.

Periksa apakah Anda dapat memilih Tambahkan region untuk Tindakan pada AWS Management Console ketika klaster DB Aurora Anda dipilih. Jika Anda dapat memilihnya, Anda dapat menggunakan klaster DB Aurora tersebut untuk klaster global Aurora Anda. Untuk informasi selengkapnya, lihat Menambahkan Wilayah AWS ke basis data global Amazon Aurora.

Sebelum membuat basis data global Aurora, kami sarankan Anda memahami semua persyaratan konfigurasi.

Topik

- Persyaratan konfigurasi basis data global Amazon Aurora
- Membuat basis data global Amazon Aurora
- Menambahkan Wilayah AWS ke basis data global Amazon Aurora
- Membuat klaster DB Aurora tanpa kepala di Wilayah sekunder
- Menggunakan snapshot untuk basis data global Amazon Aurora Anda

Persyaratan konfigurasi basis data global Amazon Aurora

Basis data global Aurora menjangkau setidaknya dua Wilayah AWS. Wilayah AWS primer mendukung klaster DB Aurora yang memiliki satu instans DB Aurora penulis. Wilayah AWS sekunder menjalankan klaster DB Aurora hanya-baca yang seluruhnya terdiri dari Replika Aurora. Diperlukan setidaknya satu Wilayah AWS sekunder, tetapi basis data global Aurora dapat memiliki hingga lima Wilayah AWS sekunder. Tabel ini mencantumkan jumlah maksimum klaster DB Aurora, instans DB Aurora, dan Replika Aurora yang diperbolehkan dalam basis data global Aurora.

Deskripsi	Wilayah AWS Primer	Wilayah AWS Sekunder
Klaster DB Aurora	1	5 (maksimum)
Instans penulis	1	0
Instans hanya-baca (replika Aurora), per klaster DB Aurora	15 (maks)	16 (total)

Deskripsi	Wilayah AWS Primer	Wilayah AWS Sekunder
Instans hanya-baca (maks. yang diperbolehkan, dengan jumlah aktual dari Wilayah sekunder)	15 - s	s = jumlah total Wilayah AWS sekunder

Klaster DB Aurora yang membentuk basis data global Aurora memiliki persyaratan khusus berikut:

- Persyaratan kelas instans DB — Sebuah basis data global Aurora membutuhkan kelas instans DB yang dioptimalkan untuk aplikasi sarat memori. Untuk informasi tentang kelas instans DB memori yang dioptimalkan, lihat kelas instans DB. Kami menyarankan Anda menggunakan kelas instans db.r5 atau lebih tinggi.
- Persyaratan Wilayah AWS – Basis data global Aurora membutuhkan klaster DB Aurora primer di satu Wilayah AWS, dan setidaknya satu klaster DB Aurora sekunder di Wilayah yang berbeda. Anda dapat membuat hingga lima klaster DB Aurora sekunder (hanya-baca), dan masing-masing harus di Wilayah yang berbeda. Dengan kata lain, tidak ada dua klaster DB Aurora dalam basis data global Aurora dapat berada di Wilayah AWS yang sama.
- Persyaratan penamaan — Nama-nama yang Anda pilih untuk masing-masing klaster DB Aurora Anda harus unik, di semua Wilayah AWS. Anda tidak dapat menggunakan nama yang sama untuk klaster DB Aurora yang berbeda meskipun berada di Wilayah yang berbeda.
- Persyaratan kapasitas untuk Aurora Serverless v2 — Untuk basis data global dengan Aurora Serverless v2, kapasitas minimum yang diperlukan untuk klaster DB di Wilayah AWS primer adalah 8 ACU.

Sebelum Anda dapat mengikuti prosedur di bagian ini, Anda memerlukan Akun AWS. Lengkapi tugas penyiapan untuk menggunakan Amazon Aurora. Untuk informasi selengkapnya, lihat Menyiapkan lingkungan Anda untuk Amazon Aurora. Anda juga perlu menyelesaikan langkah awal lainnya untuk membuat klaster DB Aurora. Untuk mempelajari selengkapnya, lihat Membuat klaster DB Amazon Aurora.

Membuat basis data global Amazon Aurora

Dalam beberapa kasus, Anda mungkin memiliki klaster DB Aurora yang disediakan untuk menjalankan mesin basis data Aurora yang kompatibel secara global. Jika demikian, Anda dapat menambahkan Wilayah AWS lainnya ke sana untuk membuat basis data global Aurora Anda. Untuk melakukannya, lihat Menambahkan Wilayah AWS ke basis data global Amazon Aurora.

Untuk membuat basis data global Aurora dengan menggunakan AWS Management Console, AWS CLI, atau API RDS, gunakan langkah-langkah berikut.

Konsol

Langkah-langkah untuk membuat basis data global Aurora dimulai dengan masuk ke Wilayah AWS yang mendukung fitur basis data global Aurora. Untuk daftar lengkap, lihat Basis data global Aurora.

Salah satu langkah berikut adalah memilih cloud privat virtual (VPC) berdasarkan Amazon VPC untuk klaster DB Aurora Anda. Untuk menggunakan VPC Anda sendiri, sebaiknya Anda membuatnya terlebih dahulu sehingga VPC tersedia untuk Anda pilih. Pada saat yang sama, buat subnet terkait apa pun, serta grup subnet dan grup keamanan sesuai keperluan. Untuk mempelajari caranya, lihat Tutorial: Membuat Amazon VPC untuk digunakan bersama instans DB.

Untuk informasi umum tentang cara membuat klaster DB Aurora klaster, lihat Membuat klaster DB Amazon Aurora.

Membuat basis data global Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Buat basis data. Di halaman Buat basis data, lakukan hal berikut:
   - Untuk metode pembuatan basis data, pilih Pembuatan Standar. (Jangan pilih Pembuatan mudah.)
   - Untuk `Engine type` di bagian Opsi mesin, pilih jenis mesin yang berlaku, Aurora (Kompatibel dengan MySQL) atau Aurora (Kompatibel dengan PostgreSQL).
3. Lanjutkan membuat basis data global Aurora Anda dengan menggunakan langkah-langkah dari prosedur berikut.

Membuat global basis data menggunakan Aurora MySQL

Langkah-langkah berikut berlaku untuk semua versi Aurora MySQL.

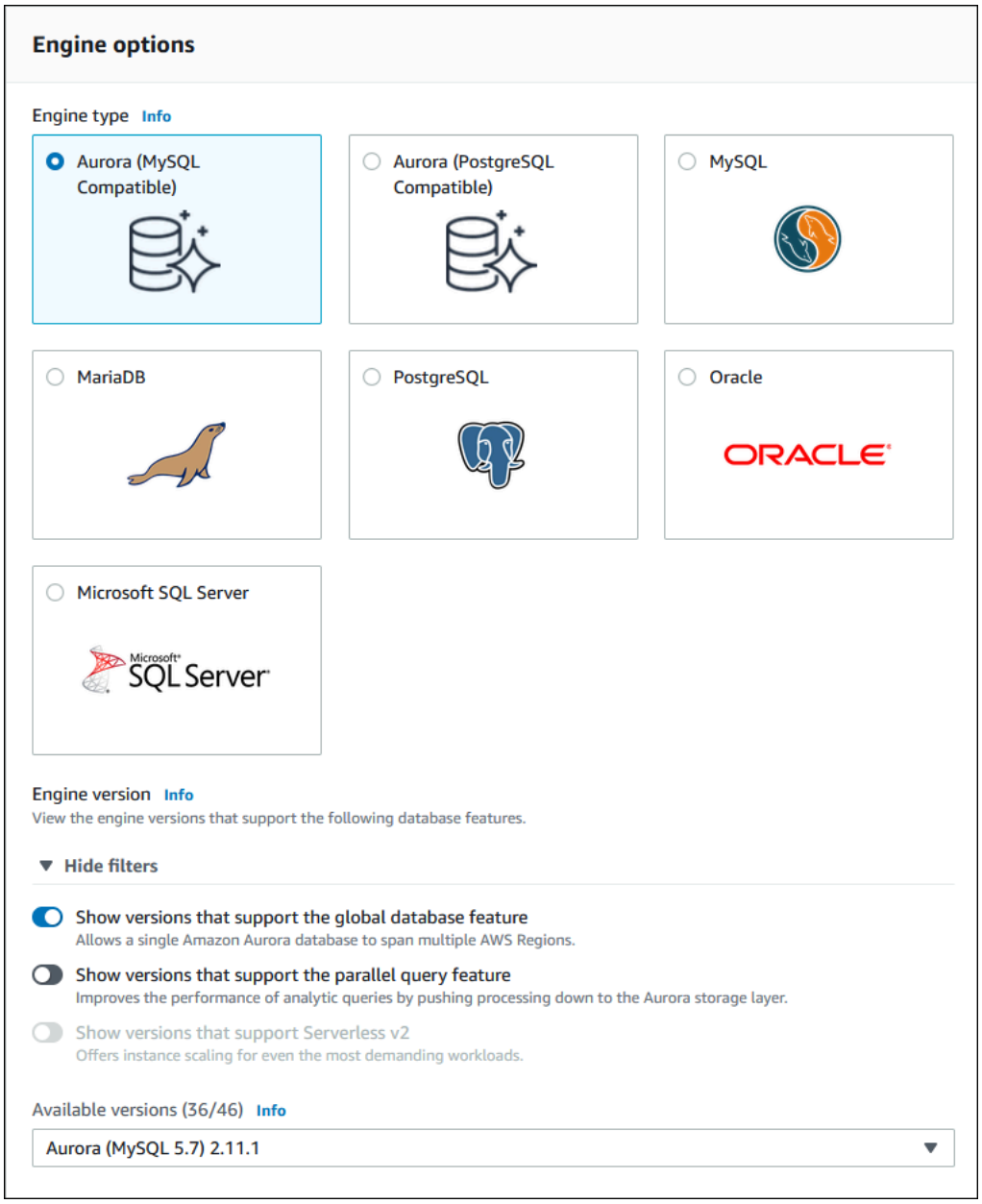
Membuat basis data global Aurora menggunakan Aurora MySQL

Lengkapi halaman Buat basis data.

1. Untuk Opsi mesin, pilih salah satu hal berikut:

a. Perluas Tampilkan filter, lalu aktifkan Tampilkan versi yang mendukung fitur basis data global.

b. Untuk Versi mesin, pilih versi Aurora MySQL yang ingin Anda gunakan untuk basis data global Aurora Anda.

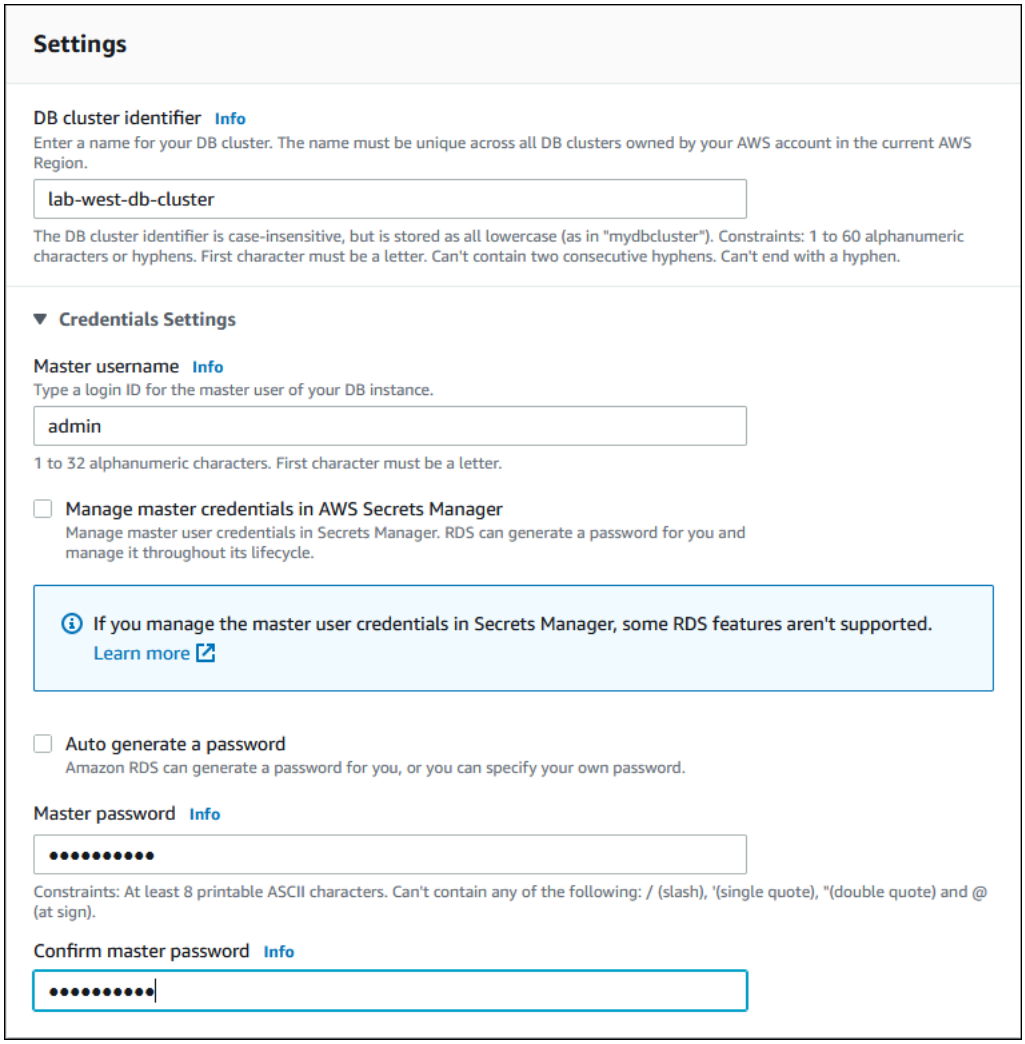


2. Untuk Templat, pilih Produksi. Atau Anda dapat memilih Dev/Tes jika sesuai untuk kasus penggunaan Anda. Jangan gunakan Dev/Tes dalam lingkungan produksi.

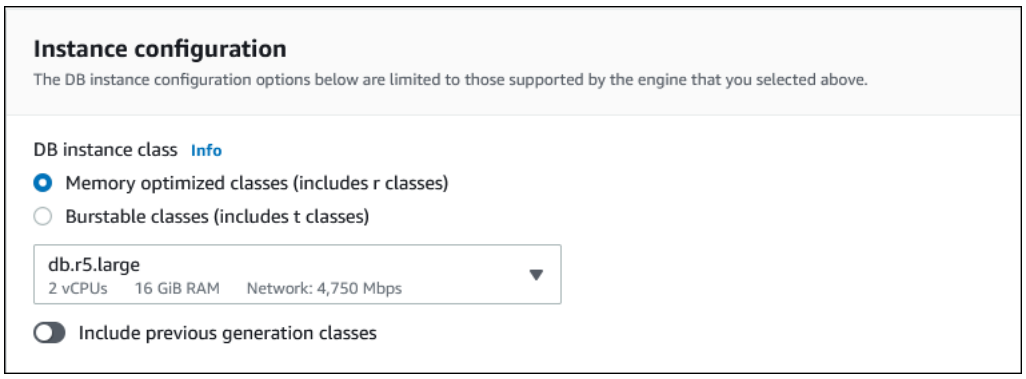
3. Untuk Pengaturan, lakukan hal berikut:

a. Masukkan nama yang bermakna untuk pengidentifikasi klaster DB. Ketika Anda selesai membuat basis data global Aurora, nama ini mengidentifikasi klaster DB primer.

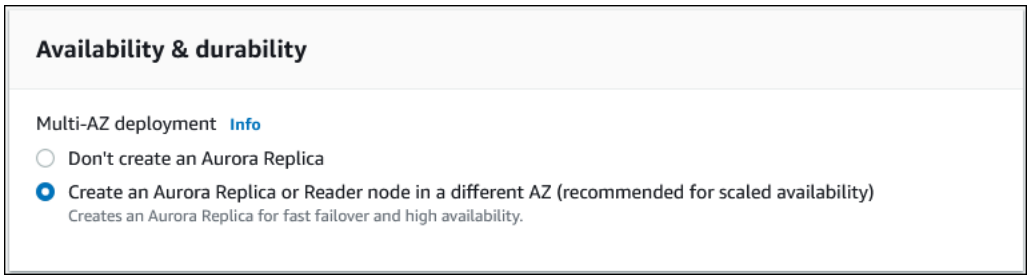
b. Masukkan kata sandi Anda sendiri untuk akun pengguna `admin` untuk instans DB tersebut, atau minta Aurora untuk membuatkannya untuk Anda. Jika Anda memilih untuk membuat kata sandi secara otomatis, Anda akan mendapatkan opsi untuk menyalin kata sandi.



4. Untuk Kelas instans DB, pilih `db.r5.large` atau kelas instans DB memori yang dioptimalkan lainnya. Kami menyarankan Anda menggunakan kelas instans db.r5 atau lebih tinggi.



5. Untuk Ketersediaan & daya tahan, kami sarankan Anda memilih untuk meminta Aurora membuatkan Replika Aurora di Zona Ketersediaan (AZ) yang berbeda untuk Anda. Jika Anda tidak membuat Replika Aurora sekarang, Anda harus melakukannya nanti.



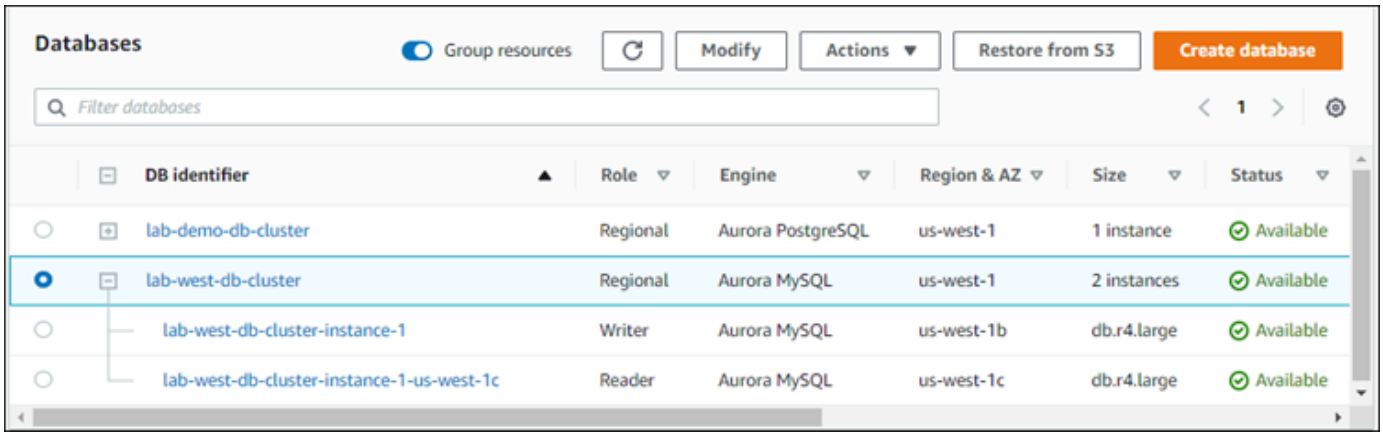
6. Untuk Konektivitas, pilih cloud privat virtual (VPC) berdasarkan Amazon VPC yang menetapkan lingkungan jaringan virtual untuk instans DB ini. Anda dapat memilih default untuk menyederhanakan tugas ini.
7. Lengkapi pengaturan Autentikasi basis data. Untuk menyederhanakan prosesnya, Anda dapat memilih Autentikasi kata sandi sekarang dan menyiapkan AWS Identity and Access Management (IAM) nanti.
8. Pada Konfigurasi tambahan, lakukan hal berikut:

   a. Masukkan nama untuk Nama basis data awal untuk membuat instans DB Aurora primer untuk klaster ini. Ini adalah simpul penulis untuk klaster DB Aurora primer.

      Tetap pilih default untuk grup parameter klaster DB dan grup parameter DB, kecuali Anda memiliki grup parameter kustom Anda sendiri yang ingin Anda gunakan.

   b. Kosongkan kotak centang Aktifkan lacak mundur jika dipilih. Basis data global Aurora tidak mendukung pelacakan mundur. Jika tidak, terima pengaturan default lainnya untuk Konfigurasi tambahan.
9. Pilih Buat basis data.

Mungkin diperlukan beberapa menit bagi Aurora untuk menyelesaikan proses pembuatan instans DB Aurora, Replika Aurora, dan klaster DB Aurora. Anda akan tahu ketika klaster DB Aurora siap untuk digunakan sebagai klaster DB primer dalam basis data global Aurora berdasarkan statusnya. Ketika sudah siap, statusnya dan status simpul penulis serta replika adalah Tersedia, seperti yang ditunjukkan berikut ini.



Ketika klaster DB primer Anda tersedia, buat basis data global Aurora dengan menambahkan klaster sekunder untuk itu. Untuk melakukannya, ikuti langkah-langkah di Menambahkan Wilayah AWS ke basis data global Amazon Aurora.

Membuat global basis data menggunakan Aurora PostgreSQL

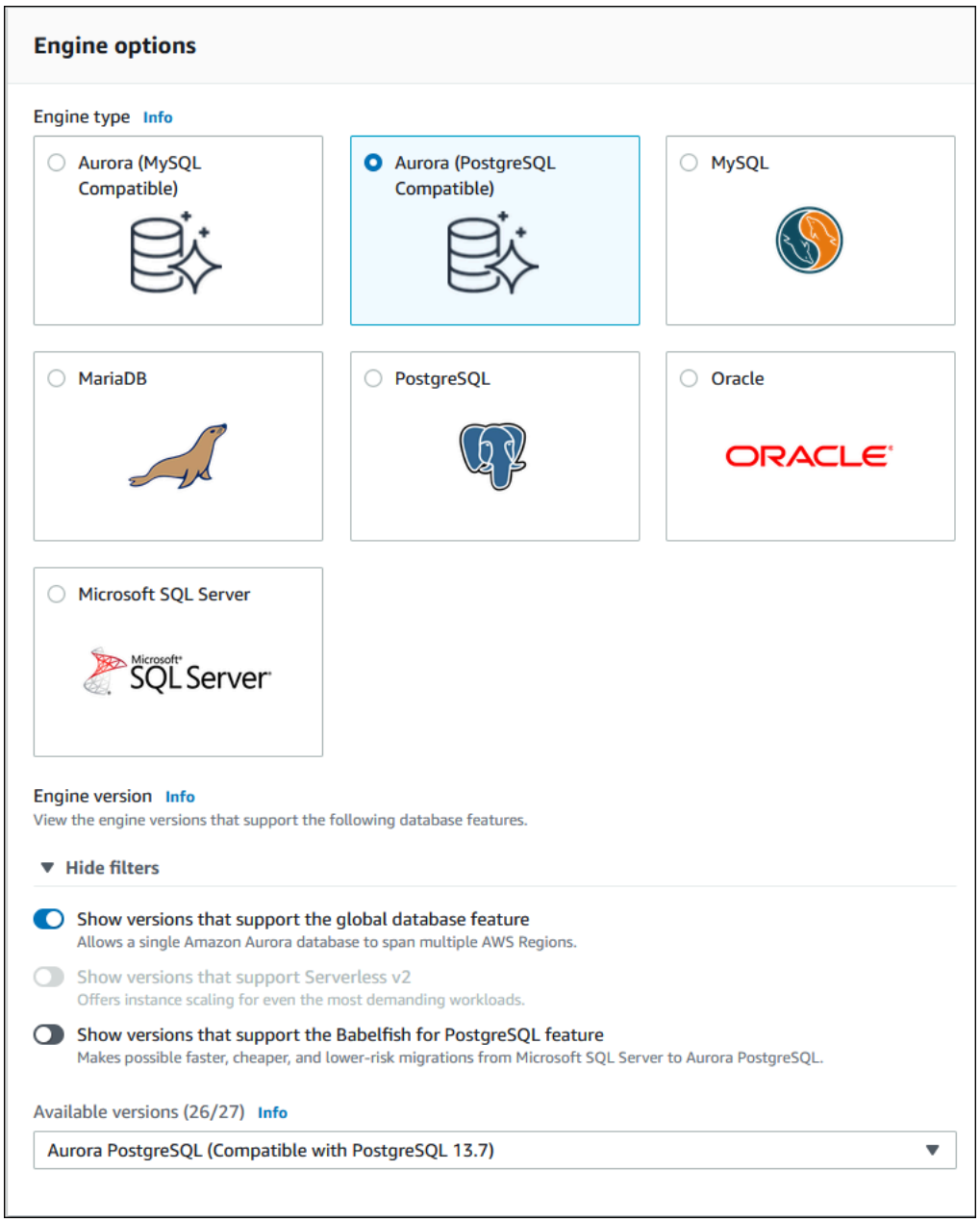
Membuat basis data global Aurora menggunakan Aurora PostgreSQL

Lengkapi halaman Buat basis data.

1. Untuk Opsi mesin, pilih salah satu hal berikut:

   a. Perluas Tampilkan filter, lalu aktifkan Tampilkan versi yang mendukung fitur basis data global.

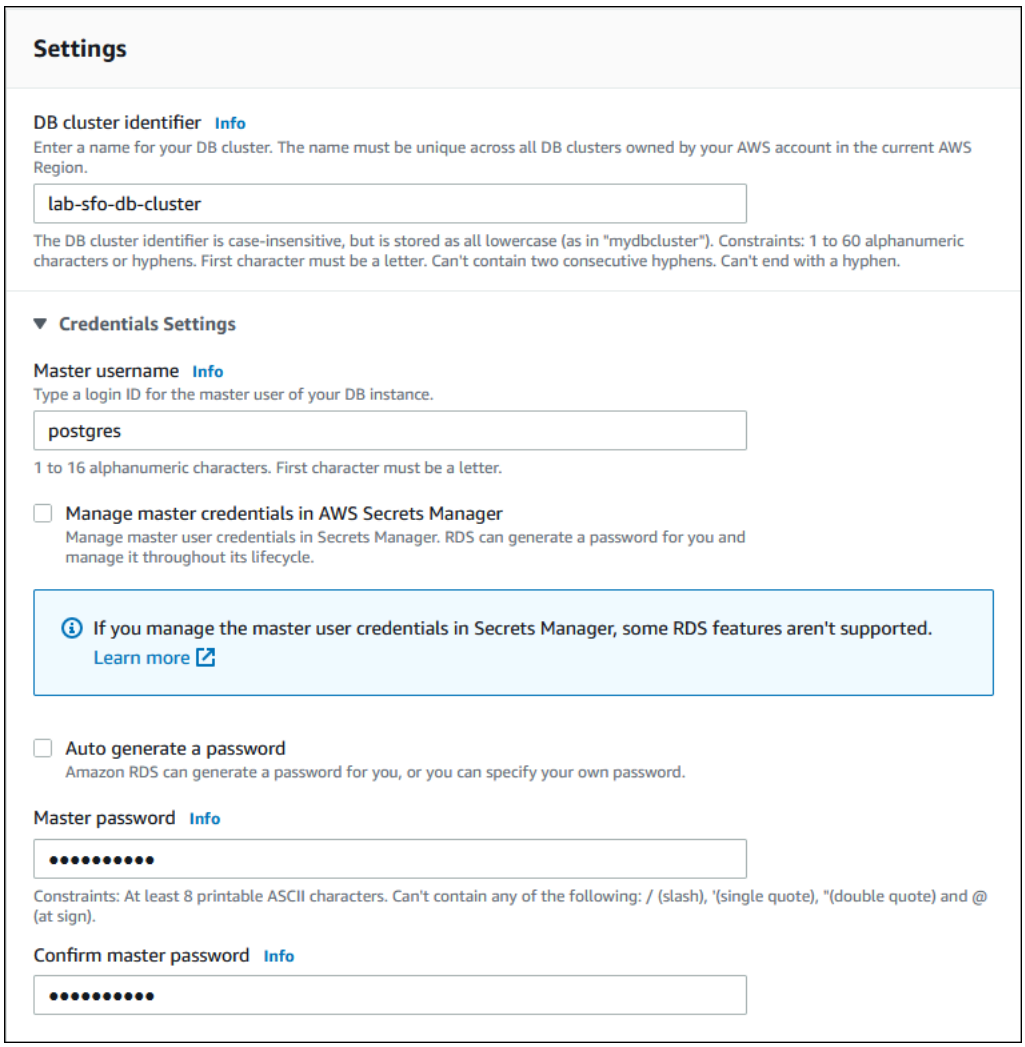
   b. Untuk Versi mesin, pilih versi Aurora PostgreSQL yang ingin Anda gunakan untuk basis data global Aurora Anda.



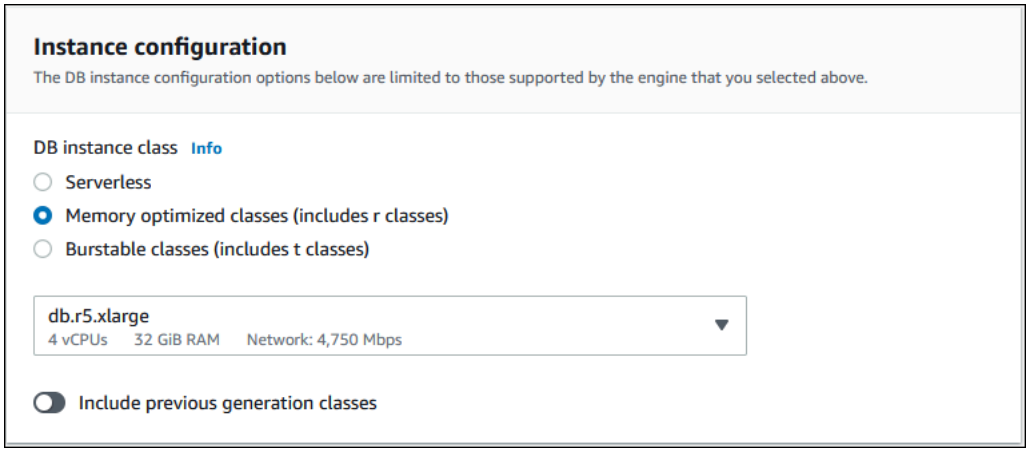
2. Untuk Templat, pilih Produksi. Atau Anda dapat memilih Dev/Tes jika sesuai. Jangan gunakan Dev/Tes dalam lingkungan produksi.
3. Untuk Pengaturan, lakukan hal berikut:

   a. Masukkan nama yang bermakna untuk pengidentifikasi klaster DB. Ketika Anda selesai membuat basis data global Aurora, nama ini mengidentifikasi klaster DB primer.

   b. Masukkan kata sandi Anda sendiri untuk akun admin default untuk klaster DB, atau minta Aurora membuatkannya untuk Anda. Jika Anda memilih untuk membuat kata sandi secara otomatis, Anda akan mendapatkan opsi untuk menyalin kata sandi.



4. Untuk Kelas instans DB, pilih `db.r5.large` atau kelas instans DB memori yang dioptimalkan lainnya. Kami menyarankan Anda menggunakan kelas instans db.r5 atau lebih tinggi.



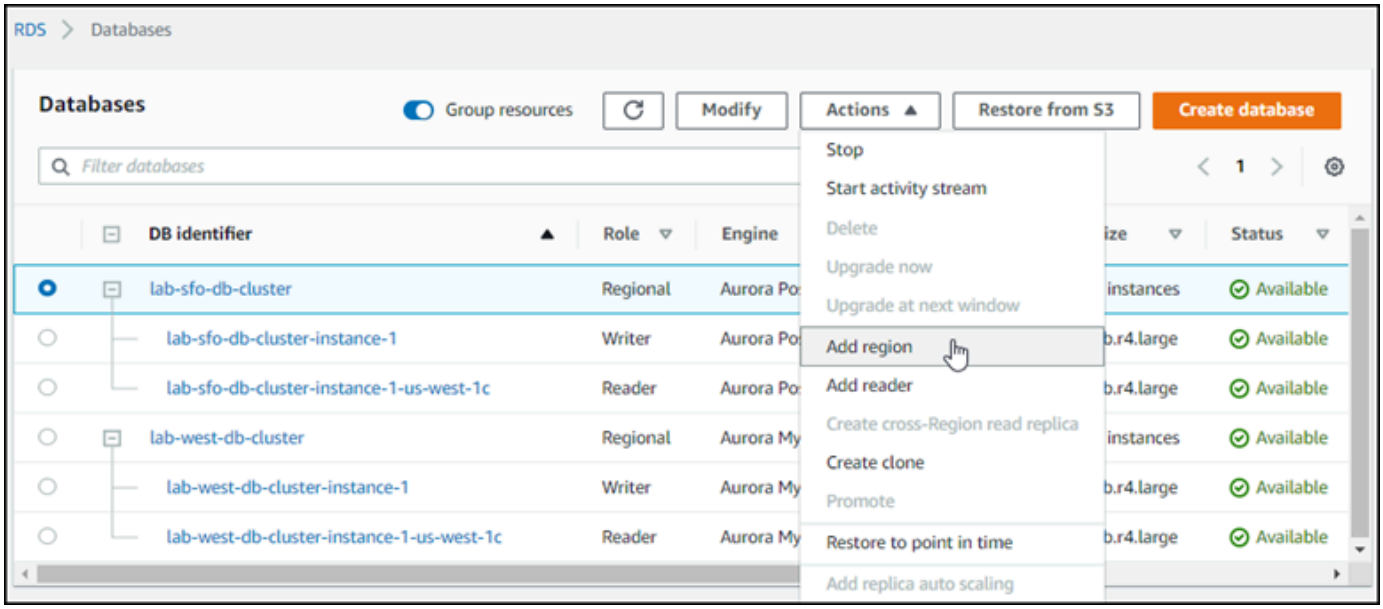
5. Untuk Ketersediaan & daya tahan, kami sarankan Anda memilih untuk meminta Aurora membuatkan Replika Aurora di AZ yang berbeda untuk Anda. Jika Anda tidak membuat Replika Aurora sekarang, Anda harus melakukannya nanti.
6. Untuk Konektivitas, pilih cloud privat virtual (VPC) berdasarkan Amazon VPC yang menetapkan lingkungan jaringan virtual untuk instans DB ini. Anda dapat memilih default untuk menyederhanakan tugas ini.
7. (Opsional) Lengkapi pengaturan Autentikasi basis data. Autentikasi kata sandi selalu diaktifkan. Untuk menyederhanakan proses, Anda dapat melewati bagian ini dan mengatur IAM atau kata sandi dan autentikasi Kerberos nanti.
8. Pada Konfigurasi tambahan, lakukan hal berikut:

   a. Masukkan nama untuk Nama basis data awal untuk membuat instans DB Aurora primer untuk klaster ini. Ini adalah simpul penulis untuk klaster DB Aurora primer.

      Tetap pilih default untuk grup parameter klaster DB dan grup parameter DB, kecuali Anda memiliki grup parameter kustom Anda sendiri yang ingin Anda gunakan.

   b. Terima semua pengaturan default lainnya untuk Konfigurasi tambahan, seperti Enkripsi, Ekspor log, dan sebagainya.
9. Pilih Buat basis data.

   Mungkin diperlukan beberapa menit bagi Aurora untuk menyelesaikan proses pembuatan instans DB Aurora, Replika Aurora, dan klaster DB Aurora. Ketika klaster siap untuk digunakan, klaster DB Aurora beserta simpul penulis dan replikanya akan menampilkan status Tersedia. Ini menjadi klaster DB Aurora primer dari basis data global Aurora Anda, setelah Anda menambahkan klaster sekunder.



Saat klaster DB primer Anda tersedia, buat satu atau beberapa klaster sekunder dengan mengikuti langkah-langkah dalam Menambahkan Wilayah AWS ke basis data global Amazon Aurora.

AWS CLI

Perintah AWS CLI dalam prosedur berikut menyelesaikan tugas-tugas berikut:

1. Buat basis data global Aurora, dengan memberikan nama dan menentukan jenis mesin basis data Aurora yang ingin Anda gunakan.
2. Buat klaster DB Aurora primer untuk basis data global Aurora.
3. Buat instans DB Aurora untuk klaster. Ini adalah klaster DB Aurora primer untuk basis data global.
4. Buat instans DB Aurora kedua untuk klaster DB Aurora. Ini adalah pembaca untuk melengkapi klaster DB Aurora.
5. Buat klaster DB Aurora kedua di Wilayah lain kemudian tambahkan ke basis data global Aurora Anda, dengan mengikuti langkah-langkah di Menambahkan Wilayah AWS ke basis data global Amazon Aurora.

Ikuti prosedur untuk mesin basis data Aurora Anda.

Membuat global basis data menggunakan Aurora MySQL

Membuat basis data global Aurora menggunakan Aurora MySQL

1. Gunakan perintah CLI `create-global-cluster`, yang meloloskan nama Wilayah AWS, mesin basis data Aurora, dan versi.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-global-cluster --region primary_region  \
       --global-cluster-identifier global_database_id \
       --engine aurora-mysql \
       --engine-version version # optional
   ```

   Untuk Windows:

   ```
   aws rds create-global-cluster ^
       --global-cluster-identifier global_database_id ^
       --engine aurora-mysql ^
       --engine-version version # optional
   ```

   Tindakan ini menciptakan basis data global Aurora “kosong”, hanya dengan nama (pengidentifikasi) dan mesin basis data Aurora. Perlu waktu beberapa menit agar basis data global Aurora tersedia. Sebelum menuju langkah berikutnya, gunakan perintah CLI `describe-global-clusters` untuk melihat apakah basis data tersebut tersedia.

   ```
   aws rds describe-global-clusters --region primary_region --global-cluster-
   identifier global_database_id
   ```

   Ketika basis data global Aurora tersedia, Anda dapat membuat klaster DB Aurora primer.

2. Untuk membuat klaster DB Aurora primer, gunakan perintah CLI `create-db-cluster`. Sertakan nama basis data global Aurora Anda dengan menggunakan parameter `--global-cluster-identifier`.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-cluster \
     --region primary_region \
     --db-cluster-identifier primary_db_cluster_id \
   ```

```
  --master-username userid \
  --master-user-password password \
  --engine aurora-mysql \
  --engine-version version \
  --global-cluster-identifier global_database_id
```

Untuk Windows:

```
aws rds create-db-cluster ^
  --region primary_region ^
  --db-cluster-identifier primary_db_cluster_id ^
  --master-username userid ^
  --master-user-password password ^
  --engine aurora-mysql ^
  --engine-version version ^
  --global-cluster-identifier global_database_id
```

Gunakan perintah `describe-db-clusters` AWS CLI untuk mengonfirmasi bahwa klaster DB Aurora sudah siap. Untuk memilih sebuah klaster DB Aurora tertentu, gunakan parameter `--db-cluster-identifier`. Atau Anda dapat meninggalkan nama klaster DB Aurora dalam perintah untuk mendapatkan detail tentang semua klaster DB Aurora Anda di Wilayah tertentu.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

Ketika respons menunjukkan `"Status": "available"` untuk klaster, berarti klaster sudah siap untuk digunakan.

3. Buat instans DB untuk klaster DB Aurora primer Anda. Untuk melakukannya, gunakan perintah CLI `create-db-instance`. Berikan nama klaster DB Aurora Anda dalam perintah, dan tentukan detail konfigurasi untuk instans. Anda tidak perlu memasukkan parameter `--master-username` dan `--master-user-password` dalam perintah, karena parameter tersebut diperoleh dari klaster DB Aurora.

   Untuk `--db-instance-class`, Anda hanya dapat menggunakan dari kelas dengan memori yang dioptimalkan, seperti `db.r5.large`. Kami menyarankan Anda menggunakan kelas instans db.r5 atau lebih tinggi. Untuk informasi tentang kelas-kelas ini, lihat kelas instans DB.

   Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier db_instance_id \
  --engine aurora-mysql \
  --engine-version version \
  --region primary_region
```

Untuk Windows:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier db_instance_id ^
  --engine aurora-mysql ^
  --engine-version version ^
  --region primary_region
```

Operasi `create-db-instance` mungkin membutuhkan beberapa waktu untuk diselesaikan. Periksa status untuk melihat apakah instans Aurora DB tersedia sebelum melanjutkan.

```
aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
```

Ketika perintah mengembalikan status “tersedia,” Anda dapat membuat instans DB Aurora lain untuk klaster DB primer Anda. Ini adalah instans pembaca (Aurora Replica) untuk klaster DB Aurora.

4. Untuk membuat instans DB Aurora lain untuk klaster tersebut, gunakan perintah CLI `create-db-instance`.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-instance \
     --db-cluster-identifier primary_db_cluster_id \
     --db-instance-class instance_class \
     --db-instance-identifier replica_db_instance_id \
     --engine aurora-mysql
   ```

   Untuk Windows:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier replica_db_instance_id ^
  --engine aurora-mysql
```

Ketika instans DB tersedia, replikasi dimulai dari simpul penulis ke replika. Sebelum melanjutkan, periksa bahwa instans DB tersedia dengan perintah CLI `describe-db-instances`.

Pada titik ini, Anda memiliki basis data global Aurora dengan klaster DB Aurora primer yang berisi instans DB penulis dan Replika Aurora. Anda sekarang dapat menambahkan klaster DB Aurora hanya-baca di Wilayah yang berbeda untuk menyelesaikan basis data global Aurora Anda. Untuk melakukannya, ikuti langkah yang ada di Menambahkan Wilayah AWS ke basis data global Amazon Aurora.

Membuat global basis data menggunakan Aurora PostgreSQL

Ketika Anda membuat objek Aurora untuk basis data global Aurora dengan menggunakan perintah berikut, mungkin diperlukan beberapa menit sampai setiap objek tersedia. Kami merekomendasikan bahwa setelah menyelesaikan perintah tertentu, Anda memeriksa status objek Aurora tertentu untuk memastikan statusnya tersedia.

Untuk melakukannya, gunakan perintah CLI `describe-global-clusters`.

```
aws rds describe-global-clusters --region primary_region
    --global-cluster-identifier global_database_id
```

Membuat basis data global Aurora menggunakan Aurora PostgreSQL

1. Gunakan perintah CLI `create-global-cluster`.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-global-cluster --region primary_region  \
       --global-cluster-identifier global_database_id \
       --engine aurora-postgresql \
       --engine-version version # optional
   ```

   Untuk Windows:

```
aws rds create-global-cluster ^
    --global-cluster-identifier global_database_id ^
    --engine aurora-postgresql ^
    --engine-version version # optional
```

Ketika basis data global Aurora tersedia, Anda dapat membuat klaster DB Aurora primer.

2. Untuk membuat klaster DB Aurora primer, gunakan perintah CLI `create-db-cluster`. Sertakan nama basis data global Aurora Anda dengan menggunakan parameter `--global-cluster-identifier`.

   Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
  --region primary_region \
  --db-cluster-identifier primary_db_cluster_id \
  --master-username userid \
  --master-user-password password \
  --engine aurora-postgresql \
  --engine-version version \
  --global-cluster-identifier global_database_id
```

   Untuk Windows:

```
aws rds create-db-cluster ^
  --region primary_region ^
  --db-cluster-identifier primary_db_cluster_id ^
  --master-username userid ^
  --master-user-password password ^
  --engine aurora-postgresql ^
  --engine-version version ^
  --global-cluster-identifier global_database_id
```

   Periksa apakah klaster DB Aurora sudah siap. Ketika respons dari perintah berikut menunjukkan `"Status": "available"` untuk klaster DB Aurora, Anda dapat melanjutkan.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier primary_db_cluster_id
```

3. Buat instans DB untuk klaster DB Aurora primer Anda. Untuk melakukannya, gunakan perintah CLI `create-db-instance`.

   Loloskan nama klaster DB Aurora Anda dengan parameter `--db-cluster-identifier`.

   Anda tidak perlu memasukkan parameter `--master-username` dan `--master-user-password` dalam perintah, karena parameter tersebut diperoleh dari klaster DB Aurora.

   Untuk `--db-instance-class`, Anda hanya dapat menggunakan dari kelas dengan memori yang dioptimalkan, seperti `db.r5.large`. Kami menyarankan Anda menggunakan kelas instans db.r5 atau lebih tinggi. Untuk informasi tentang kelas-kelas ini, lihat kelas instans DB.

   Untuk Linux, macOS, atau Unix:

   ```
   aws rds create-db-instance \
     --db-cluster-identifier primary_db_cluster_id \
     --db-instance-class instance_class \
     --db-instance-identifier db_instance_id \
     --engine aurora-postgresql \
     --engine-version version \
     --region primary_region
   ```

   Untuk Windows:

   ```
   aws rds create-db-instance ^
     --db-cluster-identifier primary_db_cluster_id ^
     --db-instance-class instance_class ^
     --db-instance-identifier db_instance_id ^
     --engine aurora-postgresql ^
     --engine-version version ^
     --region primary_region
   ```

4. Periksa status instans Aurora DB sebelum melanjutkan.

   ```
   aws rds describe-db-clusters --db-cluster-identifier primary_db_cluster_id
   ```

   Jika respons menunjukkan bahwa status instans DB Aurora adalah “tersedia,” Anda dapat membuat instans DB Aurora lain untuk klaster DB primer Anda.

5. Untuk membuat Replika Aurora untuk klaster DB Aurora, gunakan perintah CLI `create-db-instance`.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-instance \
  --db-cluster-identifier primary_db_cluster_id \
  --db-instance-class instance_class \
  --db-instance-identifier replica_db_instance_id \
  --engine aurora-postgresql
```

Untuk Windows:

```
aws rds create-db-instance ^
  --db-cluster-identifier primary_db_cluster_id ^
  --db-instance-class instance_class ^
  --db-instance-identifier replica_db_instance_id ^
  --engine aurora-postgresql
```

Ketika instans DB tersedia, replikasi dimulai dari simpul penulis ke replika. Sebelum melanjutkan, periksa bahwa instans DB tersedia dengan perintah CLI `describe-db-instances`.

Basis data global Aurora Anda ada, tetapi hanya memiliki Wilayah primernya dengan klaster DB Aurora yang terdiri dari instans DB penulis dan Replika Aurora. Anda sekarang dapat menambahkan klaster DB Aurora hanya-baca di Wilayah yang berbeda untuk menyelesaikan basis data global Aurora Anda. Untuk melakukannya, ikuti langkah yang ada di Menambahkan Wilayah AWS ke basis data global Amazon Aurora.

API RDS

Untuk membuat Aurora global database dengan RDS API, jalankan operasi CreateGlobalCluster .

Menambahkan Wilayah AWS ke basis data global Amazon Aurora

Basis data global Aurora membutuhkan setidaknya satu klaster DB Aurora sekunder di Wilayah AWS yang berbeda dari klaster DB Aurora primer. Anda dapat melampirkan hingga lima klaster DB sekunder ke basis data global Aurora Anda. Untuk setiap klaster DB sekunder yang Anda tambahkan ke basis data global Aurora Anda, kurangi jumlah Replika Aurora yang diperbolehkan untuk klaster DB primer sebanyak satu buah.

Sebagai contoh, jika basis data global Aurora Anda memiliki 5 Region sekunder, klaster DB primer Anda dapat memiliki hanya 10 (bukan 15) Replika Aurora. Untuk informasi selengkapnya, lihat Persyaratan konfigurasi basis data global Amazon Aurora.

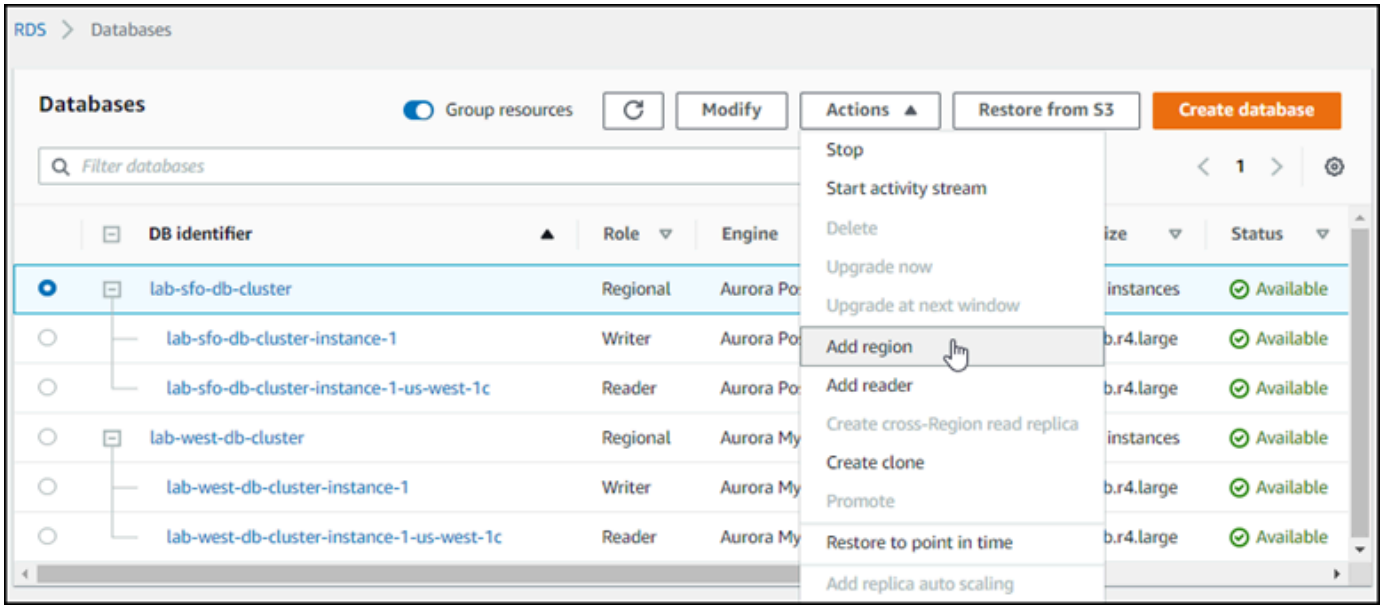
Jumlah Replika Aurora (instans pembaca) di klaster DB primer menentukan jumlah klaster DB sekunder yang dapat Anda tambahkan. Jumlah total instans pembaca dalam klaster DB primer ditambah jumlah klaster sekunder tidak boleh lebih dari 15. Misalnya, jika Anda memiliki 14 instans pembaca di klaster DB primer dan 1 klaster sekunder, Anda tidak dapat menambahkan klaster sekunder lainnya ke basis data global.

> **Note**
>
> Untuk Aurora MySQL versi 3, saat Anda membuat klaster sekunder, pastikan nilai `lower_case_table_names` cocok dengan nilai di klaster primer. Pengaturan ini adalah parameter basis data yang memengaruhi cara server menangani sensitivitas huruf besar/kecil pengidentifikasi. Untuk informasi selengkapnya tentang parameter basis data, lihat Bekerja dengan grup parameter.
>
> Kami menyarankan ketika Anda membuat klaster sekunder, gunakan versi mesin DB yang sama untuk primer dan sekunder. Jika perlu, tingkatkan klaster primer menjadi versi yang sama dengan versi sekunder. Untuk informasi selengkapnya, lihat Kompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola.

Konsol

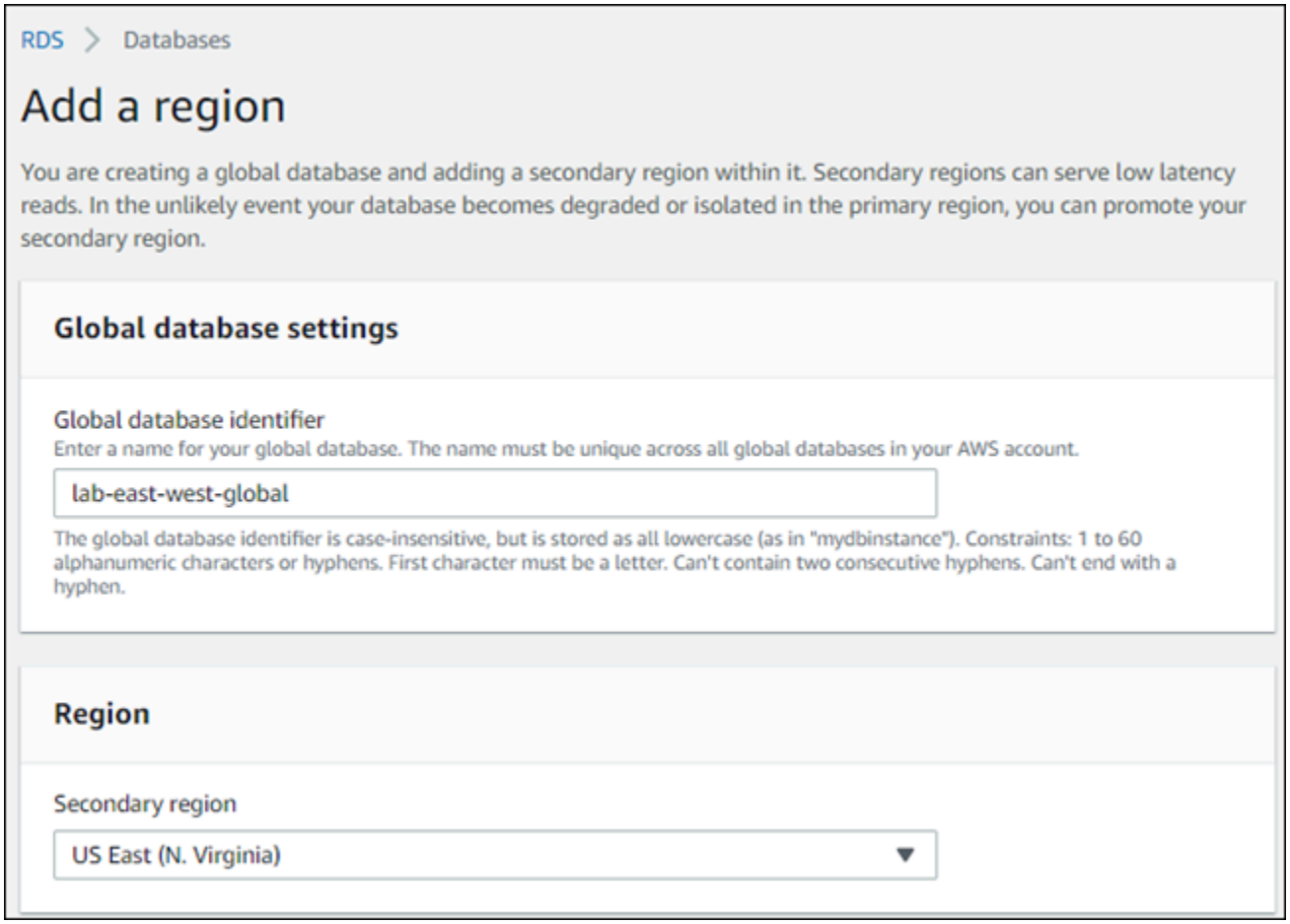
Menambahkan Wilayah AWS ke basis data global Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi AWS Management Console, pilih Basis data.
3. Pilih basis data global Aurora yang membutuhkan klaster DB Aurora sekunder. Pastikan klaster DB Aurora primer berstatus `Available`.
4. Untuk Tindakan, pilih Tambahkan wilayah.



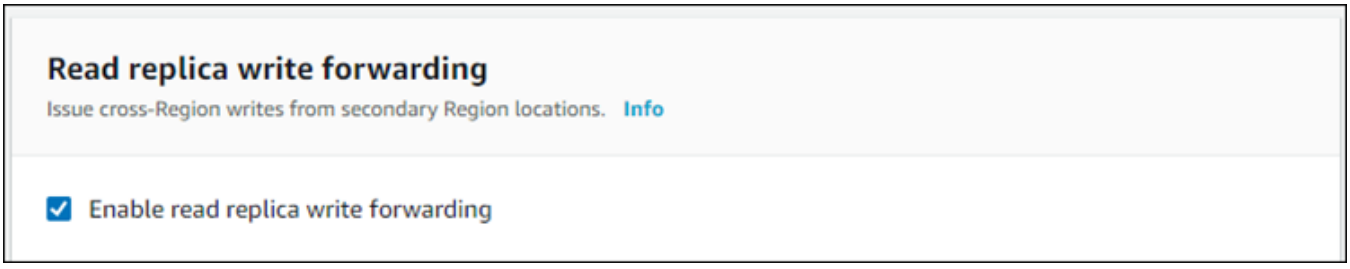
5. Di halaman Tambahkan wilayah, pilih Wilayah AWS sekunder.

   Anda tidak dapat memilih Wilayah AWS yang sudah memiliki klaster DB Aurora sekunder untuk basis data global Aurora yang sama. Selain itu, Wilayah tersebut tidak boleh sama dengan Wilayah klaster DB Aurora primer.



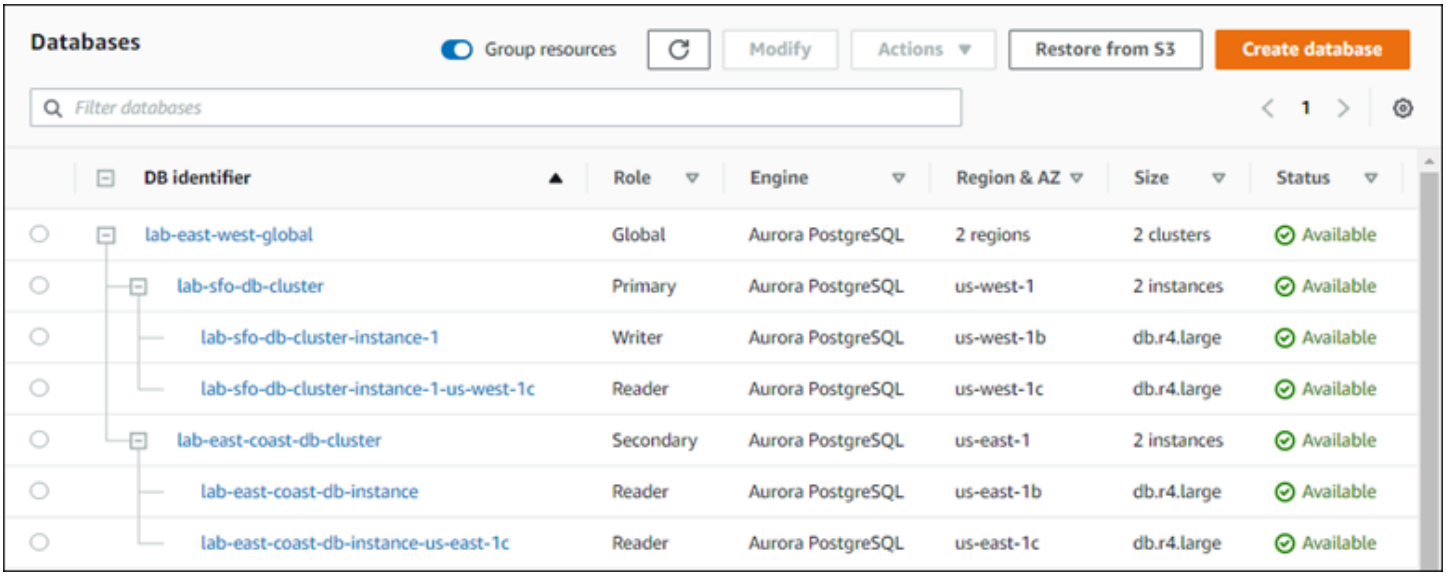
6. Lengkapi bidang yang tersisa untuk klaster Aurora sekunder di Wilayah AWS baru. Opsi konfigurasi ini sama seperti opsi konfigurasi untuk setiap instans klaster DB Aurora, kecuali opsi berikut yang ditujukan khusus untuk basis data global berbasis Aurora MySQL:

   - Aktifkan penerusan tulis replika baca - Pengaturan opsional ini memungkinkan klaster DB sekunder milik basis data global Aurora Anda meneruskan operasi tulis ke klaster primer. Untuk informasi selengkapnya, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.



7. Pilih Tambah wilayah.

Setelah Anda selesai menambahkan Wilayah ke basis data global Aurora Anda, Anda dapat melihatnya dalam daftar Basis data di AWS Management Console seperti yang ditunjukkan dalam tangkapan layar.



AWS CLI

Menambahkan Wilayah AWS sekunder ke basis data global Aurora

1. Gunakan perintah CLI `create-db-cluster` dengan nama (`--global-cluster-identifier`) basis data global Aurora Anda. Untuk parameter lainnya, lakukan hal berikut:
2. Untuk `--region`, pilih Wilayah AWS yang berbeda dari yang dimiliki Wilayah primer Aurora Anda.
3. Pilih nilai spesifik untuk parameter `--engine` dan `--engine-version`. Nilai-nilai tersebut sama dengan nilai untuk klaster DB Aurora primer dalam basis data global Aurora Anda.
4. Untuk klaster terenkripsi, tentukan Wilayah AWS primer Anda sebagai `--source-region` untuk enkripsi.

Contoh berikut ini membuat klaster DB Aurora baru dan melampirkannya ke basis data global Aurora sebagai klaster DB Aurora sekunder hanya-baca. Pada langkah terakhir, instans DB Aurora ditambahkan ke klaster DB Aurora baru.

Untuk Linux, macOS, atau Unix:

```
aws rds --region secondary_region \
  create-db-cluster \
```

```
    --db-cluster-identifier secondary_cluster_id \
    --global-cluster-identifier global_database_id \
    --engine aurora-mysql|aurora-postgresql
    --engine-version version

aws rds --region secondary_region \
  create-db-instance \
    --db-instance-class instance_class \
    --db-cluster-identifier secondary_cluster_id \
    --db-instance-identifier db_instance_id \
    --engine aurora-mysql|aurora-postgresql
```

Untuk Windows:

```
aws rds --region secondary_region ^
  create-db-cluster ^
    --db-cluster-identifier secondary_cluster_id ^
    --global-cluster-identifier global_database_id_id ^
    --engine aurora-mysql|aurora-postgresql ^
    --engine-version version

aws rds --region secondary_region ^
  create-db-instance ^
    --db-instance-class instance_class ^
    --db-cluster-identifier secondary_cluster_id ^
    --db-instance-identifier db_instance_id ^
    --engine aurora-mysql|aurora-postgresql
```

API RDS

Untuk menambahkan Wilayah AWS ke basis data global Aurora dengan API RDS, jalankan operasi CreateDBCluster. Tentukan pengidentifikasi basis data global yang sudah ada dengan menggunakan parameter `GlobalClusterIdentifier`.

Membuat klaster DB Aurora tanpa kepala di Wilayah sekunder

Meskipun basis data global Aurora membutuhkan setidaknya satu klaster DB Aurora sekunder di Wilayah AWS yang berbeda dari klaster primer, Anda dapat menggunakan konfigurasi tanpa kepala untuk klaster sekunder. Klaster DB Aurora sekunder tanpa kepala adalah klaster DB Aurora sekunder yang tidak memiliki instans DB. Jenis konfigurasi ini dapat menurunkan biaya untuk basis data global Aurora. Dalam klaster DB Aurora, komputasi dan penyimpanan dipisahkan. Tanpa instans DB, Anda

tidak dikenakan biaya untuk komputasi, hanya untuk penyimpanan. Jika diatur dengan benar, volume penyimpanan sekunder tanpa kepala tetap sinkron dengan klaster DB Aurora primer.

Anda menambahkan klaster sekunder seperti yang biasanya Anda lakukan saat membuat basis data global Aurora. Namun, setelah klaster DB Aurora primer memulai replikasi ke klaster sekunder, Anda menghapus instans DB hanya-baca Aurora dari klaster DB Aurora sekunder. Klaster sekunder ini sekarang dianggap “tanpa kepala” karena sudah tidak memiliki instans DB. Namun, volume penyimpanan tetap sinkron dengan klaster DB Aurora primer.

> **Warning**
>
> Dengan Aurora PostgreSQL, untuk membuat klaster tanpa kepala di Wilayah AWS sekunder, gunakan AWS CLI atau API RDS untuk menambahkan Wilayah AWS sekunder. Lewati langkah ini untuk membuat instans DB pembaca untuk klaster sekunder. Saat ini, membuat klaster tanpa kepala tidak didukung di konsol RDS. Untuk prosedur CLI dan API yang akan digunakan, lihat Menambahkan Wilayah AWS ke basis data global Amazon Aurora.
> Jika basis data global Anda menggunakan versi mesin yang lebih rendah dari 13.4, 12.8, atau 11.13, membuat instans DB pembaca di Wilayah sekunder kemudian menghapusnya dapat menyebabkan masalah vakum Aurora PostgreSQL pada instans DB penulis di Wilayah primer. Jika Anda mengalami masalah ini, mulai ulang instans DB penulis Wilayah primer setelah Anda menghapus instans DB pembaca sekunder Wilayah tersebut.

Menambahkan klaster DB Aurora sekunder tanpa kepala ke basis data global Aurora Anda

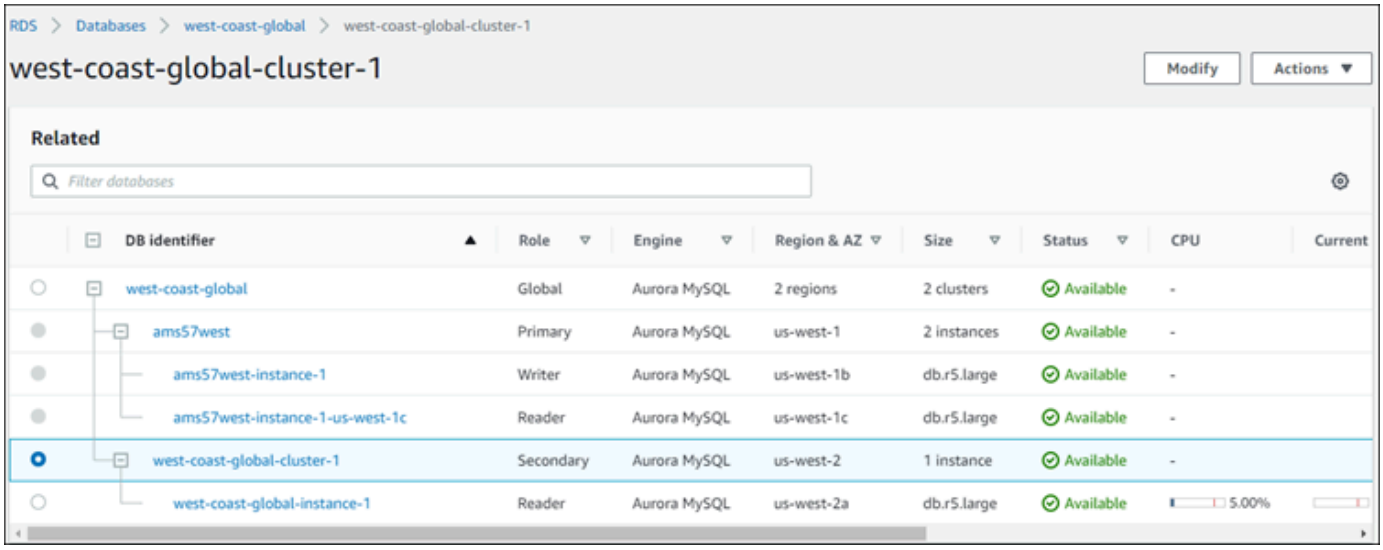
1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi AWS Management Console, pilih Basis data.
3. Pilih basis data global Aurora yang membutuhkan klaster DB Aurora sekunder. Pastikan klaster DB Aurora primer berstatus `Available`.
4. Untuk Tindakan, pilih Tambahkan wilayah.
5. Di halaman Tambahkan wilayah, pilih Wilayah AWS sekunder.

   Anda tidak dapat memilih Wilayah AWS yang sudah memiliki klaster DB Aurora sekunder untuk basis data global Aurora yang sama. Selain itu, Wilayah tersebut tidak boleh sama dengan Wilayah klaster DB Aurora primer.

6. Lengkapi bidang yang tersisa untuk klaster Aurora sekunder di Wilayah AWS baru. Opsi konfigurasi ini sama dengan opsi konfigurasi untuk setiap instans klaster DB Aurora.

   Untuk basis data global Aurora berbasis Aurora MySQL, abaikan opsi Aktifkan penerusan tulis replika baca. Opsi ini tidak memiliki fungsi setelah Anda menghapus instans pembaca.

7. Pilih Tambah wilayah. Setelah Anda selesai menambahkan Wilayah ke basis data global Aurora Anda, Anda dapat melihatnya dalam daftar Basis data di AWS Management Console seperti yang ditunjukkan dalam tangkapan layar.

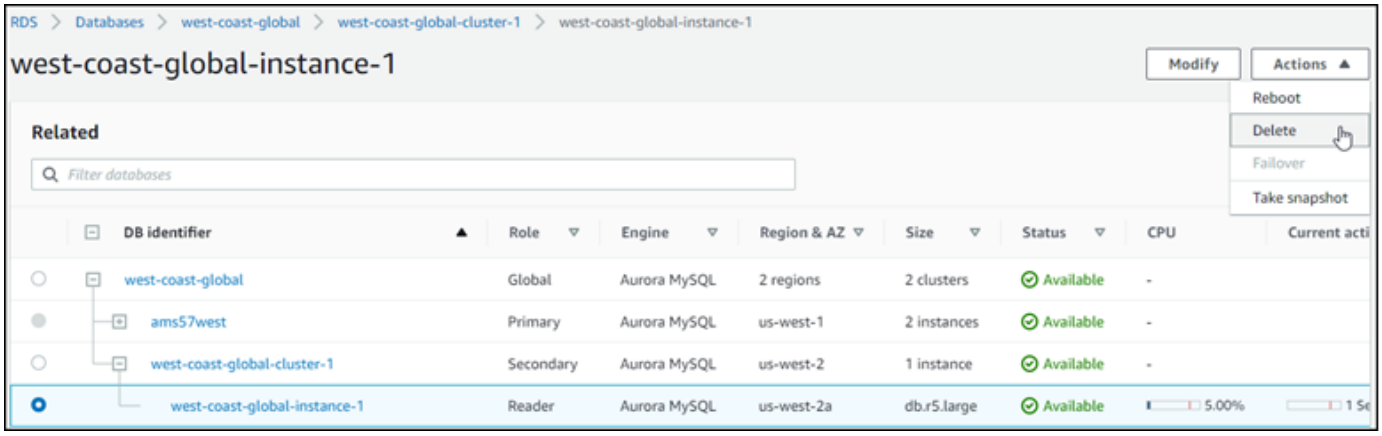


8. Periksa status klaster DB Aurora sekunder dan instans pembacanya sebelum melanjutkan, dengan menggunakan AWS Management Console atau AWS CLI. Sebagai contoh:

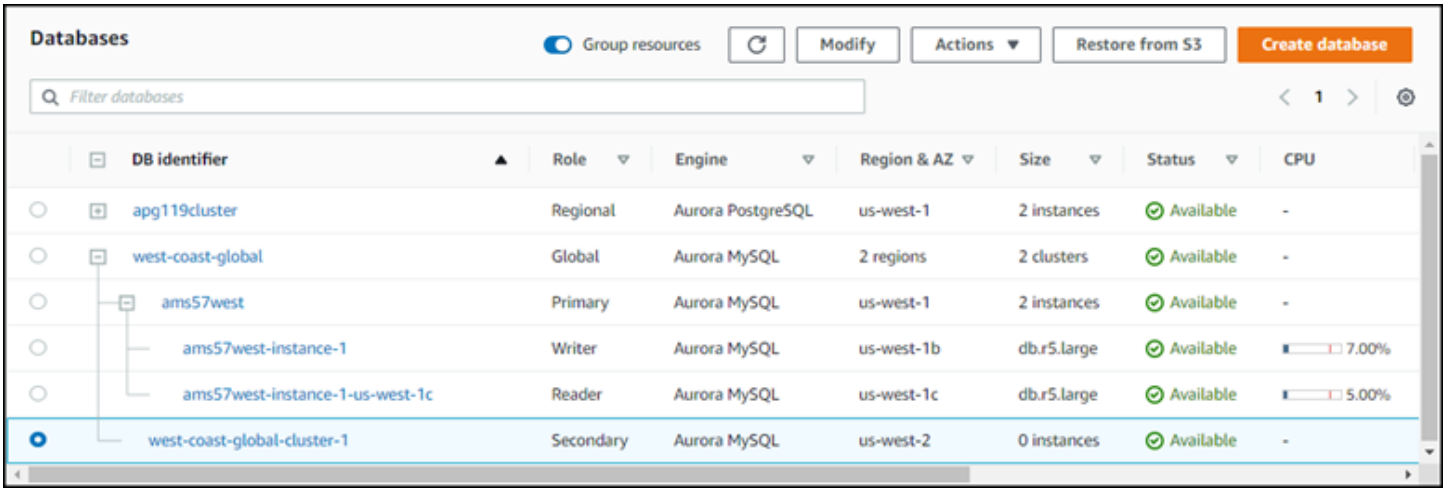
```
$ aws rds describe-db-clusters --db-cluster-identifier secondary-cluster-id --query
 '*[].[Status]' --output text
```

   Mungkin diperlukan waktu beberapa menit hingga status klaster DB Aurora sekunder yang baru ditambahkan berubah dari `creating` menjadi `available`. Ketika klaster DB Aurora tersedia, Anda dapat menghapus instans pembaca.

9. Pilih instans pembaca di klaster DB Aurora sekunder, kemudian pilih Hapus.



Setelah menghapus instans pembaca, klaster sekunder tetap menjadi bagian basis data global Aurora. Klaster tersebut tidak memiliki instans yang terkait dengannya, seperti yang ditunjukkan berikut ini.



Anda dapat menggunakan klaster DB Aurora sekunder tanpa kepala ini untuk secara manual memulihkan basis data global Amazon Aurora Anda dari pemadaman yang tidak direncanakan di Wilayah AWS primer jika terjadi pemadaman seperti itu.

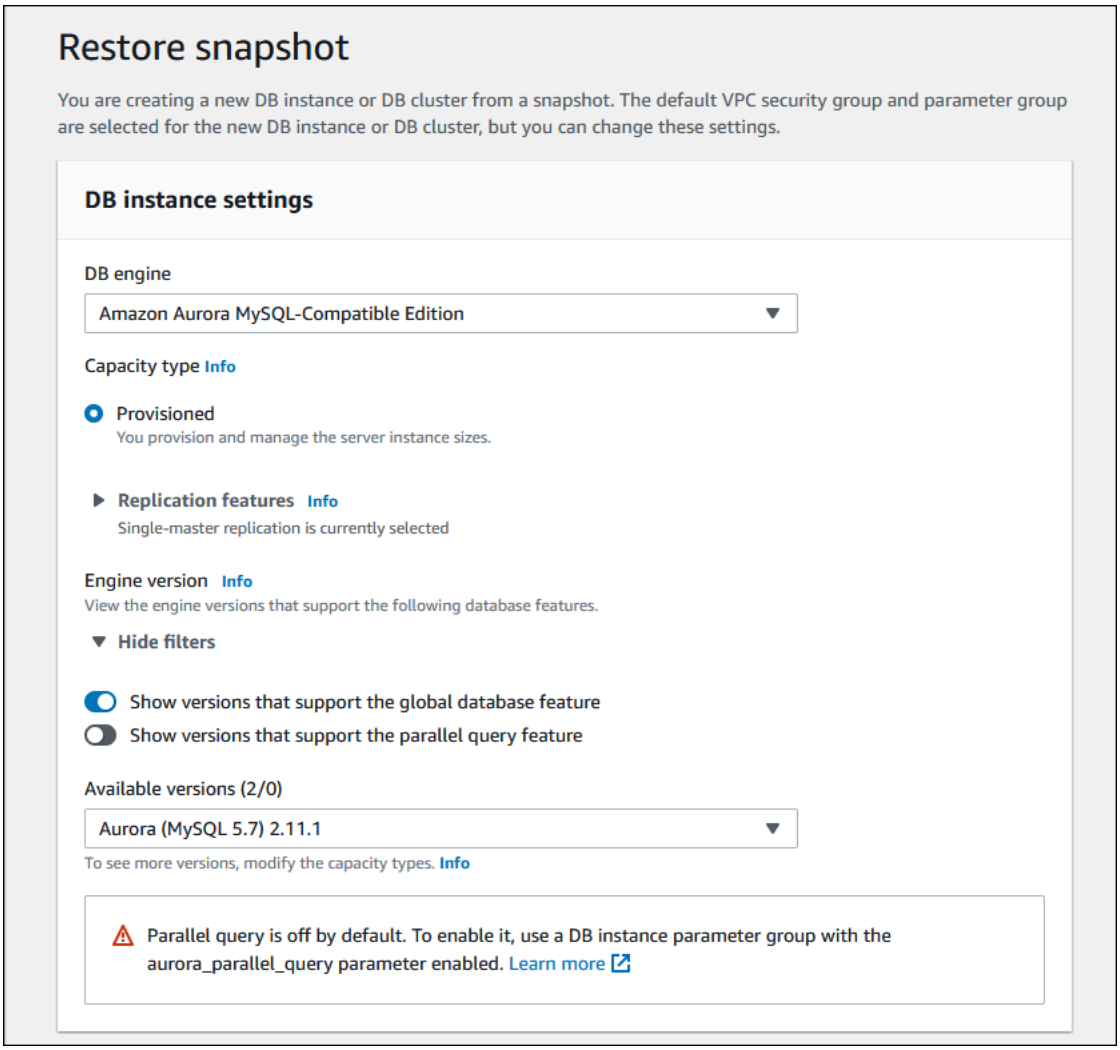
Menggunakan snapshot untuk basis data global Amazon Aurora Anda

Anda dapat memulihkan snapshot dari klaster DB Aurora atau dari instans DB Amazon RDS untuk digunakan sebagai titik awal basis data global Aurora Anda. Anda memulihkan snapshot dan membuat klaster DB baru yang disediakan Aurora pada waktu yang sama. Anda kemudian menambahkan Wilayah AWS lainnya ke klaster DB yang dipulihkan, sehingga mengubahnya menjadi basis data global Aurora. Klaster DB Aurora apa pun yang Anda buat menggunakan snapshot dengan cara ini akan menjadi klaster primer basis data global Aurora Anda.

Snapshot yang Anda gunakan dapat berasal dari klaster DB Aurora `provisioned` atau `serverless`.

Selama proses pemulihan, pilih jenis mesin DB yang sama dengan snapshot. Misalnya, anggaplah Anda ingin memulihkan snapshot yang dibuat dari klaster DB Aurora Serverless yang menjalankan Aurora PostgreSQL. Dalam hal ini, Anda membuat klaster DB Aurora PostgreSQL menggunakan mesin dan versi DB Aurora yang sama.

Klaster DB yang dipulihkan mengambil peran klaster primer untuk basis data global Aurora saat Anda menambahkan Wilayah AWS ke sana. Semua data yang terkandung dalam klaster primer ini direplikasi ke setiap klaster sekunder yang Anda tambahkan ke basis data global Aurora Anda.



Mengelola basis data global Amazon Aurora

Anda melakukan sebagian besar operasi manajemen pada klaster individu yang membentuk basis data global Aurora. Saat Anda memilih Sumber daya terkait grup di halaman Basis data dalam konsol, Anda melihat klaster primer dan klaster sekunder yang dikelompokkan di bawah basis data global yang dikaitkan. Untuk menemukan Wilayah AWS tempat klaster DB basis data global berjalan, mesin dan versi DB Aurora, dan pengidentifikasi, gunakan tab Konfigurasi.

Proses failover basis data lintas wilayah tersedia untuk basis data global Aurora saja, dan tidak tersedia untuk klaster DB Aurora tunggal. Untuk mempelajari selengkapnya, lihat Menggunakan switchover atau failover dalam basis data global Amazon Aurora.

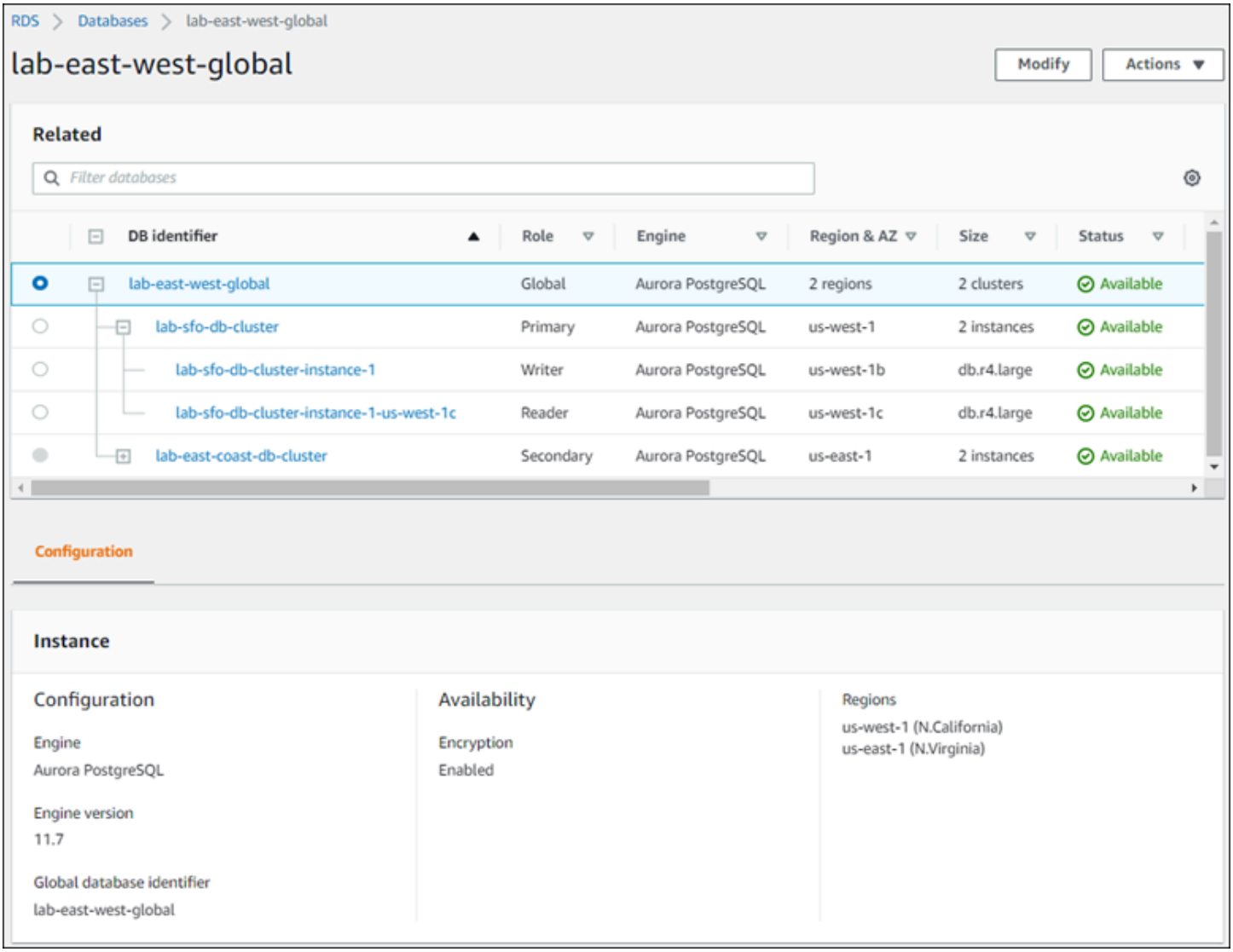
Untuk memulihkan basis data global Aurora dari pemadaman yang tidak direncanakan di Wilayah primernya, lihat Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan.

Topik

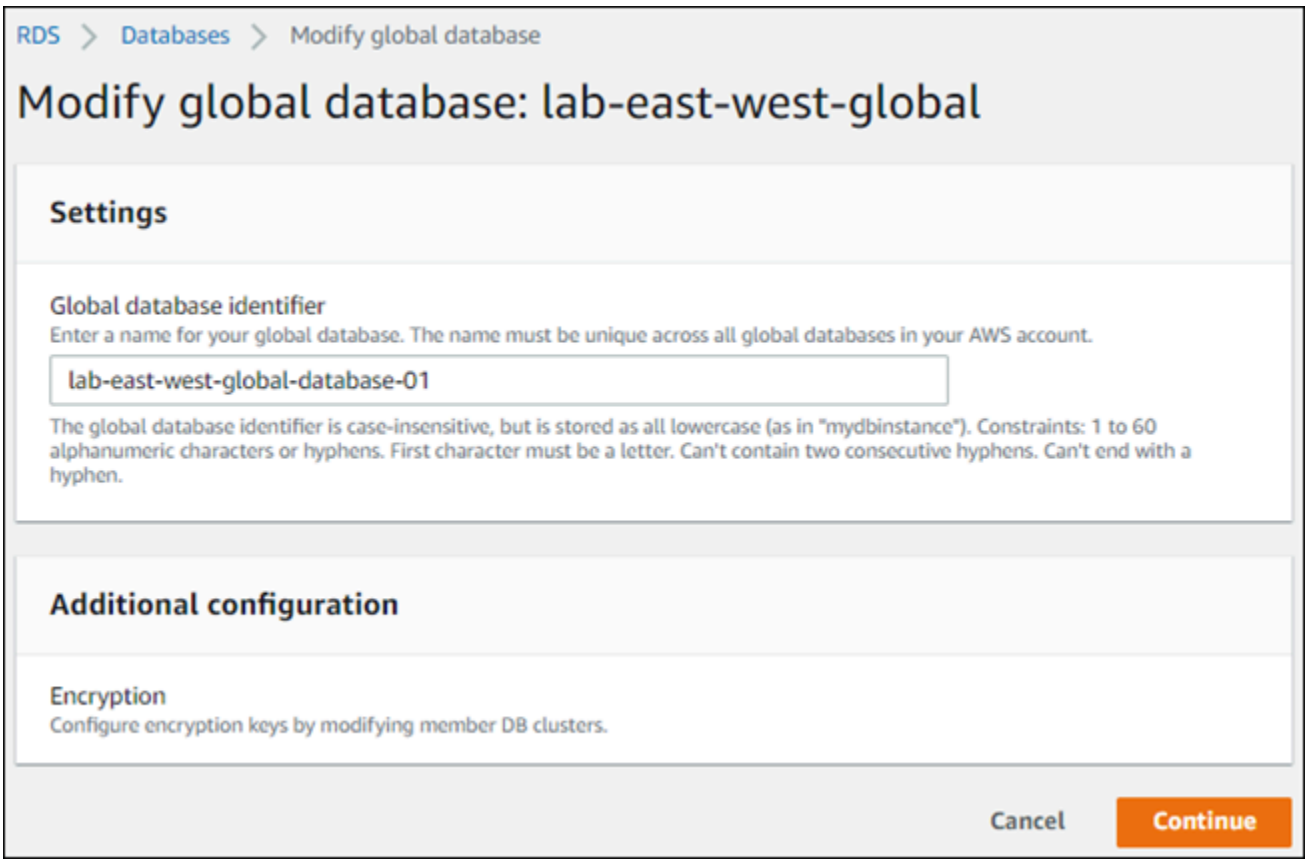
- Memodifikasi basis data global Amazon Aurora
- Memodifikasi parameter basis data global Aurora
- Menghapus klaster dari basis data global Amazon Aurora
- Menghapus basis data global Amazon Aurora

Memodifikasi basis data global Amazon Aurora

Halaman Basis data di AWS Management Console mencantumkan semua basis data global Aurora Anda, yang menunjukkan klaster primer dan klaster sekunder untuk masing-masing. Basis data global Aurora mempunyai pengaturan konfigurasinya sendiri. Secara khusus, terdapat Wilayah AWS yang dikaitkan dengan klaster primer dan sekunder, seperti yang ditunjukkan pada tangkapan layar berikut.



Ketika Anda membuat perubahan pada basis data global Aurora, Anda memiliki kesempatan untuk membatalkan perubahan, seperti yang ditunjukkan pada tangkapan layar berikut.



Ketika Anda memilih Lanjutkan, Anda mengonfirmasi perubahannya.

Memodifikasi parameter basis data global Aurora

Anda dapat mengonfigurasi grup parameter klaster DB Aurora secara independen untuk setiap klaster Aurora yang ada di dalam basis data global Aurora. Cara kerja sebagian besar parameter sama dengan jenis klaster Aurora lainnya. Kami menyarankan Anda menjaga pengaturan tetap konsisten di antara semua klaster di dalam basis data global. Tindakan tersebut membantu mencegah perubahan perilaku yang tidak terduga jika Anda menaikkan klaster sekunder menjadi klaster primer.

Misalnya, gunakan pengaturan zona waktu dan rangkaian karakter yang sama untuk mencegah perilaku yang tidak konsisten jika klaster yang berbeda mengambil alih peran sebagai klaster primer.

Pengaturan konfigurasi `aurora_enable_repl_bin_log_filtering` dan `aurora_enable_replica_log_compression` tidak berpengaruh.

Menghapus klaster dari basis data global Amazon Aurora

Anda dapat menghapus klaster DB Aurora dari basis data global Aurora Anda untuk beberapa alasan yang berbeda. Misalnya, Anda mungkin ingin menghapus klaster Aurora dari global basis data jika klaster primer tersebut terdegradasi atau terisolasi. Klaster tersebut kemudian menjadi klaster DB Aurora yang disediakan yang berdiri sendiri dan dapat digunakan untuk membuat basis data global Aurora baru. Untuk mempelajari selengkapnya, lihat Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan.

Anda juga mungkin ingin menghapus klaster DB Aurora karena Anda ingin menghapus basis data global Aurora yang sudah tidak Anda perlukan. Anda tidak dapat menghapus basis data global Aurora sampai Anda menghapus (melepaskan) semua klaster DB Aurora terkait, dan menyisakan klaster primer. Untuk informasi selengkapnya, lihat Menghapus basis data global Amazon Aurora.

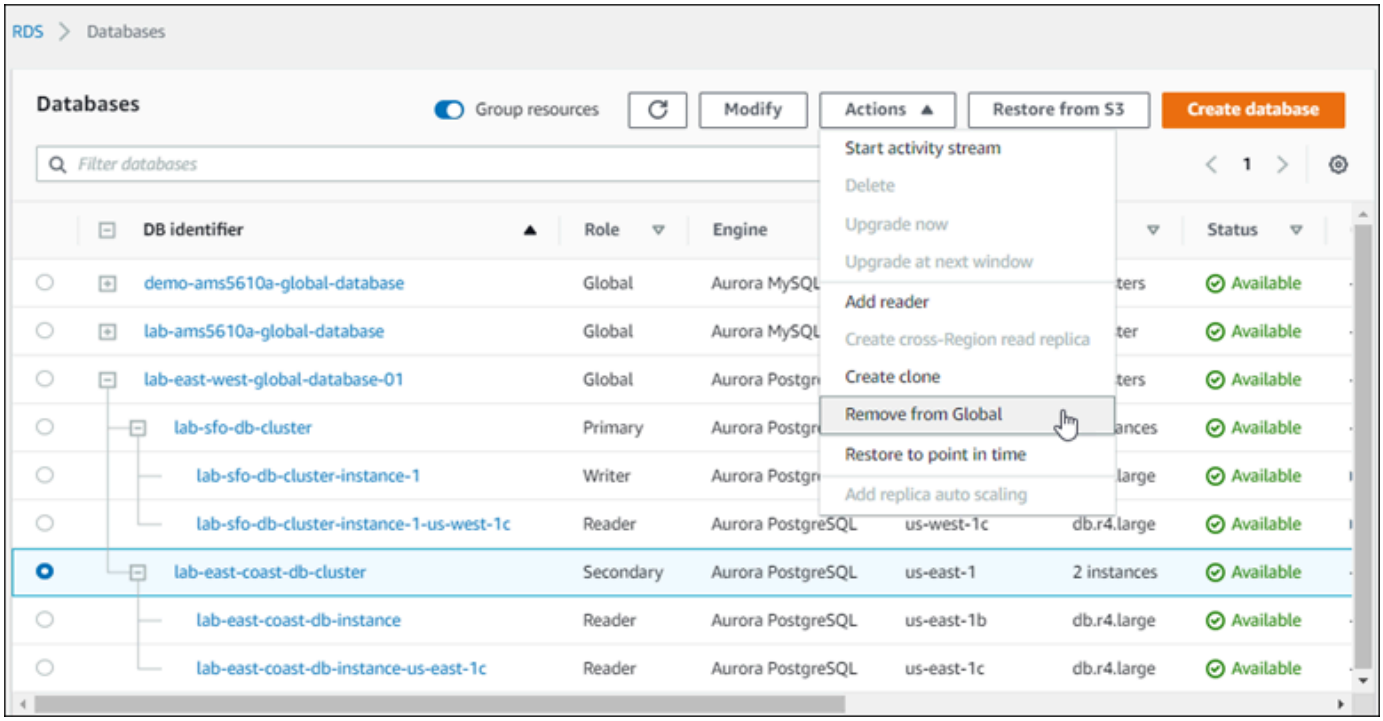
Ketika klaster DB Aurora dilepaskan dari basis data global Aurora, klaster tersebut tidak lagi disinkronkan dengan klaster primer. Klaster tersebut menjadi klaster DB Aurora yang berdiri sendiri dengan kapabilitas baca/tulis penuh.

Anda dapat menghapus klaster DB Aurora dari basis data global Aurora Anda menggunakan AWS Management Console, AWS CLI, atau API RDS.

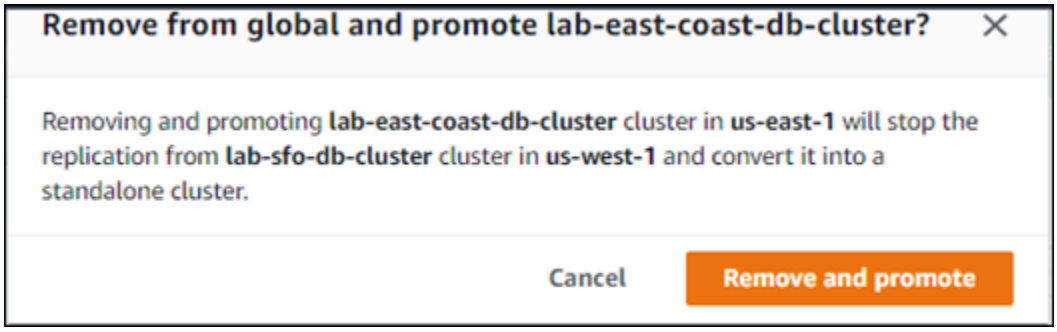
Konsol

Menghapus klaster Aurora dari basis data global Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih klaster pada halaman Basis data.
3. Untuk Tindakan, pilih Hapus dari Global.

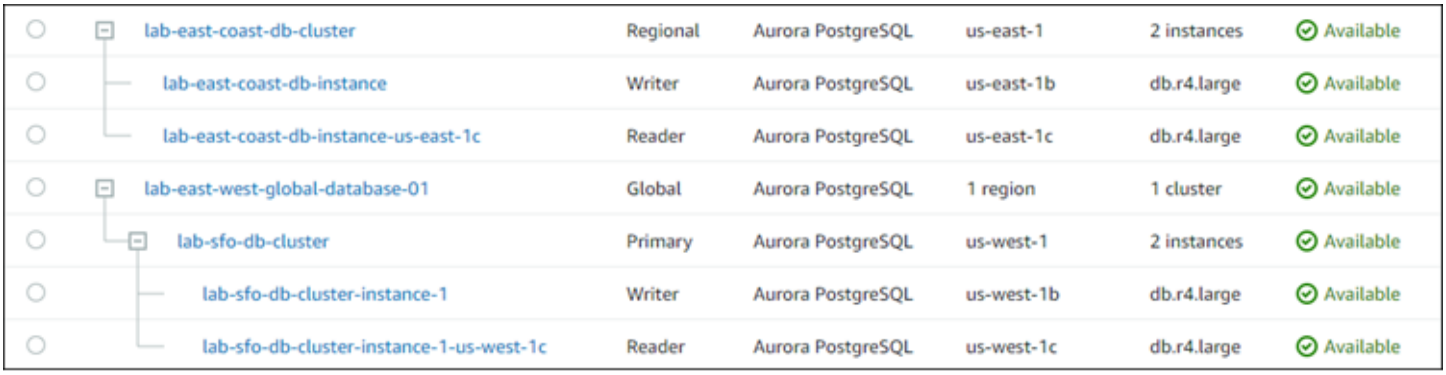


Anda melihat sebuah prompt untuk mengonfirmasi bahwa Anda ingin melepaskan klaster sekunder dari basis data global Aurora.



4. Pilih Hapus dan promosikan untuk menghapus klaster dari basis data global.

Klaster DB Aurora sudah tidak melayani sebagai klaster sekunder di dalam basis data global Aurora, dan sudah tidak disinkronkan dengan klaster DB primer. Ini adalah klaster DB Aurora yang berdiri sendiri dengan kapabilitas baca/tulis penuh.



Setelah Anda menghapus semua klaster sekunder, Anda dapat menghapus klaster primer dengan cara yang sama. Anda tidak dapat melepaskan (menghapus) klaster DB Aurora primer dari basis data global Aurora sampai Anda menghapus semua klaster sekunder.

Basis data global Aurora mungkin tetap berada dalam daftar Basis data, dengan nol Wilayah dan AZ. Anda dapat menghapusnya jika Anda sudah tidak ingin menggunakan basis data global Aurora ini. Untuk informasi selengkapnya, lihat Menghapus basis data global Amazon Aurora.

AWS CLI

Untuk menghapus cluster Aurora dari database global Aurora, jalankan perintah remove-from-global-clusterCLI dengan parameter berikut:

- `--global-cluster-identifier` — Nama (pengidentifikasi) basis data global Aurora Anda.
- `--db-cluster-identifier` — Nama setiap klaster DB Aurora yang akan dihapus dari basis data global Aurora. Hapus semua klaster DB Aurora sekunder sebelum menghapus klaster primer.

Contoh berikut ini terlebih dahulu menghapus klaster sekunder dan kemudian menghapus klaster primer dari basis data global Aurora.

Untuk Linux, macOS, atau Unix:

```
aws rds --region secondary_region \
  remove-from-global-cluster \
    --db-cluster-identifier secondary_cluster_ARN \
    --global-cluster-identifier global_database_id

aws rds --region primary_region \
  remove-from-global-cluster \
    --db-cluster-identifier primary_cluster_ARN \
```

```
    --global-cluster-identifier global_database_id
```

Ulangi perintah `remove-from-global-cluster --db-cluster-identifier` *`secondary_cluster_ARN`* untuk setiap Wilayah AWS sekunder di basis data global Aurora Anda.

Untuk Windows:

```
aws rds --region secondary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier secondary_cluster_ARN ^
    --global-cluster-identifier global_database_id

aws rds --region primary_region ^
  remove-from-global-cluster ^
    --db-cluster-identifier primary_cluster_ARN ^
    --global-cluster-identifier global_database_id
```

Ulangi perintah `remove-from-global-cluster --db-cluster-identifier` *`secondary_cluster_ARN`* untuk setiap Wilayah AWS sekunder di basis data global Aurora Anda.

API RDS

Untuk menghapus cluster Aurora dari Aurora global database dengan RDS API, jalankan tindakan RemoveFromGlobalCluster .

Menghapus basis data global Amazon Aurora

Karena basis data global Aurora biasanya menyimpan data penting bisnis, Anda tidak dapat menghapus basis data global dan klaster terkait dalam satu langkah. Untuk menghapus basis data global Aurora, lakukan hal berikut:

- Hapus klaster sekunder dari basis data global Aurora. Setiap klaster menjadi sebuah klaster DB Aurora yang berdiri sendiri. Untuk mempelajari caranya, lihat Menghapus klaster dari basis data global Amazon Aurora.
- Dari setiap klaster DB Aurora yang berdiri sendiri, hapus semua Replika Aurora.
- Hapus klaster DB primer dari basis data global Aurora. Klaster ini menjadi klaster DB Aurora yang berdiri sendiri.
- Dari klaster DB Aurora, pertama-tama hapus semua Replika Aurora, kemudian hapus instans DB penulis.

Menghapus instans penulis dari klaster DB Aurora baru yang berdiri sendiri juga biasanya menghapus klaster DB Aurora dan basis data global Aurora.

Untuk informasi umum selengkapnya, lihat Menghapus instans dari klaster DB Aurora.

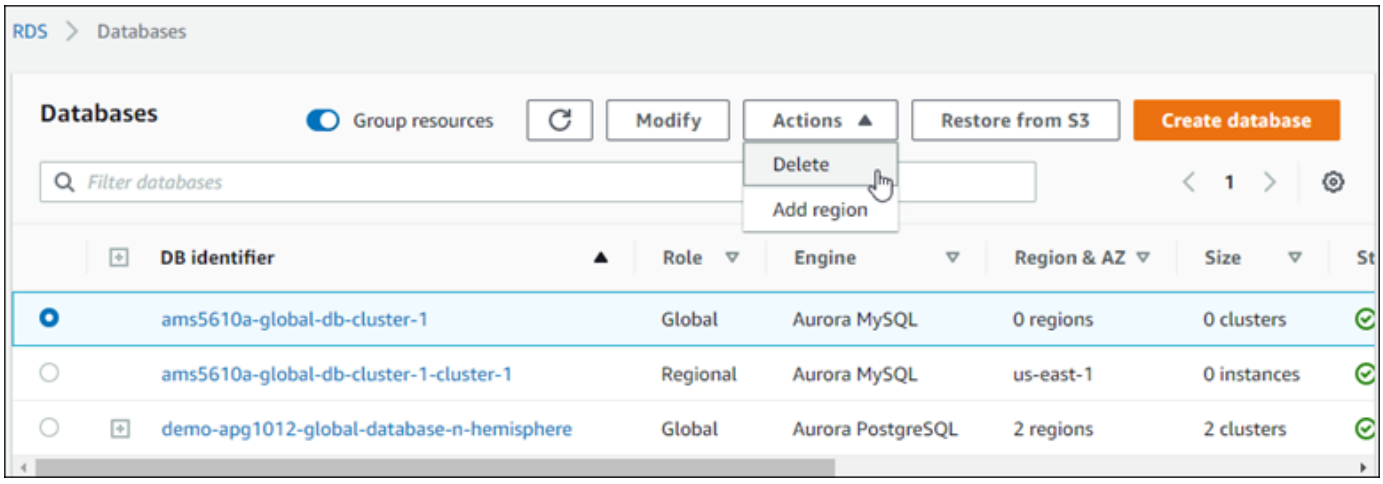
Untuk menghapus basis data global Aurora, Anda dapat menggunakan AWS Management Console, AWS CLI, atau API RDS.

Konsol

Menghapus basis data global Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data dan temukan basis data global Aurora yang ingin Anda hapus di daftar.
3. Konfirmasikan bahwa semua klaster lainnya telah dihapus dari basis data global Aurora. Basis data global Aurora harus menunjukkan 0 Wilayah dan AZ dan ukuran 0 klaster.

   Jika berisi klaster DB Aurora, basis data global Aurora tidak dapat dihapus. Jika perlu, lepaskan klaster DB Aurora primer dan sekunder dari basis data global Aurora. Untuk informasi selengkapnya, lihat Menghapus klaster dari basis data global Amazon Aurora.
4. Pilih basis data global Aurora Anda di dalam daftar, lalu pilih Hapus dari menu Tindakan.



AWS CLI

Untuk menghapus database global Aurora, jalankan perintah delete-global-clusterCLI dengan nama dan Wilayah AWS pengidentifikasi database global Aurora, seperti yang ditunjukkan pada contoh berikut.

Untuk Linux, macOS, atau Unix:

```
aws rds --region primary_region delete-global-cluster \
   --global-cluster-identifier global_database_id
```

Untuk Windows:

```
aws rds --region primary_region delete-global-cluster ^
   --global-cluster-identifier global_database_id
```

API RDS

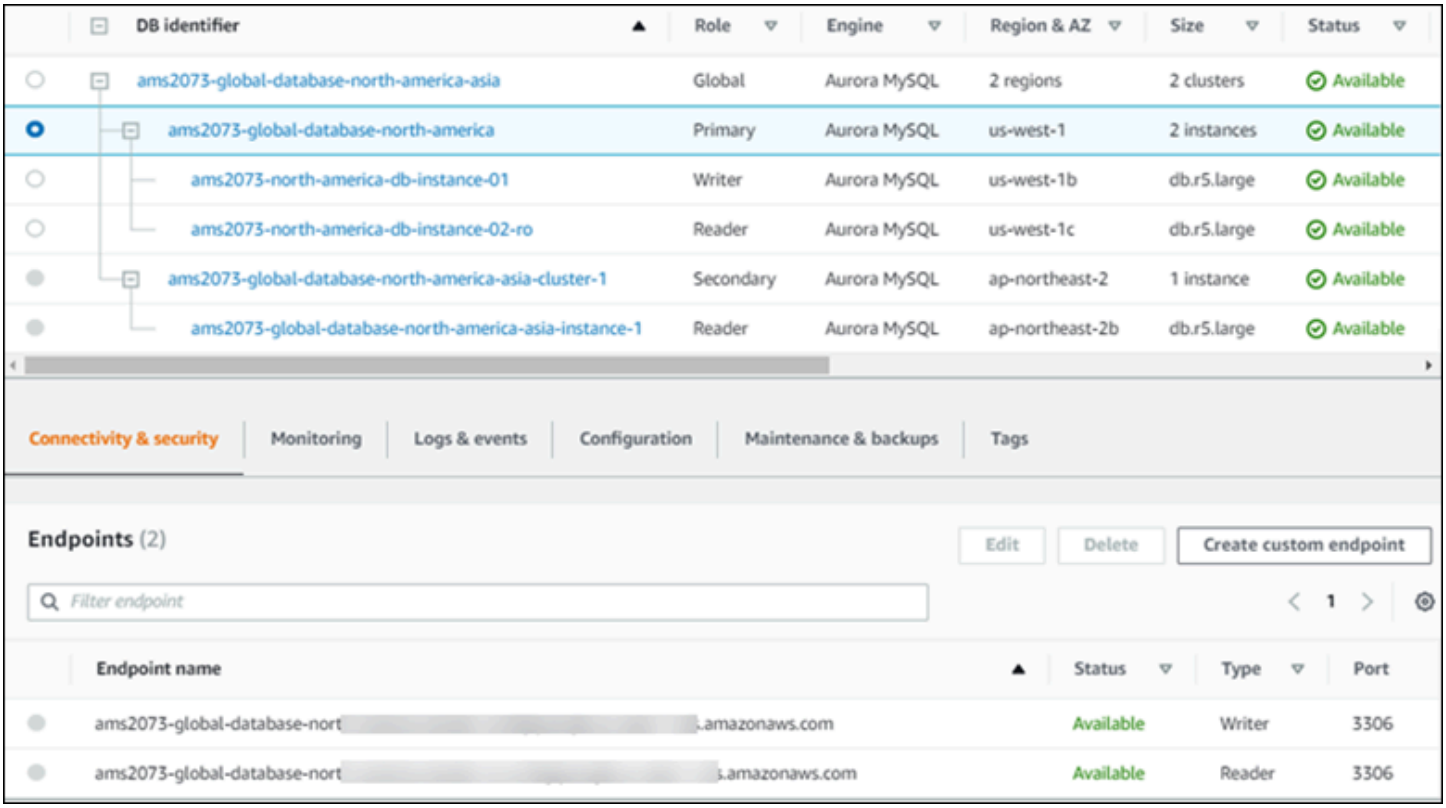
Untuk menghapus cluster yang merupakan bagian dari database global Aurora, jalankan operasi DeleteGlobalClusterAPI.

Terhubung ke basis data global Amazon Aurora

Cara menghubungkan ke basis data global Aurora tergantung pada apakah Anda perlu menulis ke basis data atau membaca dari basis data:

- Untuk permintaan atau kueri hanya-baca, Anda terhubung ke titik akhir pembaca untuk klaster Aurora di Wilayah AWS Anda.
- Untuk menjalankan pernyataan bahasa manipulasi data (DML) atau bahasa definisi data (DDL), Anda terhubung ke titik akhir klaster untuk klaster primer. Titik akhir ini mungkin berada di Wilayah AWS yang berbeda dengan aplikasi Anda.

Saat melihat basis data global Aurora di konsol, Anda dapat melihat semua titik akhir tujuan umum yang terkait dengan semua klasternya. Tangkapan layar berikut menunjukkan sebuah contoh. Terdapat titik akhir klaster tunggal yang terkait dengan klaster primer yang Anda gunakan untuk operasi tulis. Klaster primer dan setiap klaster sekunder memiliki titik akhir pembaca, yang Anda gunakan untuk kueri hanya-baca. Untuk mengurangi latensi, pilih titik akhir pembaca mana pun yang berada di Wilayah AWS Anda atau Wilayah AWS terdekat. Berikut adalah contoh Aurora MySQL.



Menggunakan penerusan menulis dalam basis data global Amazon Aurora

Anda dapat mengurangi jumlah titik akhir yang perlu dikelola untuk aplikasi yang berjalan di basis data global Aurora Anda, dengan menggunakan penerusan tulis. Dengan penerusan tulis diaktifkan, klaster sekunder dalam basis data global Aurora meneruskan pernyataan SQL yang melakukan operasi tulis ke klaster primer. Klaster primer memperbarui sumber kemudian menyebarkan perubahan yang dihasilkan kembali ke semua Wilayah AWS sekunder.

Dengan konfigurasi penerusan tulis, Anda tidak perlu menerapkan mekanisme Anda sendiri untuk mengirimkan operasi tulis dari Wilayah AWS sekunder ke Wilayah primer. Aurora menangani pengaturan jaringan lintas Wilayah. Aurora juga mengirimkan semua sesi yang diperlukan dan konteks transaksional untuk setiap pernyataan. Data selalu diubah terlebih dahulu pada klaster primer kemudian direplikasi ke klaster sekunder dalam basis data global Aurora. Dengan cara ini, klaster primer adalah sumber kebenaran dan selalu memiliki salinan terkini dari semua data Anda.

Topik

- Menggunakan penerusan tulis di dalam basis data global Aurora MySQL

- Menggunakan penerusan tulis di dalam basis data global Aurora PostgreSQL

Menggunakan penerusan tulis di dalam basis data global Aurora MySQL

Topik

- Ketersediaan wilayah dan versi penerusan tulis di Aurora MySQL
- Mengaktifkan penerusan tulis di Aurora MySQL
- Memeriksa apakah klaster sekunder telah mengaktifkan penerusan tulis di Aurora MySQL
- Kompatibilitas aplikasi dan SQL dengan penerusan tulis di Aurora MySQL
- Isolasi dan konsistensi untuk penerusan tulis di Aurora MySQL
- Menjalankan pernyataan multibagian dengan penerusan tulis di Aurora MySQL
- Transaksi dengan penerusan tulis di Aurora MySQL
- Parameter konfigurasi untuk penerusan tulis di Aurora MySQL
- Metrik Amazon CloudWatch untuk penerusan tulis di Aurora MySQL

Ketersediaan wilayah dan versi penerusan tulis di Aurora MySQL

Penerusan tulis didukung dengan Aurora MySQL 2.08.1 dan versi yang lebih tinggi, di setiap Wilayah di mana basis data global berbasis Aurora MySQL tersedia.

Untuk informasi tentang versi dan ketersediaan Wilayah basis data global Aurora MySQL, lihat Basis data global Aurora dengan Aurora MySQL.

Mengaktifkan penerusan tulis di Aurora MySQL

Secara default, penerusan tulis tidak diaktifkan saat Anda menambahkan klaster sekunder ke basis data global Aurora.

Untuk mengaktifkan penerusan tulis menggunakan AWS Management Console, pilih kotak centang Aktifkan penerusan tulis global di bagian Penerusan tulis replika baca saat Anda menambahkan Wilayah untuk basis data global. Untuk klaster sekunder yang sudah ada, ubah klaster tersebut untuk Mengaktifkan penerusan tulis global. Untuk menonaktifkan penerusan tulis, kosongkan Aktifkan penerusan tulis global saat menambahkan Wilayah atau memodifikasi klaster sekunder.

Untuk mengaktifkan penerusan tulis menggunakan AWS CLI, gunakan opsi `--enable-global-write-forwarding`. Opsi ini berfungsi saat Anda membuat klaster sekunder baru menggunakan

perintah `create-db-cluster`. Opsi ini juga berfungsi saat Anda memodifikasi klaster sekunder yang ada dengan menggunakan perintah `modify-db-cluster`. Opsi ini mengharuskan basis data global menggunakan versi Aurora yang mendukung penerusan tulis. Anda dapat menonaktifkan penerusan tulis dengan menggunakan opsi `--no-enable-global-write-forwarding` dengan perintah CLI yang sama ini.

Untuk mengaktifkan penerusan tulis menggunakan API Amazon RDS, atur parameter `EnableGlobalWriteForwarding` ke `true`. Parameter ini berfungsi saat Anda membuat klaster sekunder baru menggunakan operasi `CreateDBCluster`. Opsi ini juga berfungsi saat Anda memodifikasi klaster sekunder yang ada dengan menggunakan operasi `ModifyDBCluster`. Opsi ini mengharuskan basis data global menggunakan versi Aurora yang mendukung penerusan tulis. Anda dapat menonaktifkan penerusan tulis dengan mengatur parameter `EnableGlobalWriteForwarding` ke `false`.

> **Note**
>
> Agar sesi basis data memanfaatkan penerusan tulis, tentukan pengaturan untuk parameter konfigurasi `aurora_replica_read_consistency`. Lakukan hal ini di setiap sesi yang menggunakan fitur penerusan tulis. Untuk informasi tentang parameter ini, lihat Isolasi dan konsistensi untuk penerusan tulis di Aurora MySQL.
> Fitur RDS Proxy tidak mendukung nilai `SESSION` untuk variabel `aurora_replica_read_consistency`. Mengatur nilai ini dapat menghasilkan perilaku tak terduga.

Contoh CLI berikut menunjukkan bagaimana Anda dapat menyiapkan basis data global Aurora dengan penerusan tulis diaktifkan atau dinonaktifkan. Item yang disorot mewakili perintah dan opsi yang penting untuk menentukan dan menjaga infrastruktur yang konsisten untuk basis data global Aurora.

Contoh berikut membuat basis data global Aurora, klaster primer, dan klaster sekunder dengan penerusan tulis diaktifkan. Ganti pilihan Anda sendiri untuk nama pengguna, kata sandi, dan Wilayah AWS primer dan sekunder.

```
# Create overall global database.
aws rds create-global-cluster --global-cluster-identifier write-forwarding-test \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1
```

```
# Create primary cluster, in the same AWS Region as the global database.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-1 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --master-username user_name --master-user-password password \
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
  --db-instance-identifier write-forwarding-test-cluster-1-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-1 \
  --db-instance-identifier write-forwarding-test-cluster-1-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-1

# Create secondary cluster, in a different AWS Region than the global database,
# with write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2 \
  --enable-global-write-forwarding

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-east-2
```

Contoh berikut adalah lanjutan contoh sebelumnya. Contoh ini membuat klaster sekunder tanpa mengaktifkan penerusan tulis, kemudian mengaktifkan penerusan tulis. Setelah contoh ini selesai, semua klaster sekunder dalam basis data global mengaktifkan penerusan tulis.

```
# Create secondary cluster, in a different AWS Region than the global database,
# without write forwarding enabled.
aws rds create-db-cluster --global-cluster-identifier write-forwarding-test \
  --db-cluster-identifier write-forwarding-test-cluster-2 \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-1 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds create-db-instance --db-cluster-identifier write-forwarding-test-cluster-2 \
  --db-instance-identifier write-forwarding-test-cluster-2-instance-2 \
  --db-instance-class db.r5.large \
  --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.1 \
  --region us-west-1

aws rds modify-db-cluster --db-cluster-identifier write-forwarding-test-cluster-2 \
  --region us-east-2 \
  --enable-global-write-forwarding
```

Memeriksa apakah klaster sekunder telah mengaktifkan penerusan tulis di Aurora MySQL

Untuk menentukan apakah Anda dapat menggunakan penerusan tulis dari klaster sekunder, Anda dapat memeriksa apakah klaster tersebut memiliki atribut `"GlobalWriteForwardingStatus": "enabled"`.

Di AWS Management Console, pada tab Konfigurasi halaman detail untuk klaster, Anda melihat status Diaktifkan untuk Penerusan tulis replika baca global.

Untuk melihat status pengaturan penerusan tulis global untuk semua klaster Anda, jalankan perintah AWS CLI berikut.

Klaster sekunder menunjukkan nilai `"enabled"` atau `"disabled"` untuk menunjukkan apakah penerusan tulis diaktifkan atau dinonaktifkan. Nilai `null` menunjukkan bahwa penerusan tulis tidak tersedia untuk klaster tersebut. Klaster ini bukan bagian dari basis data global, atau merupakan klaster primer, bukan klaster sekunder. Nilainya juga bisa `"enabling"` atau `"disabling"` jika penerusan tulis sedang dalam proses diaktifkan atau dinonaktifkan.

Example

```
aws rds describe-db-clusters \
--query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu

[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

Untuk menemukan semua klaster sekunder yang memiliki penerusan tulis global, jalankan perintah berikut. Perintah ini juga mengembalikan titik akhir pembaca klaster ini. Anda menggunakan titik akhir pembaca klaster sekunder ketika Anda menggunakan penerusan tulis dari sekunder ke primer dalam basis data global Aurora Anda.

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
]
```

Kompatibilitas aplikasi dan SQL dengan penerusan tulis di Aurora MySQL

Anda dapat menggunakan jenis pernyataan SQL berikut dengan penerusan tulis:

- Pernyataan bahasa manipulasi data (DML), seperti `INSERT`, `DELETE`, dan `UPDATE`. Ada beberapa pembatasan pada properti pernyataan ini yang dapat Anda gunakan dengan penerusan tulis, seperti yang dijelaskan di bawah ini.
- Pernyataan `SELECT ... LOCK IN SHARE MODE` dan `SELECT FOR UPDATE`.
- Pernyataan `PREPARE` dan `EXECUTE`.

Pernyataan tertentu tidak diizinkan atau dapat menghasilkan hasil usang saat Anda menggunakannya dalam basis data global dengan penerusan tulis. Dengan demikian, pengaturan `EnableGlobalWriteForwarding` dinonaktifkan secara default untuk klaster sekunder. Sebelum mengaktifkan pengaturan tersebut, periksa untuk memastikan bahwa kode aplikasi Anda tidak terpengaruh oleh pembatasan ini.

Batasan berikut berlaku untuk pernyataan SQL yang Anda gunakan untuk penerusan tulis. Dalam beberapa kasus, Anda dapat menggunakan pernyataan pada klaster sekunder dengan penerusan tulis yang diaktifkan pada tingkat klaster. Pendekatan ini berfungsi jika penerusan tulis tidak diaktifkan dalam sesi oleh parameter konfigurasi `aurora_replica_read_consistency`. Mencoba menggunakan pernyataan saat tidak diizinkan karena penerusan tulis menyebabkan pesan kesalahan dengan format berikut.

```
ERROR 1235 (42000): This version of MySQL doesn't yet support 'operation with write
 forwarding'.
```

Bahasa definisi data (DDL)

Buat koneksi ke klaster primer untuk menjalankan pernyataan DDL. Anda tidak dapat menjalankannya dari instans DB pembaca.

Memperbarui tabel permanen menggunakan data dari tabel sementara

Anda dapat menggunakan tabel sementara pada klaster sekunder dengan penerusan tulis yang diaktifkan. Namun, Anda tidak dapat menggunakan pernyataan DML untuk mengubah tabel permanen jika pernyataan tersebut mengacu pada tabel sementara. Misalnya, Anda tidak dapat menggunakan pernyataan `INSERT ... SELECT` yang mengambil data dari tabel sementara. Tabel sementara ada pada klaster sekunder dan tidak tersedia ketika pernyataan berjalan pada klaster primer.

Transaksi XA

Anda tidak dapat menggunakan pernyataan berikut pada klaster sekunder saat penerusan tulis diaktifkan dalam sesi. Anda dapat menggunakan pernyataan ini pada klaster sekunder yang tidak memiliki penerusan tulis yang diaktifkan, atau dalam sesi dengan pengaturan `aurora_replica_read_consistency` yang kosong. Sebelum mengaktifkan penerusan tulis dalam sebuah sesi, periksa apakah kode Anda menggunakan pernyataan ini.

```
XA {START|BEGIN} xid [JOIN|RESUME]
XA END xid [SUSPEND [FOR MIGRATE]]
XA PREPARE xid
XA COMMIT xid [ONE PHASE]
XA ROLLBACK xid
XA RECOVER [CONVERT XID]
```

Pernyataan LOAD untuk tabel permanen

Anda tidak dapat menggunakan pernyataan berikut pada klaster sekunder dengan penerusan tulis yang diaktifkan.

```
LOAD DATA INFILE 'data.txt' INTO TABLE t1;
        LOAD XML LOCAL INFILE 'test.xml' INTO TABLE t1;
```

Anda dapat memuat data ke dalam tabel sementara pada klaster sekunder. Namun, pastikan Anda menjalankan semua pernyataan L0AD yang mengacu pada tabel permanen hanya ada pada klaster primer.

Pernyataan plugin

Anda tidak dapat menggunakan pernyataan berikut pada klaster sekunder dengan penerusan tulis yang diaktifkan.

```
INSTALL PLUGIN example SONAME 'ha_example.so';
UNINSTALL PLUGIN example;
```

Pernyataan SAVEPOINT

Anda tidak dapat menggunakan pernyataan berikut pada klaster sekunder saat penerusan tulis diaktifkan dalam sesi. Anda dapat menggunakan pernyataan ini pada klaster sekunder yang tidak memiliki penerusan tulis yang diaktifkan, atau dalam sesi dengan pengaturan

`aurora_replica_read_consistency` yang kosong. Periksa apakah kode Anda menggunakan pernyataan ini sebelum mengaktifkan penerusan tulis dalam sebuah sesi.

```
SAVEPOINT t1_save;
ROLLBACK TO SAVEPOINT t1_save;
RELEASE SAVEPOINT t1_save;
```

Isolasi dan konsistensi untuk penerusan tulis di Aurora MySQL

Dalam sesi yang menggunakan penerusan tulis, Anda hanya dapat menggunakan tingkat isolasi `REPEATABLE READ`. Meskipun Anda juga dapat menggunakan tingkat isolasi `READ COMMITTED` dengan klaster hanya-baca di Wilayah AWS sekunder, tingkat isolasi tersebut tidak berfungsi dengan penerusan tulis. Untuk informasi tentang tingkat isolasi `REPEATABLE READ` dan `READ COMMITTED`, lihat Tingkat isolasi Aurora MySQL.

Anda dapat mengontrol tingkat konsistensi baca di klaster sekunder. Tingkat konsistensi baca menentukan seberapa lama waktu tunggu klaster sekunder sebelum setiap operasi baca untuk memastikan bahwa beberapa atau semua perubahan direplikasi dari klaster primer. Anda dapat menyesuaikan tingkat konsistensi baca untuk memastikan bahwa semua operasi tulis yang diteruskan dari sesi Anda terlihat dalam klaster sekunder sebelum kueri berikutnya. Anda juga dapat menggunakan pengaturan ini untuk memastikan bahwa kueri pada klaster sekunder selalu melihat pembaruan terkini dari klaster primer. Hal ini bahkan berlaku untuk yang dikirimkan oleh sesi lain atau klaster lainnya. Untuk menentukan jenis perilaku ini untuk aplikasi Anda, Anda memilih nilai untuk parameter tingkat sesi `aurora_replica_read_consistency`.

> **Important**
>
> Selalu atur parameter `aurora_replica_read_consistency` untuk sesi apa pun yang ingin Anda terapkan penerusan tulis. Jika tidak, Aurora tidak mengaktifkan penerusan tulis untuk sesi tersebut. Parameter ini mempunyai nilai kosong secara default, jadi pilih nilai tertentu apabila Anda menggunakan parameter ini. Parameter `aurora_replica_read_consistency` memiliki efek hanya pada klaster sekunder dengan penerusan tulis diaktifkan.
>
> Untuk Aurora MySQL versi 2 dan versi 3 sebelum 3.04, gunakan `aurora_replica_read_consistency` sebagai variabel sesi. Untuk Aurora MySQL versi 3.04 dan lebih tinggi, Anda dapat menggunakan `aurora_replica_read_consistency` sebagai variabel sesi atau sebagai parameter klaster DB.

Untuk parameter `aurora_replica_read_consistency`, Anda dapat menentukan nilai `EVENTUAL`, `SESSION`, dan `GLOBAL`.

Saat Anda meningkatkan tingkat konsistensi, aplikasi Anda menghabiskan lebih banyak waktu untuk menunggu perubahan disebarkan di antara Wilayah AWS. Anda dapat memilih keseimbangan antara waktu respons yang cepat dan memastikan bahwa perubahan yang dilakukan di dalam lokasi lainnya sepenuhnya tersedia sebelum kueri Anda berjalan.

Dengan konsistensi baca diatur ke `EVENTUAL`, kueri di Wilayah AWS sekunder yang menggunakan penerusan tulis dapat melihat data yang sedikit usang karena lag replikasi. Hasil operasi tulis dalam sesi yang sama tidak terlihat sampai operasi tulis dilakukan pada Wilayah primer dan direplikasi ke Wilayah saat ini. Kueri tidak menunggu hasil yang diperbarui untuk tersedia. Dengan demikian, kueri dapat mengambil data yang lebih lama atau data yang diperbarui, bergantung pada waktu pernyataan dan jumlah lag replikasi.

Dengan konsistensi baca diatur ke `SESSION`, semua kueri di Wilayah AWS sekunder yang menggunakan penerusan tulis melihat hasil semua perubahan yang dilakukan dalam sesi tersebut. Perubahan dapat dilihat terlepas dari apakah transaksi dilakukan. Jika perlu, kueri menunggu hasil operasi tulis yang diteruskan untuk direplikasi ke Wilayah saat ini. Kueri tidak menunggu hasil yang diperbarui dari operasi tulis yang dilakukan di Wilayah lain atau dalam sesi lain di dalam Wilayah saat ini.

Dengan konsistensi baca diatur ke `GLOBAL`, sesi di Wilayah AWS sekunder melihat perubahan yang dibuat oleh sesi tersebut. Sesi tersebut juga melihat semua perubahan dari Wilayah AWS primer dan Wilayah AWS sekunder lainnya. Setiap kueri mungkin akan menunggu selama periode yang berbeda-beda tergantung jumlah lag sesi. Kueri berlanjut saat klaster sekunder dimutakhirkan dengan semua data yang di-commit dari klaster primer, pada saat kueri dimulai.

Untuk informasi selengkapnya tentang semua parameter yang terlibat dalam penerusan tulis, lihat Parameter konfigurasi untuk penerusan tulis di Aurora MySQL.

Contoh menggunakan penerusan tulis

Contoh-contoh ini menggunakan `aurora_replica_read_consistency` sebagai variabel sesi. Untuk Aurora MySQL versi 3.04 dan lebih tinggi, Anda dapat menggunakan `aurora_replica_read_consistency` sebagai variabel sesi atau sebagai parameter klaster DB.

Dalam contoh ini, klaster primer berada di Wilayah AS Timur (Virginia Utara). Klaster sekunder berada di Wilayah AS Timur (Ohio). Contoh ini menunjukkan efek menjalankan

pernyataan INSERT yang diikuti dengan pernyataan SELECT. Tergantung nilai pengaturan aurora_replica_read_consistency, hasilnya mungkin berbeda tergantung waktu pernyataan. Untuk mencapai konsistensi yang lebih tinggi, Anda mungkin menunggu sebentar sebelum mengeluarkan pernyataan SELECT. Atau Aurora dapat secara otomatis menunggu sampai hasil selesai mereplikasi sebelum melanjutkan dengan SELECT.

Dalam contoh ini, ada pengaturan konsistensi baca eventual. Menjalankan pernyataan INSERT yang langsung diikuti oleh pernyataan SELECT tetap mengembalikan nilai COUNT(*). Nilai ini mencerminkan jumlah baris sebelum baris baru disisipkan. Dengan menjalankan SELECT lagi beberapa saat kemudian, muncul hitungan baris yang diperbarui. Pernyataan SELECT tidak menunggu.

```
mysql> set aurora_replica_read_consistency = 'eventual';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
mysql> insert into t1 values (6); select count(*) from t1;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.00 sec)
```

Dengan pengaturan konsistensi baca session, pernyataan SELECT langsung setelah INSERT akan menunggu sampai perubahan dari pernyataan INSERT terlihat. Pernyataan SELECT selanjutnya tidak menunggu.

```
mysql> set aurora_replica_read_consistency = 'session';
mysql> select count(*) from t1;
```

```
+----------+
| count(*) |
+----------+
|        6 |
+----------+
1 row in set (0.01 sec)
mysql> insert into t1 values (6); select count(*) from t1; select count(*) from t1;
Query OK, 1 row affected (0.08 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.37 sec)
+----------+
| count(*) |
+----------+
|        7 |
+----------+
1 row in set (0.00 sec)
```

Dengan pengaturan konsistensi baca tetap disetel ke `session`, memperkenalkan waktu tunggu singkat setelah melaksanakan pernyataan `INSERT` membuat hitungan baris yang diperbarui tersedia pada saat pernyataan `SELECT` berikutnya berjalan.

```
mysql> insert into t1 values (6); select sleep(2); select count(*) from t1;
Query OK, 1 row affected (0.07 sec)
+----------+
| sleep(2) |
+----------+
|        0 |
+----------+
1 row in set (2.01 sec)
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.00 sec)
```

Dengan pengaturan konsistensi baca `global`, setiap pernyataan `SELECT` menunggu untuk memastikan bahwa semua data berubah sejak waktu mulai laporan terlihat sebelum melakukan kueri.

Durasi tunggu untuk setiap pernyataan `SELECT` berbeda-beda, tergantung jumlah lag replikasi antara klaster primer dan sekunder.

```
mysql> set aurora_replica_read_consistency = 'global';
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.75 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.37 sec)
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
1 row in set (0.66 sec)
```

Menjalankan pernyataan multibagian dengan penerusan tulis di Aurora MySQL

Pernyataan DML mungkin terdiri atas beberapa bagian, seperti pernyataan `INSERT ... SELECT` atau pernyataan `DELETE ... WHERE`. Dalam kasus ini, seluruh pernyataan diteruskan ke klaster primer dan berjalan di sana.

Transaksi dengan penerusan tulis di Aurora MySQL

Apakah transaksi diteruskan ke klaster primer ditentukan oleh mode akses transaksi. Anda dapat menentukan mode akses untuk transaksi dengan menggunakan pernyataan `SET TRANSACTION` atau pernyataan `START TRANSACTION`. Anda juga dapat menentukan mode akses transaksi dengan mengubah nilai variabel sesi Aurora MySQL `tx_read_only`. Anda hanya dapat mengubah nilai sesi ini saat Anda terhubung dengan klaster sekunder yang memiliki penerusan tulis diaktifkan.

Jika transaksi yang berlangsung lama tidak mengeluarkan pernyataan apa pun dalam waktu yang lama, transaksi tersebut kemungkinan melebihi periode batas waktu idle. Periode ini memiliki nilai

default satu menit. Anda dapat meningkatkannya hingga satu hari. Transaksi yang melebihi batas waktu idle dibatalkan oleh klaster primer. Pernyataan berikutnya yang Anda kirimkan akan menerima kesalahan waktu habis. Kemudian Aurora akan mengembalikan transaksi.

Jenis kesalahan ini dapat terjadi dalam kasus lain ketika penerusan tulis menjadi tidak tersedia. Misalnya, Aurora membatalkan transaksi apa pun yang menggunakan penerusan tulis jika Anda memulai ulang klaster primer atau jika Anda menonaktifkan pengaturan konfigurasi penerusan tulis.

Parameter konfigurasi untuk penerusan tulis di Aurora MySQL

Grup parameter klaster Aurora mencakup pengaturan untuk fitur penerusan tulis. Karena ini adalah parameter klaster, semua instans DB di setiap klaster memiliki nilai yang sama untuk variabel ini. Detail tentang parameter ini dirangkum dalam tabel berikut, dengan catatan penggunaan setelah tabel.

Nama	Cakupan	Tipe	Nilai default	Nilai valid
`aurora_fwd_master_idle_timeout` (Aurora MySQL versi 2)	Global	integer tanpa tanda	60	1-86.400
`aurora_fwd_master_max_connections_pct` (Aurora MySQL versi 2)	Global	integer panjang tanpa tanda	10	0–90
`aurora_fwd_writer_idle_timeout` (Aurora MySQL versi 3)	Global	integer tanpa tanda	60	1-86.400
`aurora_fwd_writer_max_connections_pct` (Aurora MySQL versi 3)	Global	integer panjang tanpa tanda	10	0–90
`aurora_replica_read_consistency`	Sesi	Enum	" (null)	`EVENTUAL`, `SESSION`, `GLOBAL`

Untuk mengontrol permintaan tulis yang masuk dari klaster sekunder, gunakan pengaturan ini pada klaster primer:

- `aurora_fwd_master_idle_timeout`, `aurora_fwd_writer_idle_timeout`: Jumlah detik yang diperlukan klaster primer untuk menunggu aktivitas pada koneksi yang diteruskan dari klaster sekunder sebelum menutupnya. Jika sesi tetap idle setelah periode ini, Aurora akan membatalkan sesi.
- `aurora_fwd_master_max_connections_pct`, `aurora_fwd_writer_max_connections_pct`: Batas atas pada koneksi basis data yang dapat digunakan pada instans DB penulis untuk menangani kueri yang diteruskan dari pembaca. Opsi ini dinyatakan dalam bentuk persentase pengaturan `max_connections` untuk instans DB dalam klaster primer. Misalnya, jika `max_connections` adalah 800 dan `aurora_fwd_master_max_connections_pct` atau `aurora_fwd_writer_max_connections_pct` adalah 10, maka penulis mengizinkan maksimal 80 sesi terusan serentak. Koneksi ini berasal dari pool koneksi yang sama yang dikelola oleh pengaturan `max_connections`.

  Pengaturan ini hanya berlaku pada klaster primer, saat satu atau beberapa klaster sekunder memiliki penerusan tulis yang diaktifkan. Jika Anda menurunkan nilai, koneksi yang ada tidak akan terpengaruh. Aurora mempertimbangkan nilai baru pengaturan saat mencoba membuat koneksi baru dari klaster sekunder. Nilai defaultnya adalah 10, mewakili 10% dari nilai `max_connections`. Jika Anda mengaktifkan penerusan kueri di klaster sekunder mana pun, pengaturan ini harus memiliki nilai bukan nol agar operasi tulis dari klaster sekunder dapat berhasil. Jika nilainya nol, operasi tulis menerima kode kesalahan `ER_CON_COUNT_ERROR` dengan pesan `Not enough connections on writer to handle your request`.

Parameter `aurora_replica_read_consistency` adalah parameter tingkat sesi yang memungkinkan penerusan tulis. Anda menggunakannya dalam setiap sesi. Anda dapat menentukan `EVENTUAL`, `SESSION`, atau `GLOBAL` untuk tingkat konsistensi baca. Untuk mempelajari selengkapnya tentang tingkat konsistensi, lihat Isolasi dan konsistensi untuk penerusan tulis di Aurora MySQL. Aturan berikut berlaku untuk parameter ini:

- Ini adalah parameter tingkat sesi. Nilai default adalah " (kosong).
- Penerusan tulis tersedia dalam sebuah sesi hanya jika `aurora_replica_read_consistency` diatur ke `EVENTUAL` atau `SESSION` atau `GLOBAL`. Parameter ini hanya relevan dalam instans pembaca klaster sekunder yang memiliki penerusan tulis diaktifkan dan yang berada di dalam basis data global Aurora.

- Anda tidak dapat mengatur variabel ini (saat kosong) atau membatalkan pengaturan (saat sudah diatur) di dalam transaksi multipernyataan. Namun, Anda dapat mengubahnya dari satu nilai yang valid (`EVENTUAL`, `SESSION`, atau `GLOBAL`) ke nilai valid lainnya (`EVENTUAL`, `SESSION`, atau `GLOBAL`) selama transaksi tersebut.
- Variabel tidak boleh `SET` ketika penerusan tulis tidak diaktifkan pada klaster sekunder.
- Mengatur variabel sesi pada klaster primer tidak memiliki efek apa pun. Jika Anda mencoba mengubah variabel ini pada klaster primer, Anda akan menerima kesalahan.

Metrik Amazon CloudWatch untuk penerusan tulis di Aurora MySQL

Metrik Amazon CloudWatch dan variabel status Aurora MySQL berikut ini berlaku saat Anda menggunakan penerusan tulis pada satu atau beberapa klaster sekunder. Semua metrik ini diukur pada instans DB penulis dalam klaster primer.

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
`ForwardingMasterDMLLatency`	–	Milidetik	Rata-rata waktu untuk memproses setiap pernyataan DML yang diteruskan pada instans DB penulis. Ini tidak termasuk waktu yang diperlukan klaster sekunder untuk meneruskan permintaan tulis, atau waktu untuk mereplikasi perubahan kembali ke klaster sekunder. Untuk Aurora MySQL versi 2.

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
`ForwardingMasterDMLThroughput`	–	Hitungan per detik	Jumlah pernyataan DML yang diteruskan yang diproses setiap detik oleh instans DB penulis ini. Untuk Aurora MySQL versi 2.
`ForwardingMasterOpenSessions`	`Aurora_fwd_master_open_sessions`	Hitungan	Jumlah sesi yang diteruskan pada instans DB tulis. Untuk Aurora MySQL versi 2.
–	`Aurora_fwd_master_dml_stmt_count`	Hitungan	Total jumlah pernyataan DML yang diteruskan ke instans DB penulis ini. Untuk Aurora MySQL versi 2.
–	`Aurora_fwd_master_dml_stmt_duration`	Mikrodetik	Total durasi pernyataan DML yang diteruskan ke instans DB penulis ini. Untuk Aurora MySQL versi 2.

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
–	`Aurora_fwd_master_select_stmt_count`	Hitungan	Total jumlah pernyataan `SELECT` yang diteruskan ke instans DB penulis ini. Untuk Aurora MySQL versi 2.
–	`Aurora_fwd_master_select_stmt_duration`	Mikrodetik	Total durasi pernyataan `SELECT` yang diteruskan ke instans DB penulis ini. Untuk Aurora MySQL versi 2.
`ForwardingWriterDMLLatency`	–	Milidetik	Rata-rata waktu untuk memproses setiap pernyataan DML yang diteruskan pada instans DB penulis. Ini tidak termasuk waktu yang diperlukan klaster sekunder untuk meneruskan permintaan tulis, atau waktu untuk mereplikasi perubahan kembali ke klaster sekunder. Untuk Aurora MySQL versi 3.

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
ForwardingWriterDMLThroughput	–	Hitungan per detik	Jumlah pernyataan DML yang diteruskan yang diproses setiap detik oleh instans DB penulis ini. Untuk Aurora MySQL versi 3.
ForwardingWriterOpenSessions	Aurora_fwd_writer_open_sessions	Hitungan	Jumlah sesi yang diteruskan pada instans DB tulis. Untuk Aurora MySQL versi 3.
–	Aurora_fwd_writer_dml_stmt_count	Hitungan	Total jumlah pernyataan DML yang diteruskan ke instans DB penulis ini. Untuk Aurora MySQL versi 3.
–	Aurora_fwd_writer_dml_stmt_duration	Mikrodetik	Total durasi pernyataan DML yang diteruskan ke instans DB penulis ini.
–	Aurora_fwd_writer_select_stmt_count	Hitungan	Total jumlah pernyataan SELECT yang diteruskan ke instans DB penulis ini. Untuk Aurora MySQL versi 3.

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
–	`Aurora_fwd_writer_select_stmt_duration`	Mikrodetik	Total durasi pernyataan `SELECT` yang diteruskan ke instans DB penulis ini. Untuk Aurora MySQL versi 3.

Metrik CloudWatch dan variabel status Aurora MySQL berikut berlaku untuk setiap klaster sekunder. Metrik-metrik ini diukur pada setiap instans DB pembaca dalam klaster sekunder dengan penerusan tulis yang diaktfikan.

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
`ForwardingReplicaDMLLatency`	–	Milidetik	Rata-rata waktu respons DML yang diteruskan pada replika.
`ForwardingReplicaDMLThroughput`	–	Hitungan per detik	Jumlah pernyataan DML yang diteruskan yang diproses setiap detiknya.
`ForwardingReplicaOpenSessions`	`Aurora_fwd_replica_open_sessions`	Hitungan	Jumlah sesi yang menggunakan penerusan tulis pada instans DB pembaca.
`ForwardingReplicaReadWaitLatency`	–	Milidetik	Rata-rata waktu tunggu yang diperlukan sebuah pernyataan `SELECT` di instans

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
			DB pembaca untuk menunggu agar dapat mengimbangi klaster primer. Batas tunggu instans DB pembaca sebelum memproses kueri ditentukan oleh pengaturan `aurora_replica_read_consistency` .
`ForwardingReplicaReadWaitThroughput`	–	Hitungan per detik	Jumlah total pernyataan `SELECT` yang diproses setiap detik di semua sesi yang meneruskan tulis.
`ForwardingReplicaSelectLatency`	(–)	Milidetik	Latensi `SELECT` yang diteruskan, rata-rata atas semua pernyataan `SELECT` yang diteruskan dalam periode pemantauan.
`ForwardingReplicaSelectThroughput`	–	Hitungan per detik	Rata-rata throughput `SELECT` per detik yang diteruskan dalam periode pemantauan.

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
–	`Aurora_fwd_replica_dml_stmt_count`	Hitungan	Total jumlah pernyataan DML yang diteruskan ke instans DB pembaca ini.
–	`Aurora_fwd_replica_dml_stmt_duration`	Mikrodetik	Total durasi semua pernyataan DML yang diteruskan ke instans DB pembaca ini.
–	`Aurora_fwd_replica_errors_session_limit`	Hitungan	Jumlah sesi yang ditolak oleh klaster primer karena salah satu kondisi kesalahan berikut: • penulis penuh • Terlalu banyak pernyataan yang diteruskan dalam proses.
–	`Aurora_fwd_replica_read_wait_count`	Hitungan	Total jumlah tunggu baca-sesudah-tulis pada instans DB pembaca ini.
–	`Aurora_fwd_replica_read_wait_duration`	Mikrodetik	Total durasi tunggu karena pengaturan konsistensi baca pada instans DB pembaca ini.

Metrik CloudWatch	Variabel status Aurora MySQL	Unit	Deskripsi
–	`Aurora_fwd_replica_select_stmt_count`	Hitungan	Total jumlah pernyataan `SELECT` yang diteruskan ke instans DB pembaca ini.
–	`Aurora_fwd_replica_select_stmt_duration`	Mikrodetik	Total durasi pernyataan `SELECT` yang diteruskan ke instans DB pembaca ini.

Menggunakan penerusan tulis di dalam basis data global Aurora PostgreSQL

Topik

- Ketersediaan wilayah dan versi penerusan tulis di Aurora PostgreSQL
- Mengaktifkan penerusan tulis di Aurora PostgreSQL
- Memeriksa apakah klaster sekunder telah mengaktifkan penerusan tulis di Aurora PostgreSQL
- Kompatibilitas aplikasi dan SQL dengan penerusan tulis di Aurora PostgreSQL
- Isolasi dan konsistensi untuk penerusan tulis di Aurora PostgreSQL
- Menjalankan pernyataan multibagian dengan penerusan tulis di Aurora PostgreSQL
- Parameter konfigurasi untuk penerusan tulis di Aurora PostgreSQL
- CloudWatch Metrik Amazon untuk penerusan tulis di Aurora PostgreSQL
- Peristiwa tunggu untuk penerusan tulis di Aurora PostgreSQL

Ketersediaan wilayah dan versi penerusan tulis di Aurora PostgreSQL

Penerusan tulis didukung dengan Aurora PostgreSQL versi 15.4 dan versi kecil yang lebih tinggi, serta versi 14.9 dan versi kecil yang lebih tinggi. Penerusan tulis tersedia di setiap Wilayah tempat basis data global berbasis Aurora PostgreSQL tersedia.

Untuk informasi lebih lanjut tentang versi dan ketersediaan Wilayah basis data global Aurora PostgreSQL, lihat Basis data global Aurora dengan Aurora PostgreSQL.

Mengaktifkan penerusan tulis di Aurora PostgreSQL

Secara default, penerusan tulis tidak diaktifkan saat Anda menambahkan klaster sekunder ke basis data global Aurora. Anda dapat mengaktifkan penerusan tulis untuk klaster DB sekunder Anda saat Anda membuatnya atau kapan saja setelah Anda membuatnya. Jika perlu, Anda dapat menonaktifkannya nanti. Mengaktifkan atau menonaktifkan penerusan tulis tidak menyebabkan waktu henti atau boot ulang.

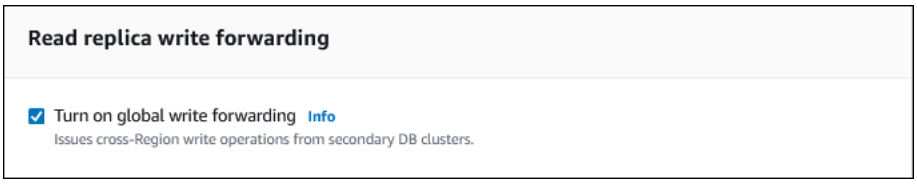
Konsol

Di konsol, Anda dapat mengaktifkan atau menonaktifkan penerusan tulis saat Anda membuat atau memodifikasi klaster DB sekunder.

Mengaktifkan atau menonaktifkan penerusan tulis saat membuat klaster DB sekunder

Saat Anda membuat klaster DB sekunder baru, Anda mengaktifkan penerusan tulis dengan memilih kotak centang Aktifkan penerusan tulis global di bagian Penerusan tulis replika baca. Atau kosongkan kotak centang untuk menonaktifkannya. Untuk membuat klaster DB sekunder, ikuti petunjuk untuk mesin DB Anda di Membuat klaster DB Amazon Aurora.

Tangkapan layar berikut menunjukkan bagian Penerusan tulis replika baca dengan kotak centang Aktifkan penerusan tulis global dipilih.



Mengaktifkan atau menonaktifkan penerusan tulis saat memodifikasi klaster DB sekunder

Di konsol, Anda dapat memodifikasi klaster DB sekunder untuk mengaktifkan atau menonaktifkan penerusan tulis.

Mengaktifkan atau menonaktifkan penerusan tulis untuk klaster DB sekunder dengan menggunakan konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data.

3. Pilih klaster DB sekunder, dan pilih Modifikasi.
4. Di bagian Penerusan tulis replika baca, pilih atau kosongkan kotak centang Aktifkan penerusan tulis global.
5. Pilih Lanjutkan.
6. Untuk Jadwalkan modifikasi, pilih Terapkan segera. Jika Anda memilih Terapkan selama periode pemeliharaan terjadwal berikutnya, Aurora akan mengabaikan pengaturan ini dan langsung mengaktifkan penerusan tulis.
7. Pilih Modifikasi klaster.

AWS CLI

Untuk mengaktifkan penerusan tulis dengan menggunakan AWS CLI, gunakan opsi. `--enable-global-write-forwarding` Opsi ini berfungsi saat Anda membuat cluster sekunder baru menggunakan create-db-clusterperintah. Ini juga berfungsi ketika Anda memodifikasi cluster sekunder yang ada menggunakan modify-db-clusterperintah. Opsi ini mengharuskan basis data global menggunakan versi Aurora yang mendukung penerusan tulis. Anda dapat menonaktifkan penerusan tulis dengan menggunakan opsi `--no-enable-global-write-forwarding` dengan perintah CLI yang sama ini.

Prosedur berikut ini menjelaskan cara mengaktifkan atau menonaktifkan penerusan tulis untuk klaster DB sekunder di klaster global Anda dengan menggunakan AWS CLI.

Mengaktifkan atau menonaktifkan penerusan tulis untuk klaster DB sekunder yang ada

- Panggil modify-db-cluster AWS CLI perintah dan berikan nilai-nilai berikut:
  - `--db-cluster-identifier` – Nama klaster DB.
  - `--enable-global-write-forwarding` untuk mengaktifkan atau `--no-enable-global-write-forwarding` untuk menonaktifkan.

  Contoh berikut mengaktifkan penerusan tulis untuk klaster DB `sample-secondary-db-cluster`.

  Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-secondary-db-cluster \
```

```
    --enable-global-write-forwarding
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-secondary-db-cluster ^
    --enable-global-write-forwarding
```

API RDS

Untuk mengaktifkan penerusan tulis menggunakan API Amazon RDS, atur parameter `EnableGlobalWriteForwarding` ke `true`. Parameter ini berfungsi saat Anda membuat klaster sekunder baru menggunakan operasi CreateDBCluster. Opsi ini juga berfungsi saat Anda memodifikasi klaster sekunder yang ada dengan menggunakan operasi ModifyDBCluster. Opsi ini mengharuskan basis data global menggunakan versi Aurora yang mendukung penerusan tulis. Anda dapat menonaktifkan penerusan tulis dengan mengatur parameter `EnableGlobalWriteForwarding` ke `false`.

Memeriksa apakah klaster sekunder telah mengaktifkan penerusan tulis di Aurora PostgreSQL

Untuk menentukan apakah Anda dapat menggunakan penerusan tulis dari klaster sekunder, Anda dapat memeriksa apakah klaster tersebut memiliki atribut `"GlobalWriteForwardingStatus": "enabled"`.

Di AWS Management Console, Anda lihat `Read replica write forwarding` pada tab Konfigurasi halaman detail untuk cluster. Untuk melihat status setelan penerusan tulis global untuk semua cluster Anda, jalankan perintah berikut. AWS CLI

Klaster sekunder menunjukkan nilai `"enabled"` atau `"disabled"` untuk menunjukkan apakah penerusan tulis diaktifkan atau dinonaktifkan. Nilai `null` menunjukkan bahwa penerusan tulis tidak tersedia untuk klaster tersebut. Klaster ini bukan bagian dari basis data global, atau merupakan klaster primer, bukan klaster sekunder. Nilainya juga bisa `"enabling"` atau `"disabling"` jika penerusan tulis sedang dalam proses diaktifkan atau dinonaktifkan.

Example

```
aws rds describe-db-clusters --query '*[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
```

```
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    },
    {
        "GlobalWriteForwardingStatus": "disabled",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-2"
    },
    {
        "GlobalWriteForwardingStatus": null,
        "DBClusterIdentifier": "non-global-cluster"
    }
]
```

Untuk menemukan semua klaster sekunder yang memiliki penerusan tulis global, jalankan perintah berikut. Perintah ini juga mengembalikan titik akhir pembaca klaster ini. Anda menggunakan titik akhir pembaca klaster sekunder ketika Anda menggunakan penerusan tulis dari sekunder ke primer dalam basis data global Aurora Anda.

Example

```
aws rds describe-db-clusters --query 'DBClusters[].
{DBClusterIdentifier:DBClusterIdentifier,GlobalWriteForwardingStatus:GlobalWriteForwardingStatu
 | [?GlobalWriteForwardingStatus == `enabled`]'
[
    {
        "GlobalWriteForwardingStatus": "enabled",
        "ReaderEndpoint": "aurora-write-forwarding-test-replica-1.cluster-ro-
cnpexample.us-west-2.rds.amazonaws.com",
        "DBClusterIdentifier": "aurora-write-forwarding-test-replica-1"
    }
]
```

Kompatibilitas aplikasi dan SQL dengan penerusan tulis di Aurora PostgreSQL

Pernyataan tertentu tidak diizinkan atau dapat menghasilkan hasil usang saat Anda menggunakannya dalam basis data global dengan penerusan tulis. Selain itu, fungsi yang ditentukan pengguna dan prosedur yang ditentukan pengguna tidak didukung. Dengan demikian, pengaturan `EnableGlobalWriteForwarding` dinonaktifkan secara default untuk klaster sekunder. Sebelum mengaktifkan pengaturan tersebut, periksa untuk memastikan bahwa kode aplikasi Anda tidak terpengaruh oleh pembatasan ini.

Anda dapat menggunakan jenis pernyataan SQL berikut dengan penerusan tulis:

- Pernyataan bahasa manipulasi data (DML), seperti `INSERT`, `DELETE`, dan `UPDATE`
- Pernyataan `SELECT FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE }`
- Pernyataan `PREPARE` dan `EXECUTE`
- Pernyataan `EXPLAIN` dengan pernyataan di dalam daftar ini

Jenis pernyataan SQL berikut ini tidak didukung dengan penerusan tulis:

- Pernyataan bahasa definisi data (DDL)
- `ANALYZE`
- `CLUSTER`
- `COPY`
- Kursor - Kursor tidak didukung, jadi pastikan untuk menutupnya sebelum menggunakan penerusan tulis.
- `GRANT|REVOKE|REASSIGN OWNED|SECURITY LABEL`
- `LOCK`
- Pernyataan `SAVEPOINT`
- `SELECT INTO`
- `SET CONSTRAINTS`
- `TRUNCATE`
- `VACUUM`

Isolasi dan konsistensi untuk penerusan tulis di Aurora PostgreSQL

Dalam sesi yang menggunakan penerusan tulis, Anda dapat menggunakan tingkat isolasi `REPEATABLE READ` dan `READ COMMITTED`. Namun, tingkat isolasi `SERIALIZABLE` tidak didukung.

Anda dapat mengontrol tingkat konsistensi baca di klaster sekunder. Tingkat konsistensi baca menentukan seberapa lama waktu tunggu klaster sekunder sebelum setiap operasi baca untuk memastikan bahwa beberapa atau semua perubahan direplikasi dari klaster primer. Anda dapat menyesuaikan tingkat konsistensi baca untuk memastikan bahwa semua operasi tulis yang diteruskan dari sesi Anda terlihat dalam klaster sekunder sebelum kueri berikutnya. Anda juga dapat menggunakan pengaturan ini untuk memastikan bahwa kueri pada klaster sekunder selalu melihat

pembaruan terkini dari klaster primer. Hal ini bahkan berlaku untuk yang dikirimkan oleh sesi lain atau klaster lainnya. Untuk menentukan jenis perilaku ini untuk aplikasi Anda, Anda memilih nilai yang sesuai untuk parameter tingkat sesi `apg_write_forward.consistency_mode`. Parameter `apg_write_forward.consistency_mode` memiliki efek hanya pada klaster sekunder dengan penerusan tulis diaktifkan.

> **Note**
>
> Untuk parameter `apg_write_forward.consistency_mode`, Anda dapat menentukan nilai `SESSION`, `EVENTUAL`, `GLOBAL`, atau `OFF`. Secara default, nilainya diatur ke `SESSION`. Mengatur nilai ini ke `OFF` akan menonaktifkan penerusan tulis dalam sesi.

Saat Anda meningkatkan tingkat konsistensi, aplikasi Anda menghabiskan lebih banyak waktu menunggu perubahan disebarkan antar AWS Wilayah. Anda dapat memilih keseimbangan antara waktu respons yang cepat dan memastikan bahwa perubahan yang dilakukan di dalam lokasi lainnya sepenuhnya tersedia sebelum kueri Anda berjalan.

Dengan setiap pengaturan mode konsistensi yang tersedia, efeknya adalah sebagai berikut:

- `SESSION`— Semua kueri di AWS Wilayah sekunder yang menggunakan penerusan tulis melihat hasil dari semua perubahan yang dibuat dalam sesi itu. Perubahan dapat dilihat terlepas dari apakah transaksi dilakukan. Jika perlu, kueri menunggu hasil operasi tulis yang diteruskan untuk direplikasi ke Wilayah saat ini. Kueri tidak menunggu hasil yang diperbarui dari operasi tulis yang dilakukan di Wilayah lain atau dalam sesi lain di dalam Wilayah saat ini.
- `EVENTUAL`— Kueri di AWS Wilayah sekunder yang menggunakan penerusan tulis mungkin melihat data yang sedikit basi karena kelambatan replikasi. Hasil operasi tulis dalam sesi yang sama tidak terlihat sampai operasi tulis dilakukan pada Wilayah primer dan direplikasi ke Wilayah saat ini. Kueri tidak menunggu hasil yang diperbarui untuk tersedia. Dengan demikian, kueri dapat mengambil data yang lebih lama atau data yang diperbarui, bergantung pada waktu pernyataan dan jumlah lag replikasi.
- `GLOBAL`— Sesi di AWS Wilayah sekunder melihat perubahan yang dibuat oleh sesi itu. Ini juga melihat semua perubahan yang dilakukan dari AWS Wilayah primer dan AWS Wilayah sekunder lainnya. Setiap kueri mungkin akan menunggu selama periode yang berbeda-beda tergantung jumlah lag sesi. Kueri berlangsung ketika cluster sekunder up-to-date dengan semua data yang dikomit dari cluster primer, pada saat kueri dimulai.
- `OFF` – Penerusan tulis dinonaktifkan dalam sesi.

Untuk informasi selengkapnya tentang semua parameter yang terlibat dalam penerusan tulis, lihat Parameter konfigurasi untuk penerusan tulis di Aurora PostgreSQL.

Menjalankan pernyataan multibagian dengan penerusan tulis di Aurora PostgreSQL

Pernyataan DML mungkin terdiri atas beberapa bagian, seperti pernyataan `INSERT ... SELECT` atau pernyataan `DELETE ... WHERE`. Dalam kasus ini, seluruh pernyataan diteruskan ke klaster primer dan berjalan di sana.

Parameter konfigurasi untuk penerusan tulis di Aurora PostgreSQL

Grup parameter klaster Aurora mencakup pengaturan untuk fitur penerusan tulis. Karena ini adalah parameter klaster, semua instans DB di setiap klaster memiliki nilai yang sama untuk variabel ini. Detail tentang parameter ini dirangkum dalam tabel berikut, dengan catatan penggunaan setelah tabel.

Nama	Cakupan	Tipe	Nilai default	Nilai valid
`apg_write_forward.connect_timeout`	Sesi	detik	30	0–2147483647
`apg_write_forward.consistency_mode`	Sesi	enum	Sesi	`SESSION`, `EVENTUAL`, `GLOBAL`, `OFF`
`apg_write_forward.idle_in_transaction_session_timeout`	Sesi	milidetik	86400000	0–2147483647
`apg_write_forward.idle_session_timeout`	Sesi	milidetik	300000	0–2147483647
`apg_write_forward.max_forwarding_connections_percent`	Global	int	25	1–100

Parameter `apg_write_forward.max_forwarding_connections_percent` adalah batas atas pada slot koneksi basis data yang dapat digunakan untuk menangani kueri yang diteruskan

dari pembaca. Opsi ini dinyatakan dalam bentuk persentase pengaturan `max_connections` untuk instans DB dalam klaster primer. Misalnya, jika `max_connections` adalah `800` dan `apg_write_forward.max_forwarding_connections_percent` adalah `10`, maka penulis mengizinkan maksimum 80 sesi yang diteruskan secara serentak. Koneksi ini berasal dari pool koneksi yang sama yang dikelola oleh pengaturan `max_connections`. Pengaturan ini hanya berlaku pada klaster primer saat setidaknya ada satu klaster sekunder yang memiliki penerusan tulis yang diaktifkan.

Gunakan pengaturan berikut pada klaster sekunder:

- `apg_write_forward.consistency_mode` – Parameter tingkat sesi yang mengontrol tingkat konsistensi baca pada klaster sekunder. Nilai yang valid adalah `SESSION`, `EVENTUAL`, `GLOBAL`, atau `OFF`. Secara default, nilainya diatur ke `SESSION`. Mengatur nilai ini ke `OFF` akan menonaktifkan penerusan tulis dalam sesi. Untuk mempelajari selengkapnya tentang tingkat konsistensi, lihat Isolasi dan konsistensi untuk penerusan tulis di Aurora PostgreSQL. Parameter ini hanya relevan dalam instans pembaca klaster sekunder yang memiliki penerusan tulis diaktifkan dan yang berada di dalam basis data global Aurora.
- `apg_write_forward.connect_timeout` – Jumlah maksimum detik yang diperlukan klaster sekunder untuk menunggu saat membuat koneksi ke klaster primer sebelum menyerah. Nilai `0` berarti menunggu tanpa batas.
- `apg_write_forward.idle_in_transaction_session_timeout` – Jumlah milidetik yang diperlukan klaster primer untuk menunggu aktivitas pada koneksi yang diteruskan dari klaster sekunder yang memiliki transaksi terbuka sebelum menutupnya. Jika sesi tetap idle dalam transaksi setelah periode ini, Aurora akan mengakhiri sesi. Nilai `0` menonaktifkan batas waktu.
- `apg_write_forward.idle_session_timeout` – Jumlah milidetik yang diperlukan klaster primer untuk menunggu aktivitas pada koneksi yang diteruskan dari klaster sekunder sebelum menutupnya. Jika sesi tetap idle setelah periode ini, Aurora akan mengakhiri sesi. Nilai `0` menonaktifkan batas waktu.

CloudWatch Metrik Amazon untuk penerusan tulis di Aurora PostgreSQL

CloudWatch Metrik Amazon berikut berlaku untuk klaster utama saat Anda menggunakan penerusan tulis pada satu atau beberapa klaster sekunder. Semua metrik ini diukur pada instans DB penulis dalam klaster primer.

CloudWatch Metrik	Unit dan deskripsi
AuroraForwardingWriterDMLThroughput	Hitungan (per detik). Jumlah pernyataan DML yang diteruskan yang diproses setiap detik oleh instans DB penulis ini.
AuroraForwardingWriterOpenSessions	Hitungan. Jumlah sesi terbuka pada instans DB penulis ini yang memproses kueri yang diteruskan.
AuroraForwardingWriterTotalSessions	Hitungan. Total jumlah sesi yang diteruskan pada instans DB penulis ini.

CloudWatch Metrik berikut berlaku untuk setiap cluster sekunder. Metrik-metrik ini diukur pada setiap instans DB pembaca dalam klaster sekunder dengan penerusan tulis yang diaktifkan.

CloudWatch Metrik	Unit dan deskripsi
AuroraForwardingReplicaCommitThroughput	Hitungan (per detik). Jumlah commit dalam sesi yang diteruskan oleh replika ini setiap detik.
AuroraForwardingReplicaDMLLatency	Milidetik. Rata-rata waktu respons dalam milidetik DML yang diteruskan pada replika.
AuroraForwardingReplicaDMLThroughput	Hitungan (per detik). Jumlah pernyataan DML yang diteruskan yang diproses pada replika ini setiap detiknya.
AuroraForwardingReplicaErrorSessionsLimit	Hitungan. Jumlah sesi yang ditolak oleh klaster primer karena batas untuk koneksi maksimum atau koneksi penerusan tulis maksimum tercapai.
AuroraForwardingReplicaOpenSessions	Hitungan. Jumlah sesi yang menggunakan penerusan tulis pada instans replika.
AuroraForwardingReplicaReadWaitLatency	Milidetik. Rata-rata waktu tunggu dalam milidetik yang diperlukan oleh replika untuk

CloudWatch Metrik	Unit dan deskripsi
	menunggu agar konsisten dengan LSN klaster primer. Batas tunggu instans DB pembaca ditentukan oleh pengaturan `apg_write_forward.consistency_mode`. Untuk informasi tentang pengaturan ini, lihat the section called “Parameter konfigurasi untuk penerusan tulis di Aurora PostgreSQL”.

Peristiwa tunggu untuk penerusan tulis di Aurora PostgreSQL

Amazon Aurora menghasilkan peristiwa tunggu berikut ini saat Anda menggunakan penerusan tulis dengan Aurora PostgreSQL.

Topik

- IPC: AuroraWriteForwardConnect
- IPC: AuroraWriteForwardConsistencyPoint
- IPC: AuroraWriteForwardExecute
- IPC: AuroraWriteForwardGetGlobalConsistencyPoint
- IPC: AuroraWriteForwardXactAbort
- IPC: AuroraWriteForwardXactCommit
- IPC: AuroraWriteForwardXactStart

IPC: AuroraWriteForwardConnect

Peristiwa `IPC:AuroraWriteForwardConnect` terjadi ketika proses backend pada klaster DB sekunder sedang menunggu koneksi ke simpul penulis klaster DB primer dibuka.

Kemungkinan menyebabkan peningkatan waktu tunggu

Peristiwa ini meningkat seiring meningkatnya jumlah upaya koneksi dari simpul pembaca Wilayah sekunder ke simpul penulis klaster DB primer.

Tindakan

Kurangi jumlah koneksi serentak dari simpul sekunder ke simpul penulis Wilayah primer.

IPC: AuroraWriteForwardConsistencyPoint

Peristiwa `IPC:AuroraWriteForwardConsistencyPoint` menjelaskan durasi kueri dari simpul di klaster DB sekunder akan menunggu hasil operasi tulis yang diteruskan untuk direplikasi ke Wilayah saat ini. Peristiwa ini hanya dihasilkan jika parameter tingkat sesi `apg_write_forward.consistency_mode` diatur ke salah satu dari berikut ini:

- `SESSION` – kueri pada simpul sekunder menunggu hasil semua perubahan yang dibuat dalam sesi tersebut.
- `GLOBAL` – kueri pada simpul sekunder menunggu hasil perubahan yang dibuat oleh sesi tersebut, ditambah semua perubahan yang di-commit dari Wilayah primer dan Wilayah sekunder lainnya di klaster global.

Untuk informasi selengkapnya tentang pengaturan parameter `apg_write_forward.consistency_mode`, lihat the section called “Parameter konfigurasi untuk penerusan tulis di Aurora PostgreSQL”.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum untuk waktu tunggu yang lebih lama termasuk yang berikut:

- Peningkatan lag replika, yang diukur dengan CloudWatch `ReplicaLag` metrik Amazon. Untuk informasi selengkapnya tentang metrik ini, lihat Memantau replikasi Aurora PostgreSQL.
- Peningkatan beban pada simpul penulis Wilayah primer atau pada simpul sekunder.

Tindakan

Ubah mode konsistensi Anda, tergantung kebutuhan aplikasi Anda.

IPC: AuroraWriteForwardExecute

Peristiwa `IPC:AuroraWriteForwardExecute` terjadi ketika proses backend pada klaster DB sekunder sedang menunggu kueri yang diteruskan selesai dan mendapatkan hasil dari simpul penulis klaster DB primer.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Mengambil banyak baris dari simpul penulis Wilayah primer.
- Peningkatan latensi jaringan antara simpul sekunder dan simpul penulis Wilayah primer meningkatkan waktu yang dibutuhkan simpul sekunder untuk menerima data dari simpul penulis.
- Peningkatan beban pada simpul sekunder dapat menunda pengiriman permintaan kueri dari simpul sekunder ke simpul penulis Wilayah primer.
- Peningkatan beban pada simpul penulis Wilayah primer dapat menunda pengiriman data dari simpul penulis ke simpul sekunder.

Tindakan

Kami merekomendasikan tindakan yang berbeda-beda tergantung penyebab peristiwa tunggu Anda.

- Optimalkan kueri untuk mengambil hanya data yang diperlukan.
- Optimalkan operasi bahasa manipulasi data (DML) untuk hanya memodifikasi data yang diperlukan.
- Jika simpul sekunder atau simpul penulis Wilayah primer dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instans dengan kapasitas CPU yang lebih besar atau lebih banyak bandwidth jaringan.

IPC: AuroraWriteForwardGetGlobalConsistencyPoint

Peristiwa `IPC:AuroraWriteForwardGetGlobalConsistencyPoint` terjadi ketika proses backend pada klaster DB sekunder yang menggunakan mode konsistensi GLOBAL sedang menunggu untuk mendapatkan titik konsistensi global dari simpul penulis sebelum menjalankan kueri.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Peningkatan latensi jaringan antara simpul sekunder dan simpul penulis Wilayah primer meningkatkan waktu yang dibutuhkan simpul sekunder untuk menerima data dari simpul penulis.
- Peningkatan beban pada simpul sekunder dapat menunda pengiriman permintaan kueri dari simpul sekunder ke simpul penulis Wilayah primer.
- Peningkatan beban pada simpul penulis Wilayah primer dapat menunda pengiriman data dari simpul penulis ke simpul sekunder.

Tindakan

Kami merekomendasikan tindakan yang berbeda-beda tergantung penyebab peristiwa tunggu Anda.

- Ubah mode konsistensi Anda, tergantung kebutuhan aplikasi Anda.
- Jika simpul sekunder atau simpul penulis Wilayah primer dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instans dengan kapasitas CPU yang lebih besar atau lebih banyak bandwidth jaringan.

IPC: AuroraWriteForwardXactAbort

Peristiwa `IPC:AuroraWriteForwardXactAbort` terjadi ketika proses backend pada klaster DB sekunder sedang menunggu hasil kueri pembersihan jarak jauh. Kueri pembersihan dikeluarkan untuk mengembalikan proses ke keadaan yang sesuai setelah transaksi dengan penerusan tulis dibatalkan. Amazon Aurora melakukannya karena ada kesalahan yang ditemukan atau karena pengguna mengeluarkan perintah `ABORT` eksplisit atau membatalkan kueri yang sedang berjalan.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Peningkatan latensi jaringan antara simpul sekunder dan simpul penulis Wilayah primer meningkatkan waktu yang dibutuhkan simpul sekunder untuk menerima data dari simpul penulis.
- Peningkatan beban pada simpul sekunder dapat menunda pengiriman permintaan kueri pembersihan dari simpul sekunder ke simpul penulis Wilayah primer.
- Peningkatan beban pada simpul penulis Wilayah primer dapat menunda pengiriman data dari simpul penulis ke simpul sekunder.

Tindakan

Kami merekomendasikan tindakan yang berbeda-beda tergantung penyebab peristiwa tunggu Anda.

- Selidiki penyebab transaksi yang dibatalkan.
- Jika simpul sekunder atau simpul penulis Wilayah primer dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instans dengan kapasitas CPU yang lebih besar atau lebih banyak bandwidth jaringan.

IPC: AuroraWriteForwardXactCommit

Peristiwa `IPC:AuroraWriteForwardXactCommit` terjadi ketika proses backend pada klaster DB sekunder sedang menunggu hasil perintah transaksi commit yang diteruskan.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Peningkatan latensi jaringan antara simpul sekunder dan simpul penulis Wilayah primer meningkatkan waktu yang dibutuhkan simpul sekunder untuk menerima data dari simpul penulis.
- Peningkatan beban pada simpul sekunder dapat menunda pengiriman permintaan kueri dari simpul sekunder ke simpul penulis Wilayah primer.
- Peningkatan beban pada simpul penulis Wilayah primer dapat menunda pengiriman data dari simpul penulis ke simpul sekunder.

Tindakan

Jika simpul sekunder atau simpul penulis Wilayah primer dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instans dengan kapasitas CPU yang lebih besar atau lebih banyak bandwidth jaringan.

IPC: AuroraWriteForwardXactStart

Peristiwa `IPC:AuroraWriteForwardXactStart` terjadi ketika proses backend pada klaster DB sekunder sedang menunggu hasil perintah transaksi mulai yang diteruskan.

Kemungkinan menyebabkan peningkatan waktu tunggu

Penyebab umum peningkatan waktu tunggu antara lain:

- Peningkatan latensi jaringan antara simpul sekunder dan simpul penulis Wilayah primer meningkatkan waktu yang dibutuhkan simpul sekunder untuk menerima data dari simpul penulis.
- Peningkatan beban pada simpul sekunder dapat menunda pengiriman permintaan kueri dari simpul sekunder ke simpul penulis Wilayah primer.
- Peningkatan beban pada simpul penulis Wilayah primer dapat menunda pengiriman data dari simpul penulis ke simpul sekunder.

Tindakan

Jika simpul sekunder atau simpul penulis Wilayah primer dibatasi oleh CPU atau bandwidth jaringan, pertimbangkan untuk mengubahnya menjadi tipe instans dengan kapasitas CPU yang lebih besar atau lebih banyak bandwidth jaringan.

Menggunakan switchover atau failover dalam basis data global Amazon Aurora

Basis data global Aurora menghadirkan perlindungan kelangsungan bisnis dan pemulihan bencana (BCDR) yang lebih tinggi daripada ketersediaan tinggi standar yang dihadirkan oleh klaster DB Aurora dalam satu Wilayah AWS. Dengan menggunakan basis data global Aurora, Anda dapat membuat rencana dan melakukan pemulihan dari bencana Regional yang riil atau menyelesaikan pemadaman tingkat layanan dengan cepat. Pemulihan dari bencana biasanya didorong oleh dua tujuan bisnis berikut:

- Sasaran waktu pemulihan (RTO)—Waktu yang diperlukan sistem untuk kembali ke status kerja aktif setelah terjadinya bencana atau pemadaman layanan. Dengan kata lain, RTO mengukur waktu henti operasional. Untuk basis data global Aurora, RTO dapat berada dalam hitungan menit.
- Sasaran titik pemulihan (RPO)—Jumlah data yang dapat hilang (diukur berdasarkan waktu) setelah terjadinya bencana atau pemadaman layanan. Kehilangan data tersebut biasanya disebabkan oleh keterlambatan replikasi asinkron. Untuk basis data global Aurora, RPO biasanya diukur dalam hitungan detik. Dengan basis data global berbasis Aurora PostgreSQL, Anda dapat menggunakan parameter `rds.global_db_rpo` untuk mengatur dan melacak batas atas RPO, tetapi melakukannya dapat memengaruhi pemrosesan transaksi pada simpul penulis di klaster primer. Untuk informasi selengkapnya, lihat Mengelola RPO untuk basis data global berbasis Aurora PostgreSQL.

Melakukan switchover atau failover terhadap basis data global Aurora melibatkan tindakan promosi klaster DB di salah satu Wilayah sekunder basis data global Anda menjadi klaster DB primer. Istilah "pemadaman regional" sering kali digunakan untuk mendeskripsikan berbagai skenario kegagalan. Skenario terburuk dapat berupa pemadaman luas karena peristiwa bencana yang memengaruhi wilayah seluas ratusan mil persegi. Namun, kebanyakan pemadaman sudah jauh lebih terlokalisasi, dengan hanya memengaruhi sebagian kecil subset layanan cloud atau sistem pelanggan. Pertimbangkan cakupan pemadaman secara menyeluruh untuk memastikan bahwa failover lintas Wilayah merupakan solusi yang tepat dan untuk memilih metode failover yang sesuai dengan situasi tersebut. Penggunaan pendekatan failover atau switchover bergantung pada skenario pemadaman tertentu:

- Failover—Gunakan pendekatan ini untuk melakukan pemulihan dari pemadaman yang tidak direncanakan. Dengan pendekatan ini, Anda melakukan failover lintas Wilayah ke salah satu klaster DB sekunder dalam basis data global Aurora Anda. RPO untuk pendekatan ini biasanya merupakan nilai bukan nol yang diukur dalam hitungan detik. Jumlah kehilangan data tergantung pada kelambatan replikasi database global Aurora Wilayah AWS pada saat kegagalan. Untuk mempelajari selengkapnya, lihat Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan.
- Switchover—Operasi ini sebelumnya disebut "failover terencana terkelola". Gunakan pendekatan ini untuk skenario terkontrol, seperti pemeliharaan operasional dan prosedur operasional terencana lainnya. Karena fitur ini menyinkronkan klaster DB sekunder dengan primer sebelum membuat perubahan lain, RPO adalah 0 (tidak ada kehilangan data). Untuk mempelajari selengkapnya, lihat Melakukan switchover untuk basis data global Amazon Aurora.

> **Note**
>
> Jika ingin melakukan switchover atau failover ke klaster DB Aurora sekunder headless, Anda harus terlebih dahulu menambahkan instans DB ke klaster tersebut. Untuk informasi selengkapnya tentang klaster DB headless, lihat Membuat klaster DB Aurora tanpa kepala di Wilayah sekunder.

Topik

- Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan
- Melakukan switchover untuk basis data global Amazon Aurora
- Mengelola RPO untuk basis data global berbasis Aurora PostgreSQL

Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan

Pada kesempatan yang sangat langka, basis data global Aurora Anda mungkin mengalami pemadaman yang tak terduga di Wilayah AWS primernya. Jika hal ini terjadi, klaster DB Aurora primer dan simpul penulisnya tidak akan tersedia, dan replikasi antara klaster DB primer dan sekunder akan terhenti. Untuk meminimalkan waktu henti (RTO) dan kehilangan data (RPO), Anda dapat segera melakukan failover lintas Wilayah.

Terdapat dua metode untuk melakukan failover dalam situasi pemulihan bencana:

- Failover terkelola—Metode ini direkomendasikan untuk pemulihan bencana. Jika Anda menggunakan metode ini, Aurora akan secara otomatis menambahkan kembali Wilayah primer yang lama ke basis data global sebagai Wilayah sekunder saat sudah tersedia lagi. Dengan begitu, topologi asli klaster global akan dipertahankan. Untuk mempelajari cara menggunakan metode ini, lihat Melakukan failover terkelola untuk basis data global Aurora.
- Failover manual—Metode alternatif ini dapat digunakan ketika failover terkelola bukan merupakan pilihan, misalnya, ketika Wilayah primer dan sekunder Anda menjalankan versi mesin yang tidak kompatibel. Untuk mempelajari cara menggunakan metode ini, lihat Melakukan failover manual untuk basis data global Aurora.

> **Important**
>
> Kedua metode failover tersebut dapat mengakibatkan hilangnya data transaksi tulis yang tidak direplikasi ke tujuan sekunder yang dipilih sebelum peristiwa failover terjadi. Namun, proses pemulihan yang mempromosikan instans DB pada klaster DB sekunder pilihan menjadi instans DB penulis primer akan memastikan bahwa data berada dalam kondisi yang konsisten secara transaksional.

Melakukan failover terkelola untuk basis data global Aurora

Pendekatan ini ditujukan untuk kelangsungan bisnis saat terjadi bencana alam Regional yang riil atau pemadaman tingkat layanan secara menyeluruh.

Selama failover terkelola, klaster primer Anda melakukan failover ke Wilayah sekunder yang dipilih sekaligus mempertahankan topologi replikasi yang ada pada basis data global Aurora. Klaster sekunder yang dipilih mempromosikan salah satu simpul hanya-bacanya ke status penulis penuh. Langkah ini memungkinkan klaster untuk mengambil peran sebagai klaster primer. Basis data Anda tidak akan tersedia untuk sementara saat klaster ini mengambil peran barunya. Data yang tidak direplikasi dari klaster primer lama ke klaster sekunder pilihan akan hilang saat klaster sekunder ini menjadi klaster primer yang baru.

> **Note**
>
> Anda hanya dapat melakukan failover basis data lintas Wilayah terkelola pada basis data global Aurora jika klaster DB primer dan sekunder memiliki versi mesin tingkat utama, minor,

dan patch yang sama. Namun, tingkat patch bisa berbeda, tergantung versi mesin kecil. Untuk informasi selengkapnya, lihat Kompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola. Jika versi mesin Anda tidak kompatibel, Anda dapat melakukan failover secara manual dengan mengikuti langkah-langkah dalam Melakukan failover manual untuk basis data global Aurora.

Untuk meminimalkan kehilangan data, sebaiknya lakukan hal berikut sebelum menggunakan fitur ini:

- Buat aplikasi menjadi offline untuk mencegah penulisan dikirim ke klaster primer basis data global Aurora.
- Periksa waktu keterlambatan untuk semua klaster DB Aurora sekunder dalam basis data global Aurora. Memilih Wilayah sekunder dengan keterlambatan replikasi minimum dapat meminimalkan kehilangan data dari Wilayah primer yang mengalami kegagalan. Untuk semua database global berbasis Aurora PostgreSQL dan untuk database global berbasis Aurora MySQL yang dimulai dengan versi mesin 3.04.0 dan lebih tinggi, atau 2.12.0 dan lebih tinggi, gunakan Amazon untuk melihat metrik untuk semua cluster DB sekunder. CloudWatch `AuroraGlobalDBRPOLag` Untuk basis data global berbasis Aurora MySQL versi minor yang lebih rendah, lihat metrik `AuroraGlobalDBReplicationLag`. Metrik ini menunjukkan seberapa terlambatnya (dalam milidetik) replikasi ke klaster sekunder dibandingkan ke klaster DB primer.

  Untuk informasi selengkapnya tentang CloudWatch metrik untuk Aurora, lihat. Metrik tingkat klaster untuk Amazon Aurora

Selama failover terkelola, klaster DB sekunder yang dipilih dipromosikan ke peran barunya sebagai klaster primer. Namun, klaster ini tidak mewarisi berbagai opsi konfigurasi klaster DB primer. Ketidakcocokan dalam konfigurasi dapat menyebabkan masalah performa, inkompatibilitas beban kerja, dan perilaku anomali lainnya. Untuk mencegah masalah tersebut, sebaiknya atasi perbedaan di antara klaster basis data global Aurora untuk konfigurasi berikut:

- Konfigurasi grup parameter klaster DB Aurora untuk klaster primer yang baru jika diperlukan—Anda dapat mengonfigurasi grup parameter klaster DB Aurora secara independen untuk masing-masing klaster Aurora dalam basis data global Aurora Anda. Karena itu, ketika Anda mempromosikan klaster DB sekunder untuk mengambil alih peran sebagai klaster primer, grup parameter dari klaster sekunder mungkin memiliki konfigurasi yang berbeda dengan klaster primer. Jika demikian, ubah grup parameter klaster DB sekunder yang dipromosikan agar sesuai dengan

pengaturan klaster primer Anda. Untuk mempelajari caranya, lihat Memodifikasi parameter basis data global Aurora.

- Konfigurasikan alat dan opsi pemantauan, seperti Amazon CloudWatch Events dan alarm — Konfigurasikan cluster DB yang dipromosikan dengan kemampuan logging, alarm, dan sebagainya yang sama sesuai kebutuhan untuk database global. Seperti grup parameter, konfigurasi untuk fitur ini tidak diwariskan dari klaster primer selama proses failover berlangsung. Beberapa CloudWatch metrik, seperti replikasi lag, hanya tersedia untuk Wilayah sekunder. Karena itu, failover akan mengubah cara Anda melihat metrik tersebut dan mengatur alarmnya, serta mengharuskan adanya perubahan pada dasbor yang ditentukan sebelumnya. Untuk informasi selengkapnya tentang klaster DB Aurora dan pemantauan, lihat Ikhtisar pemantauan Amazon Aurora.
- Konfigurasikan integrasi dengan AWS layanan lain — Jika database global Aurora Anda terintegrasi AWS dengan layanan, AWS Secrets Manager seperti, AWS Identity and Access Management Amazon S3, AWS Lambda dan, Anda perlu memastikan ini dikonfigurasi sesuai kebutuhan. Untuk informasi selengkapnya tentang cara mengintegrasi basis data global Aurora dengan IAM, Amazon S3, dan Lambda, lihat Menggunakan basis data global Amazon Aurora dengan layanan AWS lainnya. Untuk mempelajari Secrets Manager selengkapnya, lihat Cara mengotomatiskan replikasi rahasia secara AWS Secrets Manager keseluruhan. Wilayah AWS

Biasanya, klaster sekunder yang dipilih akan mengasumsikan peran primer dalam beberapa menit. Segera setelah simpul penulis Wilayah primer baru tersedia, Anda dapat menghubungkan aplikasi Anda ke simpul tersebut dan melanjutkan beban kerja Anda. Setelah mempromosikan klaster primer yang baru, Aurora secara otomatis membangun ulang semua klaster Wilayah sekunder tambahan.

Karena basis data global Aurora menggunakan replikasi asinkron, keterlambatan replikasi di masing-masing Wilayah sekunder dapat bervariasi. Aurora membangun kembali Wilayah sekunder ini untuk memiliki point-in-time data yang sama persis dengan cluster Region primer yang baru. Durasi penyelesaian tugas pembangunan ulang dapat memerlukan waktu beberapa menit hingga beberapa jam, bergantung pada ukuran volume penyimpanan dan jarak di antara Wilayah. Saat klaster Wilayah sekunder selesai dibuat ulang dari Wilayah primer yang baru, klaster ini menjadi tersedia untuk akses baca.

Segera setelah penulis primer baru dipromosikan dan tersedia, klaster Wilayah primer yang baru dapat menangani operasi baca dan tulis untuk basis data global Aurora. Pastikan Anda mengubah titik akhir untuk aplikasi agar dapat menggunakan titik akhir yang baru. Jika Anda menyetujui nama yang disediakan saat membuat basis data global Aurora, Anda dapat mengubah titik akhir dengan menghapus `-ro` dari string titik akhir klaster yang dipromosikan dalam aplikasi Anda.

Sebagai contoh, titik akhir klaster sekunder `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` menjadi `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` ketika klaster tersebut dipromosikan menjadi klaster primer.

Jika Anda menggunakan Proksi RDS, pastikan untuk mengalihkan operasi tulis aplikasi ke titik akhir baca/tulis yang sesuai dari proksi yang terkait dengan klaster primer baru. Titik akhir proksi ini mungkin merupakan titik akhir default atau titik akhir baca/tulis kustom. Untuk informasi selengkapnya, lihat Cara kerja titik akhir Proksi RDS dengan basis data global.

Untuk memulihkan topologi asli klaster basis data global, Aurora memantau ketersediaan Wilayah primer yang lama. Segera setelah Wilayah tersebut normal dan tersedia kembali, Aurora secara otomatis menambahkannya kembali ke klaster global sebagai Wilayah sekunder. Sebelum membuat volume penyimpanan baru di Wilayah primer yang lama, Aurora mencoba mengambil snapshot dari volume penyimpanan lama pada titik kegagalan. Hal ini dilakukan agar Anda dapat menggunakannya untuk memulihkan setiap data yang hilang. Jika operasi ini berhasil, Aurora menempatkan snapshot ini bernama “rds: unplanned-global-failover - *`name-of-old-primary-DB-cluster`* - *`timestamp`*" di bagian snapshot. AWS Management Console Anda juga dapat melihat snapshot ini tercantum dalam informasi yang ditampilkan oleh operasi DescribeDB API ClusterSnapshots.

> **Note**
>
> Snapshot dari volume penyimpanan lama adalah snapshot sistem yang tunduk pada periode retensi pencadangan yang dikonfigurasi pada klaster primer yang lama. Untuk mempertahankan snapshot ini di luar periode retensi, Anda dapat menyalin snapshot untuk disimpan sebagai snapshot manual. Untuk mempelajari selengkapnya tentang cara menyalin snapshot, termasuk harga, lihat Menyalin snapshot klaster DB.

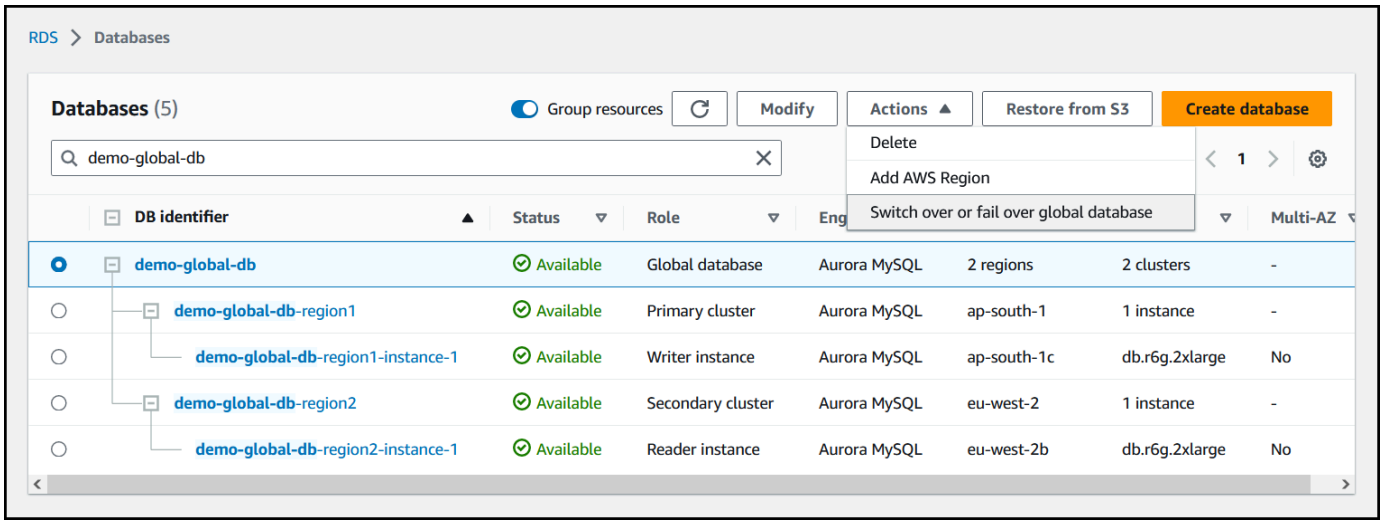
Setelah topologi asli dipulihkan, Anda dapat mengembalikan basis data global ke Wilayah primer asli dengan melakukan operasi switchover jika dianggap sebagai tindakan terbaik untuk bisnis dan beban kerja Anda. Untuk melakukannya, ikuti langkah-langkah yang ada di Melakukan switchover untuk basis data global Amazon Aurora.

Anda dapat gagal atas database global Aurora Anda menggunakan AWS Management Console, API AWS CLI, atau RDS.

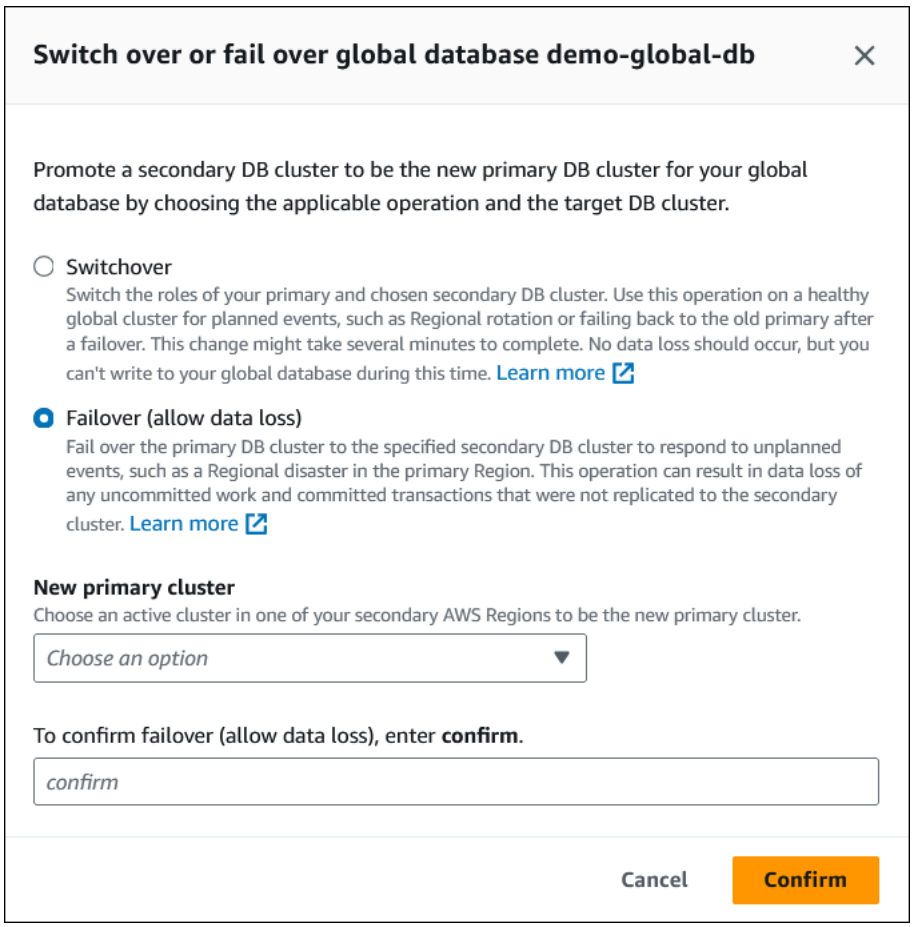
Konsol

Untuk melakukan failover terkelola pada basis data global Aurora:

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis Data, lalu temukan basis data global Aurora yang ingin Anda terapkan failover.
3. Pilih Alihkan atau lakukan failover basis data global dari menu Tindakan.

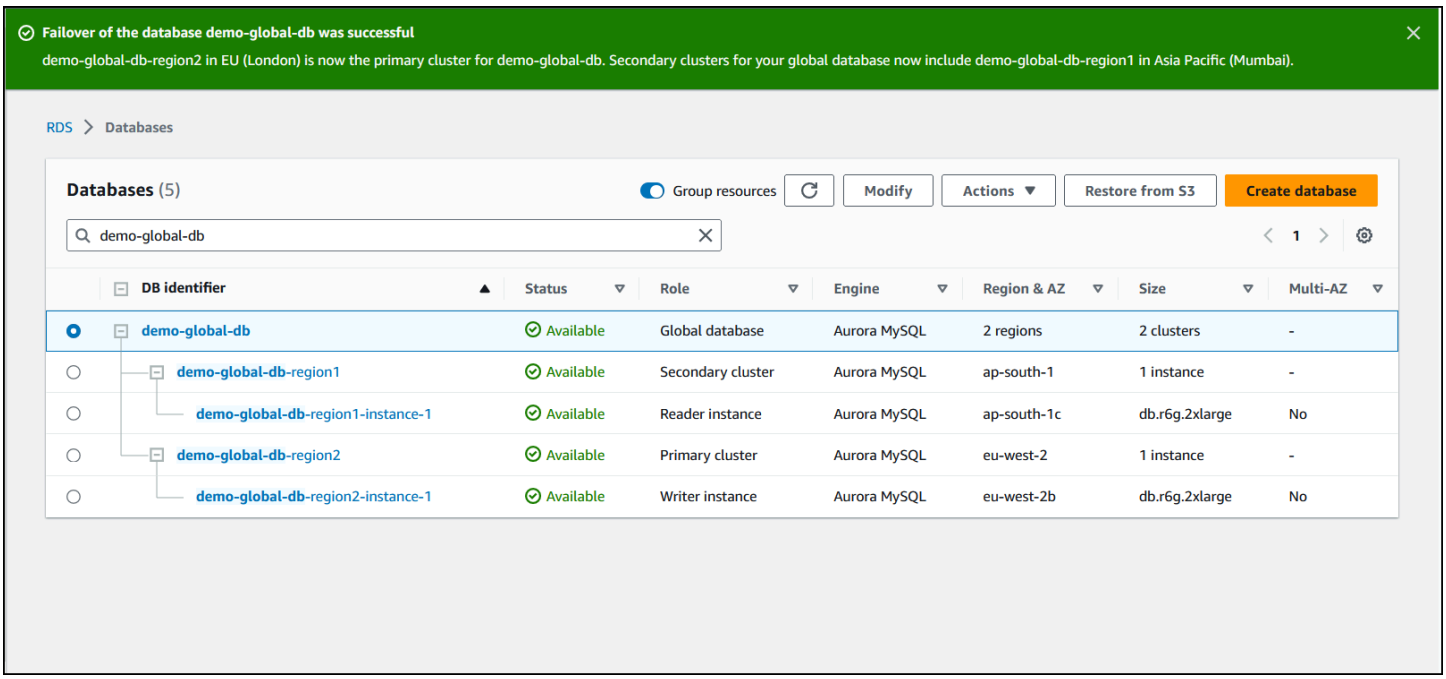


4. Pilih Failover (memungkinkan kehilangan data).



5. Untuk cluster primer baru, pilih klaster aktif di salah satu cluster sekunder Anda Wilayah AWS untuk menjadi cluster primer baru.
6. Masukkan **`confirm`**, lalu pilih Konfirmasi.

Setelah failover selesai, Anda dapat melihat klaster DB Aurora dan statusnya saat ini di daftar Basis Data, sebagaimana ditunjukkan pada gambar berikut.



AWS CLI

Untuk melakukan failover terkelola pada basis data global Aurora:

Gunakan perintah CLI `failover-global-cluster` untuk melakukan failover pada basis data global Aurora. Dengan perintah tersebut, loloskan nilai untuk parameter berikut.

- `--region`— Tentukan Wilayah AWS di mana cluster DB sekunder yang Anda inginkan menjadi primer baru untuk database global Aurora berjalan.
- `--global-cluster-identifier`—Tentukan nama basis data global Aurora.
- `--target-db-cluster-identifier`—Tentukan Amazon Resource Name (ARN) untuk klaster DB Aurora yang ingin Anda promosikan menjadi klaster primer baru untuk basis data global Aurora.
- `--allow-data-loss`—Secara eksplisit jadikan tindakan ini sebagai operasi failover, bukan operasi switchover. Operasi failover dapat membuat Anda kehilangan sejumlah data jika komponen replikasi asinkron belum selesai mengirim semua data yang direplikasi ke Wilayah sekunder.

Untuk Linux, macOS, atau Unix:

```
aws rds --region region_of_selected_secondary \
   failover-global-cluster --global-cluster-identifier global_database_id \
   --target-db-cluster-identifier arn_of_secondary_to_promote \
   --allow-data-loss
```

Untuk Windows:

```
aws rds --region region_of_selected_secondary ^
   failover-global-cluster --global-cluster-identifier global_database_id ^
   --target-db-cluster-identifier arn_of_secondary_to_promote ^
   --allow-data-loss
```

API RDS

Untuk gagal dalam database global Aurora, jalankan operasi FailoverGlobalClusterAPI.

Melakukan failover manual untuk basis data global Aurora

Dalam skenario tertentu, Anda mungkin tidak dapat menggunakan proses failover terkelola. Salah satu contohnya adalah jika klaster DB primer dan sekunder tidak menjalankan versi mesin yang kompatibel. Dalam kasus ini, Anda dapat mengikuti proses manual ini untuk melakukan failover basis data global ke Wilayah sekunder tujuan Anda.

> **Tip**
>
> Sebaiknya Anda memahami proses ini sebelum menggunakannya. Siapkan rencana untuk segera memulai saat pertama kali muncul tanda masalah tingkat Wilayah. Anda dapat siap mengidentifikasi Wilayah sekunder dengan kelambatan replikasi paling sedikit dengan menggunakan Amazon CloudWatch secara teratur untuk melacak waktu jeda untuk klaster sekunder. Pastikan Anda menguji rencana untuk memastikan prosedur sudah lengkap dan akurat, dan bahwa staf Anda terlatih untuk melakukan failover pemulihan bencana sebelum bencana benar-benar terjadi.

Untuk melakukan failover ke klaster sekunder secara manual setelah terjadi pemadaman tak terduga di Wilayah primer:

1. Berhenti mengeluarkan pernyataan DHTML dan operasi penulisan lainnya ke klaster Aurora DB utama di with the Wilayah AWS outage.
2. Identifikasi cluster Aurora DB dari sekunder Wilayah AWS untuk digunakan sebagai cluster DB primer baru. Jika Anda memiliki dua atau lebih sekunder Wilayah AWS dalam database global Aurora Anda, pilih cluster sekunder yang memiliki kelambatan replikasi paling sedikit.
3. Lepaskan klaster DB sekunder pilihan Anda dari basis data global Aurora.

Pelepasan klaster DB sekunder dari basis data global Aurora akan segera menghentikan replikasi dari klaster primer ke klaster sekunder tersebut dan mempromosikannya menjadi klaster DB Aurora yang berdiri sendiri dengan kapabilitas baca/tulis penuh. Klaster DB Aurora sekunder lainnya yang terkait dengan klaster primer di Wilayah yang mengalami pemadaman akan tetap tersedia dan dapat menerima panggilan dari aplikasi Anda. Klaster tersebut juga mengonsumsi sumber daya. Karena Anda membuat ulang basis data global Aurora, hapus klaster DB sekunder lainnya sebelum membuat basis data global Aurora yang baru dengan mengikuti langkah-langkah berikut. Tindakan ini akan mencegah inkonsistensi data di antara klaster DB dalam basis data global Aurora (masalah split-brain).

Untuk langkah-langkah pelepasan terperinci, lihat Menghapus klaster dari basis data global Amazon Aurora.

4. Konfigurasi ulang aplikasi Anda untuk mengirim semua operasi tulis ke klaster DB Aurora yang kini berdiri sendiri ini menggunakan titik akhir barunya. Jika Anda menyetujui nama yang disediakan ketika Anda membuat basis data global Aurora, Anda dapat mengubah titik akhir dengan menghapus `-ro` dari string titik akhir klaster dalam aplikasi Anda.

   Sebagai contoh, titik akhir klaster sekunder `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` menjadi `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` saat klaster tersebut dilepas dari basis data global Aurora.

   Klaster DB Aurora ini menjadi klaster primer untuk basis data global Aurora yang baru ketika Anda mulai menambahkan Wilayah ke dalamnya pada langkah berikutnya.

   Jika Anda menggunakan Proksi RDS, pastikan untuk mengalihkan operasi tulis aplikasi ke titik akhir baca/tulis yang sesuai dari proksi yang terkait dengan klaster primer baru. Titik akhir proksi ini mungkin merupakan titik akhir default atau titik akhir baca/tulis kustom. Untuk informasi selengkapnya, lihat Cara kerja titik akhir Proksi RDS dengan basis data global.

5. Tambahkan Wilayah AWS ke cluster DB. Saat Anda melakukannya, proses replikasi dari klaster primer ke klaster sekunder akan dimulai. Untuk langkah menambahkan Wilayah secara mendetail, lihat Menambahkan Wilayah AWS ke basis data global Amazon Aurora.

6. Tambahkan lebih banyak Wilayah AWS sesuai kebutuhan untuk membuat ulang topologi yang diperlukan untuk mendukung aplikasi Anda.

Pastikan penulisan aplikasi dikirim ke klaster DB Aurora yang benar, baik sebelum, selama, maupun sesudah membuat perubahan ini. Tindakan ini akan mencegah inkonsistensi data di antara klaster DB dalam basis data global Aurora (masalah split-brain).

Jika Anda mengonfigurasi ulang sebagai respons terhadap pemadaman di sebuah Wilayah AWS, Anda dapat menjadikannya primer lagi setelah pemadaman diselesaikan. Wilayah AWS Untuk melakukannya, Anda harus menambahkan Wilayah AWS yang lama ke basis data global baru, lalu menggunakan proses switchover untuk mengalihkan perannya. Basis data global Aurora harus menggunakan versi Aurora PostgreSQL atau Aurora MySQL yang mendukung switchover. Untuk informasi selengkapnya, lihat Melakukan switchover untuk basis data global Amazon Aurora.

Melakukan switchover untuk basis data global Amazon Aurora

> **Note**
>
> Switchover sebelumnya disebut "failover terencana terkelola".

Dengan menggunakan switchover, Anda dapat mengubah Wilayah cluster utama Anda secara rutin. Pendekatan ini ditujukan untuk skenario yang terkontrol, seperti pemeliharaan operasional dan prosedur operasional terencana lainnya.

Terdapat tiga kasus umum untuk penggunaan switchover.

- Untuk persyaratan "rotasi regional" yang diberlakukan pada industri tertentu. Misalnya, peraturan layanan keuangan mungkin menginginkan sistem tier-0 untuk beralih ke Wilayah yang berbeda selama beberapa bulan untuk memastikan prosedur pemulihan bencana dilaksanakan secara teratur.
- Untuk aplikasi Multi-region follow-the-sun "". Misalnya, suatu bisnis mungkin ingin menyediakan penulisan dengan latensi lebih rendah di berbagai Wilayah berdasarkan jam kerja di zona waktu yang berbeda.
- Sebagai zero-data-loss metode untuk gagal kembali ke Wilayah primer asli setelah failover.

> **Note**
>
> Switchover dirancang untuk digunakan pada basis data global Aurora yang berfungsi baik. Untuk pulih dari pemadaman yang tak terduga, ikuti prosedur yang sesuai di Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan.

Untuk melakukan switchover, klaster DB sekunder target Anda harus menjalankan versi mesin yang sama persis dengan klaster primer, termasuk tingkat patch-nya, tergantung pada versi mesin. Untuk informasi selengkapnya, lihat Kompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola. Sebelum Anda memulai switchover, periksa versi mesin di klaster global untuk memastikan versi mesin tersebut mendukung switchover lintas Wilayah terkelola, lalu tingkatkan jika diperlukan.

Selama proses switchover, Aurora mengalihkan klaster primer Anda ke Wilayah sekunder pilihan sambil mempertahankan topologi replikasi yang ada pada basis data global Anda. Sebelum memulai proses switchover, Aurora menunggu hingga semua klaster Wilayah sekunder disinkronkan sepenuhnya dengan klaster Wilayah primer. Selanjutnya, klaster DB di Wilayah primer menjadi klaster hanya-baca, dan klaster sekunder yang dipilih akan mempromosikan salah satu simpul hanya-bacanya menjadi status penulis penuh. Mempromosikan simpul ini menjadi penulis memungkinkan klaster sekunder mengambil peran klaster primer. Karena semua klaster sekunder disinkronkan dengan klaster primer pada awal proses, klaster primer yang baru akan melanjutkan operasi untuk basis data global Aurora tanpa kehilangan data apa pun. Basis data Anda tidak tersedia untuk sementara selama klaster primer dan klaster sekunder yang dipilih mengambil peran barunya masing-masing.

Untuk mengoptimalkan ketersediaan aplikasi, sebaiknya lakukan hal berikut sebelum menggunakan fitur ini:

- Lakukan operasi ini di luar jam sibuk atau pada waktu lainnya saat operasi penulisan ke klaster DB primer tidak banyak.
- Buat aplikasi menjadi offline untuk mencegah penulisan dikirim ke klaster primer basis data global Aurora.
- Periksa waktu keterlambatan untuk semua klaster DB Aurora sekunder dalam basis data global Aurora. Untuk semua database global berbasis Aurora PostgreSQL dan untuk database global berbasis Aurora MySQL yang dimulai dengan versi mesin 3.04.0 dan lebih tinggi atau 2.12.0 dan lebih tinggi, gunakan Amazon untuk melihat metrik untuk semua cluster DB sekunder. CloudWatch `AuroraGlobalDBRPOLag` Untuk basis data global berbasis Aurora MySQL versi minor yang lebih rendah, lihat metrik `AuroraGlobalDBReplicationLag`. Metrik ini menunjukkan seberapa terlambatnya (dalam milidetik) replikasi ke klaster sekunder dibandingkan ke klaster DB primer. Nilai ini berbanding lurus dengan waktu yang diperlukan Aurora untuk menyelesaikan switchover. Karena itu, semakin besar nilai keterlambatan, semakin lama durasi switchover.

Untuk informasi selengkapnya tentang CloudWatch metrik untuk Aurora, lihat. Metrik tingkat klaster untuk Amazon Aurora

Selama proses switchover, klaster DB sekunder yang dipilih akan dipromosikan ke peran barunya sebagai primer. Namun, klaster ini tidak mewarisi berbagai opsi konfigurasi klaster DB primer. Ketidakcocokan dalam konfigurasi dapat menyebabkan masalah performa, inkompatibilitas beban kerja, dan perilaku anomali lainnya. Untuk mencegah masalah tersebut, sebaiknya atasi perbedaan di antara klaster basis data global Aurora untuk konfigurasi berikut:

- Konfigurasi grup parameter klaster DB Aurora untuk klaster primer yang baru jika diperlukan— Anda dapat mengonfigurasi grup parameter klaster DB Aurora secara independen untuk masing-masing klaster Aurora dalam basis data global Aurora Anda. Hal ini berarti ketika Anda mempromosikan klaster DB sekunder untuk mengambil alih peran primer, grup parameter dari klaster sekunder mungkin memiliki konfigurasi yang berbeda dengan klaster primer. Jika demikian, ubah grup parameter klaster DB sekunder yang dipromosikan agar sesuai dengan pengaturan klaster primer Anda. Untuk mempelajari caranya, lihat Memodifikasi parameter basis data global Aurora.
- Konfigurasikan alat dan opsi pemantauan, seperti Amazon CloudWatch Events dan alarm — Konfigurasikan cluster DB yang dipromosikan dengan kemampuan logging, alarm, dan sebagainya yang sama sesuai kebutuhan untuk database global. Seperti grup parameter, konfigurasi untuk fitur ini tidak diwariskan dari klaster primer selama proses switchover berlangsung. Beberapa CloudWatch metrik, seperti replikasi lag, hanya tersedia untuk Wilayah sekunder. Karena itu, switchover akan mengubah cara Anda melihat metrik tersebut dan mengatur alarmnya, serta mengharuskan adanya perubahan pada dasbor yang ditentukan sebelumnya. Untuk informasi selengkapnya tentang klaster DB Aurora dan pemantauan, lihat Ikhtisar pemantauan Amazon Aurora.
- Konfigurasikan integrasi dengan AWS layanan lain — Jika database global Aurora Anda terintegrasi AWS dengan layanan, AWS Secrets Manager seperti, AWS Identity and Access Management Amazon S3, AWS Lambda dan, pastikan untuk mengonfigurasi integrasi Anda dengan layanan ini sesuai kebutuhan. Untuk informasi selengkapnya tentang cara mengintegrasi basis data global Aurora dengan IAM, Amazon S3, dan Lambda, lihat Menggunakan basis data global Amazon Aurora dengan layanan AWS lainnya. Untuk mempelajari Secrets Manager selengkapnya, lihat Cara mengotomatiskan replikasi rahasia secara AWS Secrets Manager keseluruhan. Wilayah AWS

> **Note**
>
> Biasanya, switchover peran dapat memerlukan waktu hingga beberapa menit. Namun, membangun klaster sekunder tambahan dapat berlangsung selama beberapa menit hingga beberapa jam, bergantung pada ukuran basis data dan jarak fisik di antara Wilayah.

Saat proses switchover selesai, klaster DB Aurora yang dipromosikan dapat menangani operasi penulisan untuk basis data global Aurora. Pastikan Anda mengubah titik akhir untuk aplikasi agar dapat menggunakan titik akhir yang baru. Jika Anda menyetujui nama yang disediakan saat membuat basis data global Aurora, Anda dapat mengubah titik akhir dengan menghapus `-ro` dari string titik akhir klaster yang dipromosikan dalam aplikasi Anda.

Sebagai contoh, titik akhir klaster sekunder `my-global.cluster-ro-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` menjadi `my-global.cluster-aaaaaabbbbbb.us-west-1.rds.amazonaws.com` ketika klaster tersebut dipromosikan menjadi klaster primer.

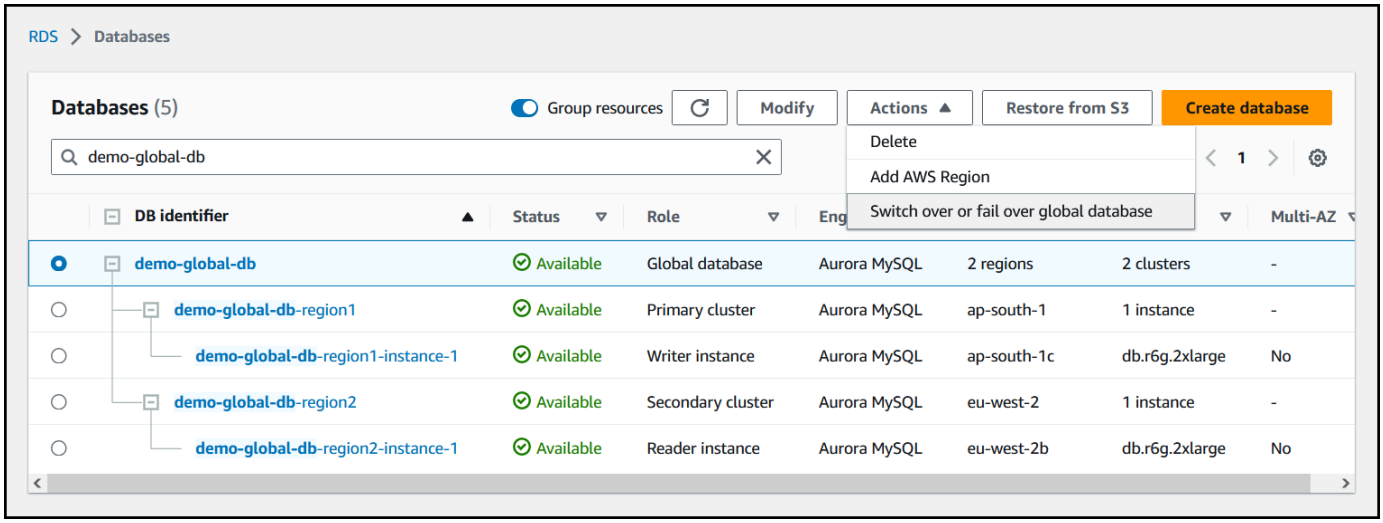
Jika Anda menggunakan Proksi RDS, pastikan untuk mengalihkan operasi tulis aplikasi ke titik akhir baca/tulis yang sesuai dari proksi yang terkait dengan klaster primer baru. Titik akhir proksi ini mungkin merupakan titik akhir default atau titik akhir baca/tulis kustom. Untuk informasi selengkapnya, lihat Cara kerja titik akhir Proksi RDS dengan basis data global.

Anda dapat beralih ke database global Aurora Anda menggunakan AWS Management Console, API AWS CLI, atau RDS.

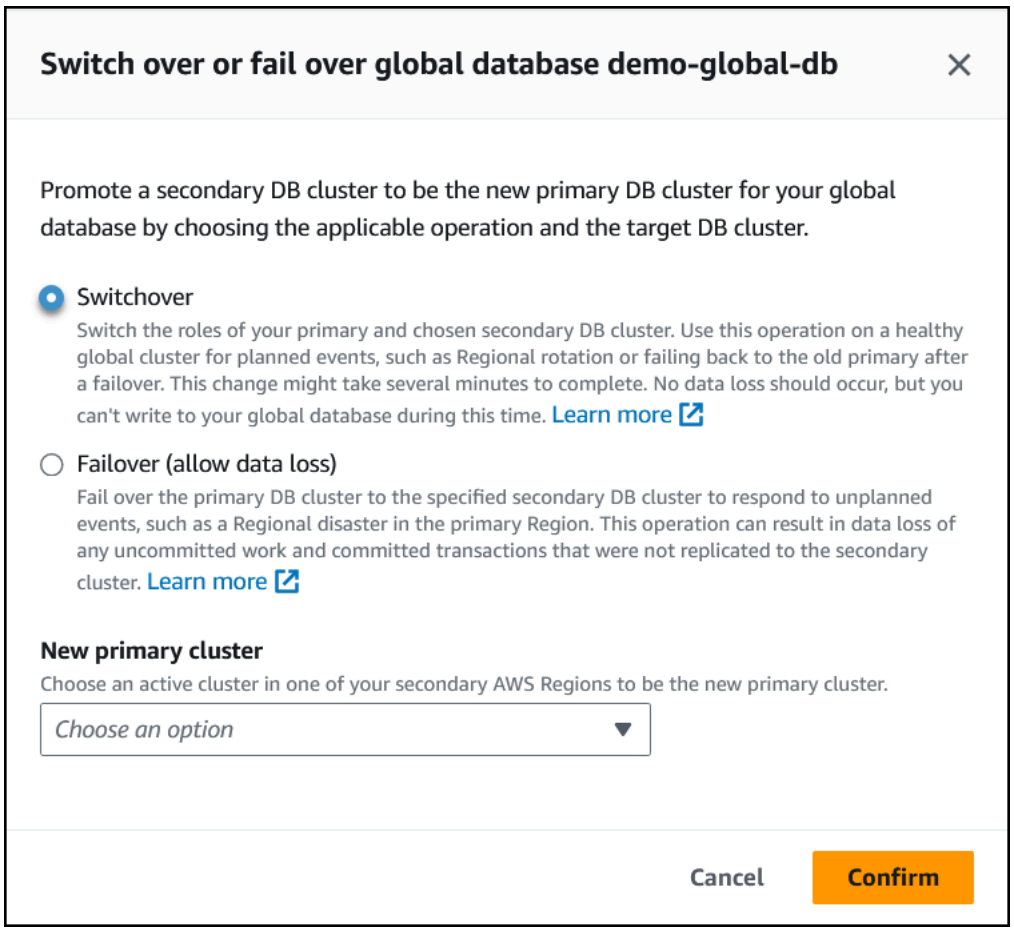
Konsol

Cara melakukan switchover pada basis data global Aurora

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis Data, lalu temukan basis data global Aurora yang ingin Anda terapkan switchover.
3. Pilih Alihkan atau lakukan failover basis data global dari menu Tindakan.

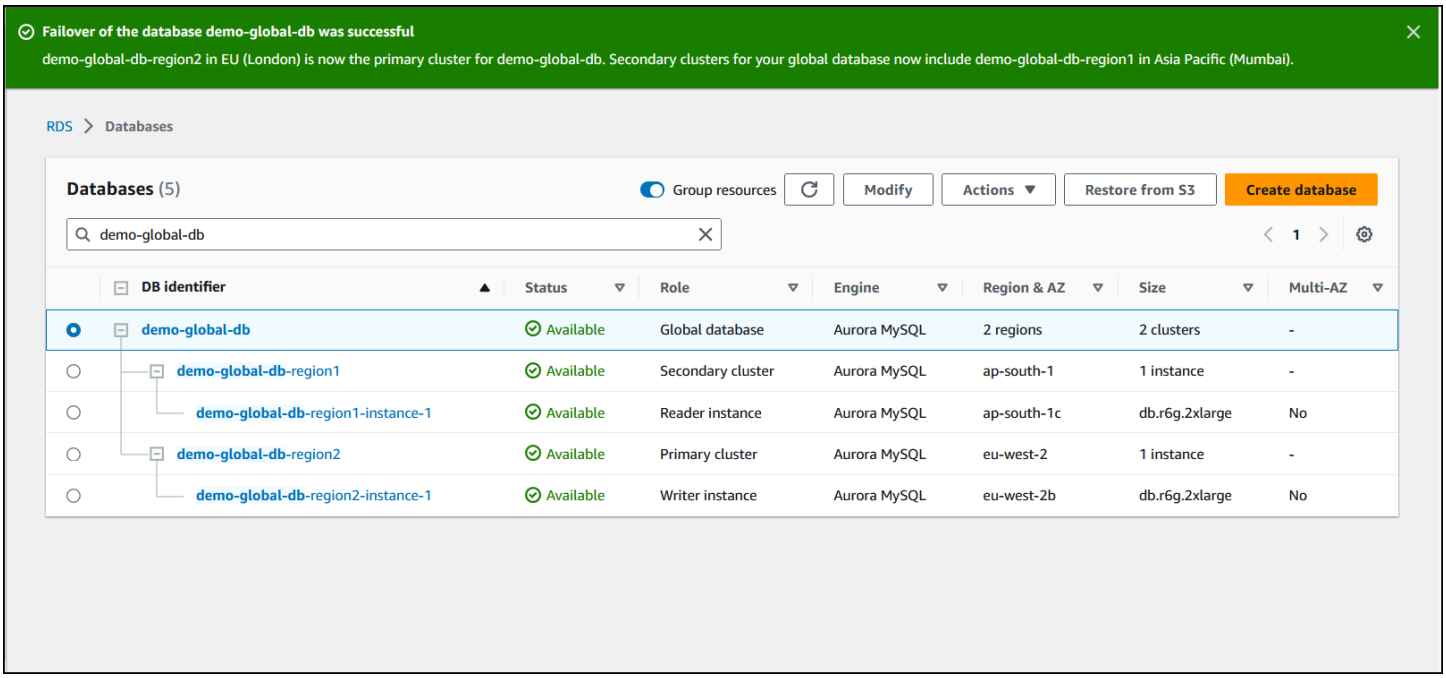


4. Pilih Switchover.



5. Untuk cluster primer baru, pilih klaster aktif di salah satu cluster sekunder Anda Wilayah AWS untuk menjadi cluster primer baru.
6. Pilih Konfirmasi.

Setelah switchover selesai, Anda dapat melihat klaster DB Aurora dan perannya saat ini di daftar Basis Data, sebagaimana ditunjukkan pada gambar berikut.



AWS CLI

Untuk melakukan switchover pada basis data global Aurora:

Gunakan perintah CLI `switchover-global-cluster` untuk melakukan switchover pada basis data global Aurora. Dengan perintah tersebut, loloskan nilai untuk parameter berikut.

- `--region`— Tentukan Wilayah AWS di mana cluster DB utama dari database global Aurora berjalan.
- `--global-cluster-identifier`—Tentukan nama basis data global Aurora.
- `--target-db-cluster-identifier`—Tentukan Amazon Resource Name (ARN) untuk klaster DB Aurora yang ingin Anda promosikan menjadi klaster primer basis data global Aurora.

Untuk Linux, macOS, atau Unix:

```
aws rds --region region_of_primary \
   switchover-global-cluster --global-cluster-identifier global_database_id \
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

Untuk Windows:

```
aws rds --region region_of_primary ^
   switchover-global-cluster --global-cluster-identifier global_database_id ^
  --target-db-cluster-identifier arn_of_secondary_to_promote
```

API RDS

Untuk beralih ke database global Aurora, jalankan operasi SwitchoverGlobalClusterAPI.

Mengelola RPO untuk basis data global berbasis Aurora PostgreSQL

Dengan basis data global berbasis Aurora PostgreSQL, Anda dapat mengelola sasaran titik pemulihan (RPO) untuk basis data global Aurora menggunakan parameter `rds.global_db_rpo`. RPO mewakili jumlah maksimum data yang dapat hilang ketika terjadi pemadaman.

Ketika Anda mengatur RPO untuk basis data global berbasis Aurora PostgreSQL, Aurora memantau Waktu keterlambatan RPO dari semua klaster sekunder untuk memastikan bahwa setidaknya satu klaster sekunder tetap dalam jendela target RPO. Waktu keterlambatan RPO adalah salah satu metrik berbasis waktu.

RPO digunakan ketika database Anda melanjutkan operasi di baru Wilayah AWS setelah failover. Aurora mengevaluasi RPO dan waktu keterlambatan RPO untuk melakukan commit (atau memblokir) transaksi pada klaster primer sebagai berikut:

- Menjalankan commit transaksi jika setidaknya satu klaster DB sekunder memiliki waktu keterlambatan RPO lebih rendah dari RPO.
- Memblokir transaksi jika semua klaster DB sekunder memiliki waktu keterlambatan RPO yang lebih tinggi dari RPO. Aurora juga mencatat peristiwa ke file log PostgreSQL dan mengeluarkan peristiwa "tunggu" yang menunjukkan sesi terblokir.

Dengan kata lain, jika semua klaster sekunder berada di belakang RPO target, Aurora akan menjeda transaksi pada klaster primer hingga setidaknya satu klaster sekunder mengimbangi. Transaksi yang dijeda akan dilanjutkan dan di-commit segera setelah waktu keterlambatan dari setidaknya satu klaster DB sekunder menjadi lebih rendah dari RPO. Hasilnya adalah bahwa tidak ada transaksi yang dapat di-commit hingga RPO terpenuhi.

Parameter `rds.global_db_rpo` bersifat dinamis. Jika Anda memutuskan untuk tidak menghentikan semua transaksi tulis hingga keterlambatan cukup berkurang, Anda dapat meresetnya dengan cepat. Dalam kasus ini, Aurora akan menerima dan mengimplementasikan perubahan setelah beberapa saat.

> **Important**
>
> Dalam basis data global yang hanya memiliki dua Wilayah, sebaiknya jangan ubah nilai default parameter `rds.global_db_rpo` di grup parameter Wilayah sekunder. Jika melakukannya, failover ke Wilayah ini akibat hilangnya Wilayah primer dapat menyebabkan Aurora menjeda transaksi. Sebaiknya tunggu hingga Aurora menyelesaikan pembangunan ulang klaster di Wilayah lama yang mengalami kegagalan sebelum Anda mengubah parameter ini untuk menerapkan RPO maksimum.

Jika menetapkan parameter ini sebagaimana diuraikan berikutnya, Anda juga dapat memantau metrik yang dihasilkannya. Anda dapat melakukannya menggunakan `psql` atau alat lain untuk membuat kueri klaster DB primer basis data global Aurora dan mendapatkan informasi terperinci tentang operasi basis data global berbasis Aurora PostgreSQL. Untuk mempelajari caranya, lihat Memantau basis data global berbasis Aurora PostgreSQL.

Topik

- Mengatur sasaran titik pemulihan
- Melihat sasaran titik pemulihan
- Menonaktifkan sasaran titik pemulihan

Mengatur sasaran titik pemulihan

Parameter `rds.global_db_rpo` mengontrol pengaturan RPO untuk basis data PostgreSQL. Parameter ini didukung oleh Aurora PostgreSQL. Nilai yang valid untuk `rds.global_db_rpo` berkisar dari 20 detik hingga 2.147.483.647 detik (68 tahun). Pilih nilai yang realistis untuk memenuhi kasus penggunaan dan kebutuhan bisnis Anda. Misalnya, Anda mungkin ingin mengizinkan durasi RPO hingga 10 menit, sehingga Anda menetapkan nilai ke 600.

Anda dapat menetapkan nilai ini untuk basis data global berbasis Aurora PostgreSQL menggunakan AWS Management Console, AWS CLI, atau API RDS.

Konsol

Cara mengatur RPO

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

2. Pilih klaster primer basis data global Aurora, lalu buka tab Konfigurasi untuk menemukan grup parameter klaster DB. Sebagai contoh, grup parameter default untuk klaster DB primer yang menjalankan Aurora PostgreSQL 11.7 adalah `default.aurora-postgresql11`.

   Grup parameter tidak dapat diedit secara langsung. Sebaliknya, lakukan hal berikut:

   - Buat grup parameter klaster DB kustom menggunakan grup parameter default yang sesuai sebagai titik awal. Misalnya, buat grup parameter klaster DB kustom berdasarkan `default.aurora-postgresql11`.
   - Pada grup parameter klaster DB kustom, tetapkan nilai parameter rds.global_db_rpo agar sesuai dengan kasus penggunaan Anda. Nilai yang valid berkisar dari 20 detik hingga nilai integer maksimum sebesar 2.147.483.647 (68 tahun).
   - Terapkan grup parameter klaster DB yang telah dimodifikasi ke klaster DB Aurora Anda.

Untuk informasi selengkapnya, lihat Mengubah parameter dalam grup parameter klaster DB.

AWS CLI

Untuk mengatur `rds.global_db_rpo` parameter, gunakan perintah modify-db-cluster-parameter-group CLI. Dalam perintahnya, tentukan nama grup parameter klaster primer dan nilai untuk parameter RPO.

Contoh berikut mengatur RPO ke 600 detik (10 menit) untuk grup parameter klaster DB primer, `my_custom_global_parameter_group`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name my_custom_global_parameter_group \
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds modify-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name my_custom_global_parameter_group ^
    --parameters
 "ParameterName=rds.global_db_rpo,ParameterValue=600,ApplyMethod=immediate"
```

API RDS

Untuk memodifikasi `rds.global_db_rpo` parameter, gunakan operasi Amazon RDS ModifyDB API ClusterParameterGroup.

Melihat sasaran titik pemulihan

Sasaran titik pemulihan (RPO) dari basis data global disimpan di parameter `rds.global_db_rpo` untuk setiap klaster DB. Anda dapat terhubung ke titik akhir klaster sekunder yang ingin Anda lihat dan menggunakan `psql` untuk membuat kueri instans bagi nilai ini.

```
db-name=>show rds.global_db_rpo;
```

Jika parameter ini belum diatur, kueri akan menampilkan sebagai berikut:

```
rds.global_db_rpo
-------------------
 -1
(1 row)
```

Respons selanjutnya ini berasal dari klaster DB sekunder yang memiliki pengaturan RPO 1 menit.

```
rds.global_db_rpo
-------------------
 60
(1 row)
```

Anda juga dapat menggunakan CLI untuk memperoleh nilai guna mengetahui apakah `rds.global_db_rpo` aktif pada salah satu klaster DB Aurora dengan menggunakan CLI untuk memperoleh nilai semua parameter `user` untuk klaster tersebut.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-cluster-parameters \
 --db-cluster-parameter-group-name lab-test-apg-global \
 --source user
```

Untuk Windows:

```
aws rds describe-db-cluster-parameters ^
 --db-cluster-parameter-group-name lab-test-apg-global *
```

```
 --source user
```

Perintah akan menampilkan output serupa dengan yang berikut untuk semua parameter `user` selain parameter klaster DB `default-engine` atau `system`.

```
{
    "Parameters": [
        {
            "ParameterName": "rds.global_db_rpo",
            "ParameterValue": "60",
            "Description": "(s) Recovery point objective threshold, in seconds, that
 blocks user commits when it is violated.",
            "Source": "user",
            "ApplyType": "dynamic",
            "DataType": "integer",
            "AllowedValues": "20-2147483647",
            "IsModifiable": true,
            "ApplyMethod": "immediate",
            "SupportedEngineModes": [
                "provisioned"
            ]
        }
    ]
}
```

Untuk mempelajari selengkapnya tentang cara melihat parameter dari grup parameter klaster, lihat Melihat nilai parameter untuk grup parameter klaster DB.

Menonaktifkan sasaran titik pemulihan

Untuk menonaktifkan RPO, reset parameter `rds.global_db_rpo`. Anda dapat mengatur ulang parameter menggunakan AWS Management Console, AWS CLI, atau RDS API.

Konsol

Cara menonaktifkan RPO

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup parameter.
3. Dalam daftar, pilih grup parameter klaster DB primer Anda.

4. Pilih Edit parameter.
5. Pilih kotak di sebelah parameter rds.global_db_rpo.
6. Pilih Reset.
7. Saat layar menampilkan Reset parameter dalam grup parameter DB, pilih Reset parameter.

Untuk informasi selengkapnya tentang cara mereset parameter dengan konsol, lihat Mengubah parameter dalam grup parameter klaster DB.

AWS CLI

Untuk mengatur ulang `rds.global_db_rpo` parameter, gunakan perintah reset-db-cluster-parameter-group.

Untuk Linux, macOS, atau Unix:

```
aws rds reset-db-cluster-parameter-group \
    --db-cluster-parameter-group-name global_db_cluster_parameter_group \
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

Untuk Windows:

```
aws rds reset-db-cluster-parameter-group ^
    --db-cluster-parameter-group-name global_db_cluster_parameter_group ^
    --parameters "ParameterName=rds.global_db_rpo,ApplyMethod=immediate"
```

API RDS

Untuk mengatur ulang `rds.global_db_rpo` parameter, gunakan operasi Amazon RDS API ClusterParameterGroupResetDB.

Memantau basis data global Amazon Aurora

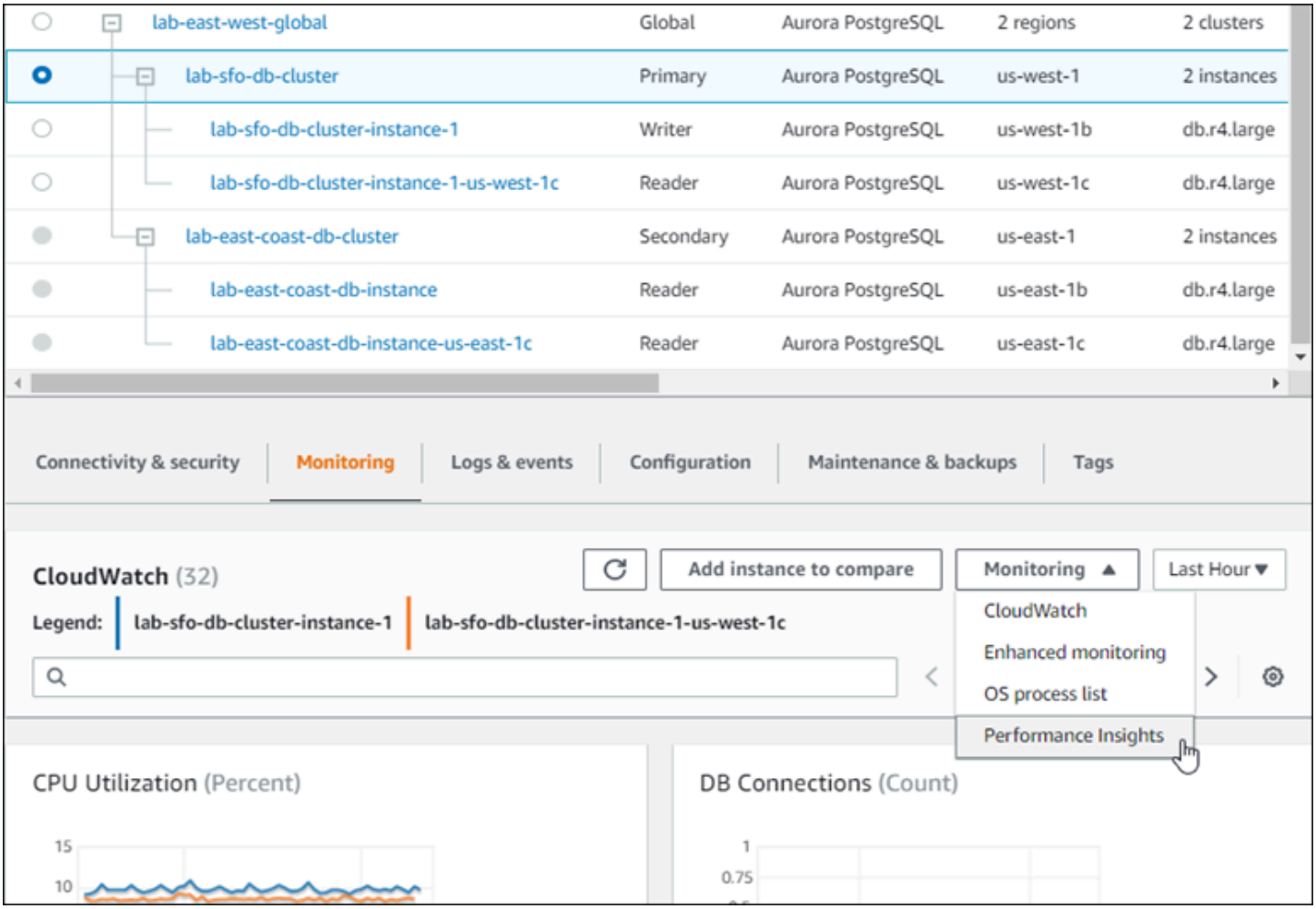
Ketika Anda membuat klaster DB Aurora yang membentuk basis data global Aurora Anda, Anda dapat memilih banyak pilihan yang memungkinkan Anda memantau kinerja DB klaster Anda. Opsi ini mencakup hal berikut:

- Wawasan Performa Amazon RDS - Mengaktifkan skema kinerja di mesin basis data Aurora yang mendasari. Untuk mempelajari selengkapnya tentang Wawasan Performa dan basis data global

Aurora, lihat Memantau basis data global Amazon Aurora dengan Wawasan Performa Amazon RDS.

- Pemantauan yang ditingkatkan – Menghasilkan metrik untuk pemanfaatan proses atau thread pada CPU. Untuk mempelajari tentang pemantauan yang ditingkatkan, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan.
- Log Amazon CloudWatch – Menerbitkan jenis log tertentu ke Log CloudWatch. Log kesalahan diterbitkan secara default, tetapi Anda dapat memilih log lain khusus untuk mesin basis data Aurora Anda.
  - Untuk klaster DB Aurora berbasis Aurora MySQL, Anda dapat mengekspor log audit, log umum, dan log kueri lambat.
  - Untuk klaster DB Aurora berbasis Aurora PostgreSQL, Anda dapat mengekspor log PostgreSQL.
- Untuk basis data global berbasis Aurora MySQL, Anda dapat mengkueri tabel `information_schema` tertentu untuk memeriksa status basis data global Aurora Anda dan instansnya. Untuk mempelajari caranya, lihat Memantau basis data global berbasis Aurora MySQL.
- Untuk basis data global berbasis Aurora PostgreSQL, Anda dapat menggunakan fungsi-fungsi tertentu untuk memeriksa status basis data global Aurora Anda dan instansnya. Untuk mempelajari caranya, lihat Memantau basis data global berbasis Aurora PostgreSQL.

Tangkapan layar berikut menunjukkan beberapa pilihan yang tersedia pada tab Pemantauan klaster DB Aurora primer dalam basis data global Aurora.



Untuk informasi selengkapnya, lihat Memantau metrik di klaster Amazon Aurora.

Memantau basis data global Amazon Aurora dengan Wawasan Performa Amazon RDS

Anda dapat menggunakan Wawasan Performa Amazon RDS untuk basis data global Aurora Anda. Anda mengaktifkan fitur ini secara individual, untuk setiap klaster DB Aurora dalam basis data global Aurora Anda. Untuk melakukannya, pilih Aktifkan Wawasan Performa di bagian Konfigurasi tambahan pada halaman Buat basis data. Atau Anda dapat memodifikasi klaster DB Aurora Anda untuk menggunakan fitur ini setelah tersedia dan berjalan. Anda dapat mengaktifkan atau menonaktifkan Wawasan Performa untuk setiap klaster yang merupakan bagian dari basis data global Aurora Anda.

Pernyataan yang dibuat oleh Wawasan Performa berlaku untuk setiap klaster dalam global basis data. Saat Anda menambahkan Wilayah AWS sekunder baru ke basis data global Aurora yang sudah

menggunakan Wawasan Performa, pastikan Anda mengaktifkan Wawasan Performa di klaster yang baru ditambahkan. Klaster tersebut tidak mewarisi pengaturan Wawasan Performa dari basis data global yang sudah ada.

Anda dapat beralih Wilayah AWS saat melihat halaman Wawasan Performa untuk instans DB yang terpasang pada basis data global. Namun, Anda mungkin tidak dapat langsung melihat informasi kinerja setelah beralih Wilayah AWS. Meskipun instans DB mungkin memiliki nama yang identik di setiap Wilayah AWS, URL Wawasan Performa yang terkait berbeda untuk setiap instans DB. Setelah beralih Wilayah AWS, pilih nama instans DB lagi di panel navigasi Wawasan Performa.

Untuk instans DB yang dikaitkan dengan global basis data, faktor yang memengaruhi kinerja mungkin berbeda di setiap Wilayah AWS. Misalnya, instans DB di setiap Wilayah AWS mungkin memiliki kapasitas yang berbeda.

Untuk mempelajari lebih lanjut tentang menggunakan Wawasan Performa, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

Memantau basis data global Aurora dengan Aliran Aktivitas Basis Data

Dengan menggunakan fitur Aliran Aktivitas Basis Data, Anda dapat memantau dan mengatur alarm untuk aktivitas audit di klaster DB di basis data global Anda. Anda memulai aliran aktivitas basis data pada setiap klaster DB secara terpisah. Setiap klaster mengirimkan data audit ke aliran Kinesis-nya sendiri dalam Wilayah AWS-nya sendiri. Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.

Memantau basis data global berbasis Aurora MySQL

Untuk melihat status basis data global berbasis Aurora MySQL, lakukan kueri pada tabel information_schema.aurora_global_db_status dan information_schema.aurora_global_db_instance_status.

> **Note**
>
> Tabel `information_schema.aurora_global_db_status` dan `information_schema.aurora_global_db_instance_status` hanya tersedia dengan Aurora MySQL versi 3.04.0 dan basis data global yang lebih tinggi.

Memantau basis data global Aurora berbasis Aurora MySQL

1. Buat koneksi ke titik akhir klaster primer basis data global yang menggunakan klien MySQL. Untuk informasi selengkapnya tentang cara membuat koneksi, lihat Terhubung ke basis data global Amazon Aurora.
2. Lakukan kueri ke tabel `information_schema.aurora_global_db_status` dalam perintah mysql untuk membuat daftar volume primer dan sekunder. Kueri ini memunculkan waktu lag klaster DB sekunder basis data global, seperti pada contoh berikut.

```
mysql> select * from information_schema.aurora_global_db_status;
```

```
AWS_REGION | HIGHEST_LSN_WRITTEN | DURABILITY_LAG_IN_MILLISECONDS |
 RPO_LAG_IN_MILLISECONDS | LAST_LAG_CALCULATION_TIMESTAMP | OLDEST_READ_VIEW_TRX_ID
-----------+---------------------+--------------------------------
+-------------------------+---------------------------------
+------------------------
us-east-1  |           183537946 |                              0   |
      0 |  1970-01-01 00:00:00.000000     |               0
us-west-2  |           183537944 |                              428 |
      0 |  2023-02-18 01:26:41.925000     |               20806982
(2 rows)
```

Output mencakup baris untuk setiap klaster DB basis data global yang berisi kolom berikut:

- AWS_REGION – Wilayah AWS tempat klaster DB ini berada. Untuk tabel yang berisi daftar Wilayah AWS berdasarkan mesin, lihat Wilayah dan Zona Ketersediaan.
- HIGHEST_LSN_WRITTEN – Log sequence number (LSN) tertinggi yang saat ini tertulis pada klaster DB ini.

  Log sequence number (LSN) adalah nomor urutan unik yang mengidentifikasi catatan di log transaksi basis data. LSN diurutkan sedemikian rupa sehingga LSN yang lebih besar mewakili transaksi berikutnya.
- DURABILITY_LAG_IN_MILLISECONDS — Perbedaan nilai stempel waktu antara `HIGHEST_LSN_WRITTEN` di klaster DB sekunder dan `HIGHEST_LSN_WRITTEN` di klaster DB primer. Nilai ini selalu 0 pada klaster DB primer basis data global Aurora.
- RPO_LAG_IN_MILLISECONDS – Lag sasaran titik pemulihan (RPO). Lag RPO adalah waktu yang dibutuhkan untuk COMMIT transaksi pengguna terbaru untuk disimpan di klaster DB

sekunder setelah disimpan di klaster DB primer basis data global Aurora. Nilai ini selalu 0 pada klaster DB primer basis data global Aurora.

Secara sederhana, metrik ini menghitung sasaran titik pemulihan untuk setiap klaster DB Aurora MySQL di basis data global Aurora, yaitu, berapa banyak data yang mungkin hilang jika ada pemadaman. Seperti halnya lag, RPO diukur dalam waktu.

- LAST_LAG_CALCULATION_TIMESTAMP – Stempel waktu yang menentukan kapan nilai dihitung terakhir kali untuk `DURABILITY_LAG_IN_MILLISECONDS` dan `RPO_LAG_IN_MILLISECONDS`. Nilai waktu seperti `1970-01-01 00:00:00+00` berarti ini adalah klaster DB primer.
- OLDEST_READ_VIEW_TRX_ID – ID transaksi terlama yang dapat dihapus oleh instans DB penulis.

3. Lakukan kueri ke tabel `information_schema.aurora_global_db_instance_status` untuk membuat daftar semua instans DB sekunder baik untuk klaster DB primer maupun klaster DB sekunder.

```
mysql> select * from information_schema.aurora_global_db_instance_status;
```

```
SERVER_ID            |              SESSION_ID              | AWS_REGION
 | DURABLE_LSN | HIGHEST_LSN_RECEIVED | OLDEST_READ_VIEW_TRX_ID |
 OLDEST_READ_VIEW_LSN | VISIBILITY_LAG_IN_MSEC
---------------------+--------------------------------------+------------
+-------------+----------------------+-------------------------
+----------------------+-----------------------
ams-gdb-primary-i2   | MASTER_SESSION_ID                    | us-east-1  |
 183537698   |                    0 |                      0 |
 0 |                      0
ams-gdb-secondary-i1 | cc43165b-bdc6-4651-abbf-4f74f08bf931 | us-west-2  |
 183537689   |            183537692 |                20806928 |
 183537682 |                       0
ams-gdb-secondary-i2 | 53303ff0-70b5-411f-bc86-28d7a53f8c19 | us-west-2  |
 183537689   |            183537692 |                20806928 |
 183537682 |                     677
ams-gdb-primary-i1   | 5af1e20f-43db-421f-9f0d-2b92774c7d02 | us-east-1  |
 183537697   |            183537698 |                20806930 |
 183537691 |                      21
(4 rows)
```

Output mencakup baris untuk setiap instans DB basis data global yang berisi kolom berikut:

- SERVER_ID – Pengidentifikasi server untuk instans DB.
- SESSION_ID – Pengidentifikasi unik untuk sesi saat ini. Nilai `MASTER_SESSION_ID` mengidentifikasi instans DB (primer) Penulis.
- AWS_REGION – Wilayah AWS tempat instans DB ini berada. Untuk tabel yang berisi daftar Wilayah AWS berdasarkan mesin, lihat Wilayah dan Zona Ketersediaan.
- DURABLE_LSN – LSN yang dibuat tahan lama di dalam penyimpanan.
- HIGHEST_LSN_RECEIVED — LSN tertinggi yang diterima oleh instans DB dari instans DB penulis.
- OLDEST_READ_VIEW_TRX_ID – ID transaksi terlama yang dapat dihapus oleh instans DB penulis.
- OLDEST_READ_VIEW_LSN – LSN terlama yang digunakan oleh instans DB untuk membaca dari penyimpanan.
- VISIBILITY_LAG_IN_MSEC – Untuk pembaca di klaster DB primer, seberapa jauh instans DB ini tertinggal di belakang instans DB penulis dalam milidetik. Untuk pembaca di DB klaster sekunder, seberapa jauh DB instans ini tetap tertinggal di belakang volume sekunder dalam milidetik.

Untuk melihat perubahan nilai ini dari waktu ke waktu, pertimbangkan blok transaksi berikut di mana sisipan tabel memakan waktu satu jam.

```
mysql> BEGIN;
mysql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
mysql> COMMIT;
```

Dalam beberapa kasus, mungkin terdapat pemutusan jaringan antara klaster DB primer dan klaster DB sekunder setelah pernyataan `BEGIN`. Jika ya, nilai DURABILITY_LAG_IN_MILLISECONDS klaster DB sekunder mulai meningkat. Pada akhir pernyataan `INSERT`, nilai DURABILITY_LAG_IN_MILLISECONDS adalah 1 jam. Namun, nilai RPO_LAG_IN_MILLISECONDS adalah 0 karena semua data pengguna yang di-commit antara klaster DB primer dan klaster DB sekunder masih sama. Segera setelah pernyataan `COMMIT` selesai, nilai RPO_LAG_IN_MILLISECONDS meningkat.

Memantau basis data global berbasis Aurora PostgreSQL

Untuk melihat status basis data global berbasis Aurora PostgreSQL, gunakan fungsi `aurora_global_db_status` dan `aurora_global_db_instance_status`.

> **Note**
>
> Hanya Aurora PostgreSQL yang mendukung fungsi `aurora_global_db_status` dan `aurora_global_db_instance_status`.

Memantau basis data global Aurora berbasis Aurora PostgreSQL

1. Buat koneksi ke titik akhir klaster primer basis data global yang menggunakan utilitas PostgreSQL seperti psql. Untuk informasi selengkapnya tentang cara membuat koneksi, lihat Terhubung ke basis data global Amazon Aurora.
2. Gunakan fungsi `aurora_global_db_status` dalam perintah psql untuk mencantumkan volume primer dan sekunder. Ini menunjukkan waktu lag klaster DB sekunder global basis data.

```
postgres=> select * from aurora_global_db_status();
```

```
aws_region | highest_lsn_written | durability_lag_in_msec | rpo_lag_in_msec |
 last_lag_calculation_time  | feedback_epoch | feedback_xmin
------------+---------------------+------------------------+-----------------
+----------------------------+----------------+--------------
us-east-1  |         93763984222 |                     -1 |              -1 |
 1970-01-01 00:00:00+00     |              0 |              0
us-west-2  |         93763984222 |                    900 |            1090 |
 2020-05-12 22:49:14.328+00 |              2 |    3315479243
(2 rows)
```

Output mencakup baris untuk setiap klaster DB basis data global yang berisi kolom berikut:

- aws_region – Wilayah AWS tempat klaster DB ini berada. Untuk tabel yang berisi daftar Wilayah AWS berdasarkan mesin, lihat Wilayah dan Zona Ketersediaan.
- highest_lsn_written – Log sequence number (LSN) tertinggi yang saat ini tertulis pada klaster DB ini.

Log sequence number (LSN) adalah nomor urutan unik yang mengidentifikasi catatan di log transaksi basis data. LSN diurutkan sedemikian rupa sehingga LSN yang lebih besar mewakili transaksi berikutnya.

- durability_lag_in_msec – Perbedaan stempel waktu antara log sequence number tertinggi pada klaster DB sekunder (`highest_lsn_written`) dan `highest_lsn_written` pada klaster DB primer.
- rpo_lag_di_msec – Lag sasaran titik pemulihan (RPO). Lag ini adalah perbedaan waktu antara transaksi pengguna terbaru yang disimpan di DB klaster sekunder dan transaksi pengguna terbaru yang disimpan di DB klaster primer.
- last_lag_calculation_time – Stempel waktu ketika nilai terakhir dihitung untuk `durability_lag_in_msec` dan `rpo_lag_in_msec`.
- feedback_epoch – Masa yang digunakan klaster DB sekunder saat menghasilkan informasi hot standby.

  Hot standby adalah ketika klaster DB dapat terhubung dan mengkueri saat server berada dalam mode pemulihan atau mode siaga. Umpan balik hot standby adalah informasi tentang klaster DB saat dalam status hot standby. Untuk informasi selengkapnya, lihat Hot standby di dokumentasi PostgreSQL.
- feedback_xmin – ID transaksi aktif minimum (terlama) yang digunakan oleh klaster DB sekunder.

3. Gunakan fungsi `aurora_global_db_instance_status` untuk membuat daftar semua instans DB sekunder baik untuk klaster DB primer maupun klaster DB sekunder.

```
postgres=> select * from aurora_global_db_instance_status();
```

```
server_id                                    |              session_id
  | aws_region | durable_lsn | highest_lsn_rcvd | feedback_epoch | feedback_xmin |
 oldest_read_view_lsn | visibility_lag_in_msec
----------------------------------------------+-------------------------------------
+------------+-------------+------------------+----------------+---------------
+----------------------+-----------------------
apg-global-db-rpo-mammothrw-elephantro-1-n1 | MASTER_SESSION_ID
 | us-east-1  | 93763985102 |                  |                |               |
                     |
apg-global-db-rpo-mammothrw-elephantro-1-n2 | f38430cf-6576-479a-b296-dc06b1b1964a
 | us-east-1  | 93763985099 |      93763985102 |              2 |    3315479243 |
        93763985095 |                     10
```

```
apg-global-db-rpo-elephantro-mammothrw-n1   | 0d9f1d98-04ad-4aa4-8fdd-e08674cbbbfe
 | us-west-2  | 93763985095 |      93763985099 |              2 |    3315479243 |
        93763985089 |                   1017
(3 rows)
```

Output mencakup baris untuk setiap instans DB basis data global yang berisi kolom berikut:

- server_id – Pengidentifikasi server untuk instans DB.
- session_id – Pengidentifikasi unik untuk sesi saat ini.
- aws_region – Wilayah AWS tempat instans DB ini berada. Untuk tabel yang berisi daftar Wilayah AWS berdasarkan mesin, lihat Wilayah dan Zona Ketersediaan.
- durable_lsn – LSN yang dibuat tahan lama di dalam penyimpanan.
- highest_lsn_received — LSN tertinggi yang diterima oleh instans DB dari instans DB penulis.
- feedback_epoch – Masa yang digunakan instans DB saat menghasilkan informasi hot standby.

  Hot standby adalah ketika instans DB dapat terhubung dan mengkueri saat server berada dalam mode pemulihan atau mode siaga. Umpan balik hot standby adalah informasi tentang instans DB saat dalam status hot standby. Untuk informasi selengkapnya, lihat dokumentasi PostgreSQL tentang Hot standby.
- feedback_xmin – ID transaksi aktif minimum (terlama) yang digunakan oleh instans DB.
- oldest_read_view_lsn – LSN terlama yang digunakan oleh instans DB untuk membaca dari penyimpanan.
- visibility_lag_in_msec – Seberapa jauh instans DB ini tertinggal di belakang instans DB penulis.

Untuk melihat perubahan nilai ini dari waktu ke waktu, pertimbangkan blok transaksi berikut di mana sisipan tabel memakan waktu satu jam.

```
psql> BEGIN;
psql> INSERT INTO table1 SELECT Large_Data_That_Takes_1_Hr_To_Insert;
psql> COMMIT;
```

Dalam beberapa kasus, mungkin terdapat pemutusan jaringan antara klaster DB primer dan klaster DB sekunder setelah pernyataan `BEGIN`. Jika ya, nilai `durability_lag_in_msec` klaster DB sekunder mulai meningkat. Pada akhir pernyataan `INSERT`, nilai `durability_lag_in_msec` adalah 1 jam. Namun, nilai `rpo_lag_in_msec` adalah 0 karena semua data pengguna yang di-

commit antara klaster DB primer dan klaster DB sekunder masih sama. Segera setelah pernyataan `COMMIT` selesai, nilai `rpo_lag_in_msec` akan meningkat.

Menggunakan basis data global Amazon Aurora dengan layanan AWS lainnya

Anda dapat menggunakan basis data global Aurora dengan layanan AWS lain, seperti Amazon S3 dan AWS Lambda. Tindakan tersebut mengharuskan semua klaster DB Aurora dalam basis data global Anda memiliki hak istimewa, fungsi eksternal, dan sebagainya dalam Wilayah AWS masing-masing. Karena klaster DB sekunder Aurora hanya-baca dalam sebuah basis data global Aurora dapat dinaikkan ke peran primer, kami sarankan Anda mengatur hak tulis di awal, di semua klaster DB Aurora untuk layanan apa pun yang ingin Anda gunakan bersama basis data global Aurora Anda.

Prosedur berikut ini berisi rangkuman tindakan yang harus diambil untuk setiap Layanan AWS.

Untuk menginvokasi fungsi AWS Lambda dari basis data global Aurora

1. Untuk semua klaster Aurora yang membentuk basis data global Aurora, lakukan prosedur di Menginvokasi fungsi Lambda dari klaster DB Amazon Aurora MySQL.
2. Untuk setiap klaster di basis data global Aurora, tetapkan (ARN) peran IAM (IAM) yang baru.
3. Untuk mengizinkan pengguna basis data dalam basis data global Aurora untuk menginvokasi fungsi Lambda, kaitkan peran yang Anda buat di Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS dengan setiap klaster di basis data global Aurora.
4. Konfigurasikan setiap klaster dalam basis data global Aurora untuk memungkinkan koneksi keluar ke Lambda. Untuk petunjuk, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora MySQL ke layanan AWS lainnya.

Memuat data dari Amazon S3

1. Untuk semua klaster Aurora yang membentuk basis data global Aurora, lakukan prosedur di Memuat data ke klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3.
2. Untuk setiap klaster Aurora di global basis data, tetapkan parameter klaster DB `aurora_load_from_s3_role` atau `aws_default_s3_role` ke Amazon Resource Name (ARN) dari peran IAM baru. Jika peran IAM tidak ditentukan untuk `aurora_load_from_s3_role`, Aurora menggunakan peran IAM yang ditentukan dalam `aws_default_s3_role`.

3. Untuk mengizinkan pengguna basis data di basis data global Aurora untuk mengakses S3, kaitkan peran yang Anda buat di Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS dengan setiap klaster Aurora dalam global basis data.
4. Konfigurasikan setiap klaster Aurora dalam basis data global untuk mengizinkan koneksi keluar ke S3. Untuk petunjuk, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora MySQL ke layanan AWS lainnya.

Untuk menyimpan data yang dikueri ke Amazon S3

1. Untuk semua klaster Aurora yang membentuk basis data global Aurora, lakukan prosedur di Menyimpan data dari klaster DB Amazon Aurora MySQL ke dalam file teks di bucket Amazon S3.
2. Untuk setiap klaster Aurora di global basis data, tetapkan parameter klaster DB `aurora_select_into_s3_role` atau `aws_default_s3_role` ke Amazon Resource Name (ARN) dari peran IAM baru. Jika peran IAM tidak ditentukan untuk `aurora_select_into_s3_role`, Aurora menggunakan peran IAM yang ditentukan dalam `aws_default_s3_role`.
3. Untuk mengizinkan pengguna basis data di basis data global Aurora untuk mengakses S3, kaitkan peran yang Anda buat di Membuat peran IAM untuk mengizinkan Amazon Aurora mengakses layanan AWS dengan setiap klaster Aurora dalam global basis data.
4. Konfigurasikan setiap klaster Aurora dalam basis data global untuk mengizinkan koneksi keluar ke S3. Untuk petunjuk, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora MySQL ke layanan AWS lainnya.

Meningkatkan basis data global Amazon Aurora

Peningkatan basis data global Aurora mengikuti prosedur yang sama seperti peningkatan klaster DB Aurora. Namun, berikut adalah beberapa perbedaan penting yang perlu diperhatikan sebelum Anda memulai prosesnya.

Kami menyarankan Anda meningkatkan klaster DB primer dan sekunder ke versi yang sama. Anda hanya dapat melakukan failover basis data lintas wilayah terkelola pada basis data global Aurora jika klasrer DB primer dan sekunder memiliki versi mesin tingkat utama, kecil, dan patch yang sama. Namun, tingkat patch bisa berbeda, tergantung versi mesin kecil. Untuk informasi selengkapnya, lihat Kompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola.

Peningkatan versi utama

Saat Anda melakukan peningkatan versi utama basis data global Amazon Aurora, Anda meningkatkan klaster basis data global, bukan klaster individual di dalamnya.

Untuk mempelajari cara meningkatkan basis data global Aurora PostgreSQL ke versi utama yang lebih tinggi, lihat Peningkatan mayor untuk basis data global.

> **Note**
>
> Dengan basis data global Aurora berdasarkan Aurora PostgreSQL, Anda tidak dapat melakukan peningkatan versi utama dari mesin DB Aurora jika fitur sasaran titik pemulihan (RPO) diaktifkan. Untuk informasi tentang fitur RPO, lihat Mengelola RPO untuk basis data global berbasis Aurora PostgreSQL.

Untuk mempelajari cara meningkatkan basis data global Aurora MySQL ke versi utama yang lebih tinggi, lihat Upgrade besar di tempat untuk basis data global.

> **Note**
>
> Dengan basis data global Aurora berdasarkan Aurora MySQL, Anda tidak dapat melakukan peningkatan in-place dari Aurora MySQL versi 2 ke versi 3 jika parameter `lower_case_table_names` diaktifkan.
> Untuk melakukan peningkatan versi utama ke Aurora MySQL versi 3 saat menggunakan `lower_case_table_names`, gunakan proses berikut:
>
> 1. Hapus semua Wilayah sekunder dari klaster global. Ikuti langkah-langkahnya di Menghapus klaster dari basis data global Amazon Aurora.
> 2. Tingkatkan versi mesin dari Wilayah primer ke Aurora MySQL versi 3. Ikuti langkah-langkahnya di Cara melakukan upgrade di tempat.
> 3. Tambahkan Wilayah sekunder ke klaster global. Ikuti langkah-langkahnya di Menambahkan Wilayah AWS ke basis data global Amazon Aurora.
>
> Anda juga dapat menggunakan metode pemulihan snapshot sebagai gantinya. Untuk informasi selengkapnya, lihat Memulihkan dari snapshot klaster DB.

Peningkatan versi kecil.

Untuk peningkatan kecil pada basis data global Aurora, Anda meningkatkan semua klaster sekunder sebelum meningkatkan klaster primer.

Untuk mempelajari cara meningkatkan basis data global Aurora PostgreSQL ke versi kecil yang lebih tinggi, lihat Cara melakukan peningkatan versi minor dan menerapkan patch. Untuk mempelajari cara meningkatkan basis data global Aurora MySQL ke versi kecil yang lebih tinggi, lihat Meningkatkan Aurora MySQL dengan mengubah versi mesin.

Sebelum Anda melakukan peningkatan, tinjau pertimbangan berikut:

- Peningkatan versi kecil klaster sekunder tidak memengaruhi ketersediaan atau penggunaan klaster primer dengan cara apa pun.
- Klaster sekunder harus memiliki setidaknya satu instans DB untuk melakukan peningkatan kecil.
- Jika Anda meningkatkan basis data global Aurora MySQL ke versi 2.11.*, Anda harus meningkatkan klaster DB primer dan sekunder Anda ke versi yang sama persis, termasuk tingkat patch-nya.
- Untuk mendukung switchover atau failover lintas wilayah yang dikelola, Anda harus meningkatkan klaster DB primer dan sekunder Anda ke versi yang sama persis, termasuk tingkat patch, tergantung versi mesin. Untuk informasi selengkapnya, lihat Kompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola.

Kompatibilitas tingkat patch untuk switchover dan failover lintas wilayah yang dikelola

Saat Anda meningkatkan basis data global Aurora Anda ke salah satu versi mesin kecil berikut, Anda dapat melakukan switchover atau failover lintas wilayah terkelola meskipun tingkat patch klaster DB primer dan sekunder Anda tidak cocok. Untuk versi mesin kecil yang lebih rendah daripada yang ada di daftar ini, Anda harus meningkatkan klaster DB primer dan sekunder Anda ke tingkat utama, kecil, dan patch yang sama untuk melakukan switchover atau failover lintas wilayah yang dikelola. Pastikan untuk meninjau informasi versi dan catatan dalam tabel berikut.

> **Note**
>
> Untuk failover lintas Wilayah manual, Anda dapat melakukan proses failover selama klaster DB sekunder target menjalankan versi mesin utama dan kecil yang sama dengan klaster DB primer. Dalam hal ini, tingkat patch tidak perlu cocok.

Mesin basis data	Versi mesin kecil	Catatan
Aurora MySQL	Tidak ada versi minor	Dengan semua versi minor, Anda dapat melakukan switchover atau failover lintas wilayah terkelola hanya jika level patch cluster DB primer dan sekunder cocok.
Aurora PostgreSQL	• Versi 14.5 atau versi minor yang lebih tinggi • Versi 13.8 atau versi minor yang lebih tinggi • Versi 12.12 atau versi minor yang lebih tinggi • Versi 11.17 atau versi kecil yang lebih tinggi	Dengan versi mesin kecil yang tercantum di kolom sebelumnya, Anda dapat melakukan switchover atau failover lintas wilayah terkelola dari klaster DB primer dengan satu tingkat patch ke klaster DB sekunder dengan tingkat patch yang berbeda. Dengan versi minor yang lebih rendah dari ini, Anda dapat melakukan switchover atau failover lintas wilayah terkelola hanya jika level patch cluster DB primer dan sekunder cocok.

Menggunakan Proksi Amazon RDS untuk Aurora

Dengan menggunakan Proksi Amazon RDS, Anda dapat mengizinkan berbagai aplikasi untuk berkumpul dan berbagi koneksi basis data untuk meningkatkan kemampuan penskalaannya. Proksi RDS membuat aplikasi lebih tangguh terhadap kegagalan basis data dengan secara otomatis menghubungkan ke sebuah instans DB siaga sekaligus menjaga koneksi aplikasi. Dengan menggunakan RDS Proxy, Anda juga dapat menerapkan autentikasi AWS Identity and Access Management (IAM) untuk database, dan menyimpan kredensil dengan aman. AWS Secrets Manager

Dengan Proksi RDS, Anda dapat menangani lonjakan yang tidak dapat diprediksi dalam lalu lintas basis data. Jika tidak, lonjakan ini dapat menyebabkan masalah karena permintaan koneksi berlebihan atau koneksi baru yang dibuat sangatlah tinggi. Proksi RDS membangun kumpulan koneksi basis data dan menggunakan ulang koneksi dalam kumpulan ini. Pendekatan ini menghindari overhead memori dan CPU dari membuka koneksi basis data baru. Untuk melindungi basis data dari permintaan berlebihan, Anda dapat mengontrol jumlah koneksi basis data yang dibuat.

Proksi RDS mengantrekan atau membatasi koneksi aplikasi yang tidak dapat dilayani segera dari kumpulan koneksi. Meskipun latensi dapat meningkat, aplikasi Anda dapat terus diskalakan tanpa kegagalan mendadak atau membanjiri basis data. Jika permintaan koneksi melebihi batas yang Anda tentukan, Proksi RDS akan menolak koneksi aplikasi (yakni melepaskan beban). Selain itu, Proksi RDS akan mempertahankan performa yang dapat diprediksi untuk beban yang dapat dilayani RDS dengan kapasitas yang tersedia.

Anda dapat mengurangi overhead untuk memproses kredensial dan membangun koneksi yang aman untuk setiap koneksi baru. Proksi RDS dapat menangani beberapa dari pekerjaan itu untuk mewakili basis data.

Proksi RDS sepenuhnya kompatibel dengan versi mesin yang didukungnya. Anda dapat mengaktifkan Proksi RDS pada sebagian besar aplikasi tanpa perubahan kode. Untuk daftar versi mesin yang didukung, lihat Proksi Amazon RDS.

Topik

- Ketersediaan wilayah dan versi
- Kuota dan Pembatasan untuk Proksi RDS
- Merencanakan lokasi penggunaan Proksi RDS
- Konsep dan terminologi Proksi RDS

- Memulai dengan Proksi RDS
- Mengelola Proksi RDS
- Bekerja dengan titik akhir Proksi Amazon RDS
- Memantau metrik Proxy RDS dengan Amazon CloudWatch
- Bekerja dengan peristiwa Proksi RDS
- Contoh baris perintah Proksi RDS
- Pemecahan masalah untuk Proksi RDS
- Penggunaan Proksi RDS dengan AWS CloudFormation
- Menggunakan Proksi RDS dengan basis data global Aurora

Ketersediaan wilayah dan versi

Untuk informasi tentang dukungan versi mesin basis data dan ketersediaan Proxy RDS dalam bentuk tertentu Wilayah AWS, lihatProksi Amazon RDS.

Kuota dan Pembatasan untuk Proksi RDS

Kuota dan batasan berikut berlaku untuk Proksi RDS:

- Anda dapat memiliki hingga 20 proxy untuk setiap ID AWS akun. Jika aplikasi Anda memerlukan lebih banyak proxy, Anda dapat meminta proxy tambahan dengan membuka tiket dengan organisasi Support. AWS
- Setiap proksi dapat memiliki hingga 200 rahasia Secrets Manager terkait. Sehingga, setiap proksi dapat terhubung ke hingga 200 akun pengguna yang berbeda pada waktu tertentu.
- Setiap proksi memiliki titik akhir default. Anda juga dapat menambahkan hingga 20 titik akhir proksi untuk setiap proksi. Anda dapat membuat, melihat, mengubah, dan menghapus titik akhir ini.
- Dalam sebuah klaster Aurora, semua koneksi yang menggunakan titik akhir proksi default ditangani oleh instans penulis Aurora. Untuk melakukan penyeimbangan beban untuk beban kerja padat baca, Anda dapat membuat titik akhir hanya-baca untuk proksi. Titik akhir tersebut meneruskan koneksi ke titik akhir pembaca klaster. Dengan begitu, koneksi proksi Anda dapat memanfaatkan skalabilitas baca Aurora. Untuk informasi selengkapnya, lihat Ikhtisar titik akhir proksi.
- Anda dapat menggunakan Proksi RDS dengan klaster Aurora Serverless v2, tetapi tidak dengan klaster Aurora Serverless v1.

- Proksi RDS Anda harus berada dalam cloud privat virtual (VPC) yang sama seperti basis data. Proksi tersebut tidak dapat diakses publik, meskipun basis datanya dapat diakses publik. Misalnya, jika membuat prototipe basis data di host lokal, Anda tidak dapat terhubung ke proksi kecuali Anda menyiapkan persyaratan jaringan yang diperlukan untuk mengizinkan koneksi ke proksi. Ini karena host lokal Anda berada di luar VPC proksi.

  > **Note**
  >
  > Untuk klaster DB Aurora, Anda dapat mengaktifkan akses lintas-VPC. Untuk melakukannya, buat titik akhir tambahan untuk proksi dan tentukan VPC, subnet, dan grup keamanan yang berbeda dengan titik akhir tersebut. Untuk informasi selengkapnya, lihat Mengakses basis data Aurora di seluruh VPC.

- Anda tidak dapat menggunakan Proksi RDS dengan VPC yang penghuniannya diatur ke `dedicated`.
- Jika Anda menggunakan Proksi RDS dengan klaster DB Aurora yang autentikasi IAM-nya diaktifkan, periksa autentikasi pengguna. Pengguna yang terhubung melalui proksi harus melakukan autentikasi melalui kredensial masuk. Untuk detail tentang Secrets Manager dan dukungan IAM di Proksi RDS, lihat Menyiapkan kredensi database di AWS Secrets Manager dan Menyiapkan AWS Identity and Access Management kebijakan (IAM).
- Anda tidak dapat menggunakan Proksi RDS dengan DNS kustom saat menggunakan validasi nama host SSL.
- Setiap proksi dapat dikaitkan dengan satu instans DB target. Namun, Anda dapat mengaitkan beberapa proksi dengan instans DB yang sama.
- Pernyataan apa pun dengan teks berukuran lebih dari 16 KB menyebabkan proksi menyematkan sesi ke koneksi saat ini.
- Wilayah tertentu memiliki batasan Zona Ketersediaan (AZ) untuk dipertimbangkan saat membuat proksi. Wilayah AS Timur (Virginia Utara) tidak mendukung Proksi RDS di Zona Ketersediaan `use1-az3`. Wilayah AS Barat (California Utara) tidak mendukung Proksi RDS di Zona Ketersediaan `usw1-az2`. Saat memilih subnet sekaligus membuat proksi, pastikan Anda tidak memilih subnet di Zona Ketersediaan yang disebutkan di atas.

Untuk batasan tambahan untuk setiap mesin DB, lihat bagian berikut:

- Batasan tambahan untuk Aurora MySQL
- Batasan tambahan untuk Aurora PostgreSQL

Batasan tambahan untuk Aurora MySQL

Batasan tambahan berikut berlaku untuk Proksi RDS dengan basis data Aurora MySQL:

- Proksi RDS tidak mendukung plugin autentikasi `sha256_password` dan `caching_sha2_password` MySQL. Plugin ini menerapkan hashing SHA-256 untuk kata sandi akun pengguna.
- Saat ini, semua proksi mendengarkan di port 3306 untuk MySQL. Proksi ini masih terhubung ke basis data Anda menggunakan port yang sudah ditentukan dalam pengaturan basis data.
- Anda tidak dapat menggunakan Proksi RDS dengan basis data MySQL yang dikelola sendiri dalam instans EC2.
- Anda tidak dapat menggunakan Proksi RDS dengan instans DB RDS for MySQL dengan parameter `read_only` dalam grup parameter DB-nya diatur ke 1.
- Proksi RDS tidak mendukung mode terkompresi MySQL. Misalnya, Proksi RDS tidak mendukung kompresi yang digunakan oleh opsi `--compress` atau `-C` perintah `mysql`.
- Koneksi basis data yang memproses perintah `GET DIAGNOSTIC` mungkin menampilkan informasi yang tidak akurat saat Proksi RDS menggunakan kembali koneksi basis data yang sama untuk menjalankan kueri lain. Hal ini bisa terjadi ketika Proksi RDS me-multipleks koneksi basis data.
- Beberapa pernyataan dan fungsi SQL seperti `SET LOCAL` dapat mengubah status koneksi tanpa menyebabkan penyematan. Untuk perilaku penyematan terbaru, lihat Menghindari penyematan.

> **Important**
>
> Untuk proksi yang terkait dengan basis data MySQL, jangan atur parameter konfigurasi `sql_auto_is_null` ke `true` atau nilai bukan nol dalam kueri inisialisasi. Tindakan ini bisa menyebabkan perilaku aplikasi yang salah.

Batasan tambahan untuk Aurora PostgreSQL

Batasan tambahan berikut berlaku untuk Proksi RDS dengan basis data Aurora PostgreSQL:

- Proksi RDS tidak mendukung filter penyematan sesi untuk PostgreSQL.
- Saat ini, semua proksi mendengarkan di port 5432 untuk PostgreSQL.

- Untuk PostgreSQL, Proksi RDS saat ini tidak mendukung pembatalan kueri dari klien dengan mengeluarkan `CancelRequest`. Misalnya, hal ini bisa terjadi ketika Anda membatalkan kueri yang berjalan lama dalam sesi psql interaktif dengan menggunakan Ctrl+C.
- Hasil dari fungsi lastval PostgreSQL tidak selalu akurat. Sebagai solusi, gunakan pernyataan INSERT dengan klausul `RETURNING`.
- Proksi RDS saat ini tidak mendukung mode replikasi streaming.

> **Important**
>
> Untuk proksi yang ada dengan basis data PostgreSQL, jika Anda mengubah autentikasi basis data untuk menggunakan `SCRAM` saja, proksi akan menjadi tidak tersedia selama maksimal 60 detik. Untuk menghindari masalah ini, lakukan salah satu hal berikut:
>
> - Pastikan basis data memungkinkan autentikasi `SCRAM` dan `MD5`.
> - Untuk hanya menggunakan autentikasi `SCRAM`, buat proksi baru, migrasi lalu lintas aplikasi Anda ke proksi baru, lalu hapus proksi yang sebelumnya terkait dengan basis data.

Merencanakan lokasi penggunaan Proksi RDS

Anda dapat menentukan instans, klaster, dan aplikasi DB mana yang mungkin paling banyak mendapatkan manfaat dari penggunaan Proksi RDS. Untuk melakukannya, pertimbangkan faktor-faktor berikut:

- klaster DB apa pun yang mengalami kesalahan "terlalu banyak koneksi" adalah kandidat yang baik untuk dikaitkan dengan proksi. Ini sering ditandai dengan nilai `ConnectionAttempts` CloudWatch metrik yang tinggi. Proksi tersebut memungkinkan aplikasi untuk membuka banyak koneksi klien sekaligus mengelola koneksi jangka panjang dalam jumlah lebih kecil ke instans DB.
- Untuk cluster DB yang menggunakan kelas AWS instans yang lebih kecil, seperti T2 atau T3, menggunakan proxy dapat membantu menghindari kondisi. out-of-memory Tindakan ini juga dapat membantu mengurangi overhead CPU untuk membangun koneksi. Kondisi ini dapat terjadi saat berurusan dengan koneksi dalam jumlah besar.
- Anda dapat memantau CloudWatch metrik Amazon tertentu untuk menentukan apakah cluster DB mendekati jenis batas tertentu. Batasan ini ditujukan untuk jumlah koneksi dan memori terkait dengan pengelolaan koneksi. Anda juga dapat memantau CloudWatch metrik tertentu untuk menentukan apakah cluster DB menangani banyak koneksi berumur pendek. Pembukaan dan

penutupan koneksi tersebut dapat membebani overhead performa pada basis data Anda. Untuk informasi tentang metrik yang akan dipantau, lihat Memantau metrik Proxy RDS dengan Amazon CloudWatch.

- Fungsi AWS Lambda juga bisa menjadi kandidat yang tepat untuk penggunaan proksi. Fungsi ini sering membuat koneksi basis data pendek yang mendapatkan manfaat dari kumpulan koneksi yang ditawarkan oleh Proksi RDS. Anda dapat memanfaatkan autentikasi IAM apa pun yang Anda miliki untuk fungsi Lambda, alih-alih mengelola kredensial basis data dalam kode aplikasi Lambda.
- Aplikasi yang biasanya membuka dan menutup koneksi basis data dalam jumlah besar dan tidak memiliki mekanisme pengumpulan koneksi bawaan adalah kandidat yang tepat untuk menggunakan proksi.
- Aplikasi yang dapat mempertahankan koneksi terbuka dalam jumlah besar selama jangka waktu yang lama biasanya merupakan kandidat yang tepat untuk penggunaan proksi. Aplikasi dalam industri seperti Perangkat Lunak sebagai Layanan (SaaS) atau e-niaga sering kali meminimalkan latensi untuk permintaan basis data dengan membiarkan koneksi terbuka. Dengan Proksi RDS, aplikasi dapat mempertahankan lebih banyak koneksi tetap terbuka daripada saat terhubung langsung ke instans DB.
- Mungkin Anda belum mengadopsi autentikasi IAM dan Secrets Manager karena kompleksitas penyiapan autentikasi tersebut untuk semua instans DB. Jika demikian, Anda dapat terus menerapkan metode autentikasi yang sudah ada dan mendelegasikan autentikasi ke proksi. Proksi dapat menerapkan kebijakan autentikasi koneksi klien untuk aplikasi tertentu. Anda dapat memanfaatkan autentikasi IAM apa pun yang Anda miliki untuk fungsi Lambda, alih-alih mengelola kredensial basis data dalam kode aplikasi Lambda.
- Proksi RDS dapat membantu membuat aplikasi lebih tangguh dan transparan terhadap kegagalan basis data. Proksi RDS melewati cache Sistem Nama Domain (DNS) untuk mengurangi waktu failover hingga 66% untuk instans DB basis data Aurora Multi-AZ. Proksi RDS juga secara otomatis merutekan lalu lintas ke instans basis data baru sekaligus mempertahankan koneksi aplikasi. Hal ini membuat failover lebih transparan untuk aplikasi.

Konsep dan terminologi Proksi RDS

Anda dapat menyederhanakan manajemen koneksi untuk klaster DB Amazon Aurora dengan menggunakan Proksi RDS.

Proksi RDS menangani lalu lintas jaringan antara aplikasi klien dan basis data. Tindakan ini dilakukan secara aktif terlebih dahulu dengan memahami protokol basis data. Lalu perilakunya disesuaikan berdasarkan operasi SQL dari aplikasi Anda dan serangkaian hasil dari basis data.

Proksi RDS mengurangi overhead memori dan CPU untuk manajemen koneksi pada basis data Anda. basis data membutuhkan lebih sedikit sumber daya memori dan CPU saat aplikasi membuka banyak koneksi secara bersamaan. Logika juga tidak dibutuhkan dalam aplikasi Anda untuk menutup dan membuka kembali koneksi yang idle dalam waktu yang lama. Demikian pula, logika aplikasi yang dibutuhkan untuk membangun kembali koneksi juga lebih sedikit jika terjadi masalah pada basis data.

Infrastruktur untuk Proksi RDS memiliki ketersediaan tinggi dan di-deploy pada berbagai Zona Ketersediaan (AZ). Komputasi, memori, dan penyimpanan untuk Proksi RDS tidak bergantung pada klaster DB Aurora. Independensi ini membantu menurunkan overhead pada server basis data Anda, sehingga dapat mencurahkan sumber dayanya untuk melayani beban kerja basis data. Sumber daya komputasi Proksi RDS bersifat nirserver, yang diskalakan secara otomatis berdasarkan beban kerja basis data Anda.

Topik

- Ikhtisar konsep Proksi RDS
- Pengumpulan koneksi
- Keamanan Proksi RDS
- Failover
- Transaksi

Ikhtisar konsep Proksi RDS

Proksi RDS menangani infrastruktur untuk melakukan pengumpulan koneksi dan fitur lain yang dijelaskan di bagian berikutnya. Anda melihat proksi ditampilkan dalam konsol RDS pada halaman Proksi.

Setiap proksi menangani koneksi ke klaster DB Aurora tunggal. Proksi tersebut secara otomatis menentukan instans penulis saat ini untuk klaster yang disediakan Aurora.

Koneksi yang dibiarkan terbuka dan disediakan oleh proksi untuk digunakan oleh aplikasi basis data Anda akan membentuk kumpulan koneksi.

Secara default, Proksi RDS dapat menggunakan ulang koneksi setelah setiap transaksi dalam sesi Anda. Penggunaan kembali tingkat transaksi ini disebut multiplexing. Jika Proksi RDS secara temporer menghapus satu koneksi dari kumpulan koneksi untuk menggunakannya kembali, operasi tersebut disebut borrowing koneksi. Jika aman dilakukan, Proksi RDS akan mengembalikan koneksi tersebut ke kumpulan koneksi.

Dalam beberapa kasus, Proksi RDS tidak dapat memastikan keamanan penggunaan ulang sebuah koneksi basis data di luar sesi saat ini. Untuk kasus seperti ini, Proksi RDS akan tetap mempertahankan sesi pada koneksi yang sama hingga sesi berakhir. Perilaku fallback ini disebut pinning.

Proksi memiliki titik akhir default. Anda terhubung ke titik akhir ini saat menggunakan klaster DB Amazon Aurora. Anda melakukannya alih-alih terhubung ke titik akhir baca/tulis yang terhubung langsung ke instans. Titik akhir tujuan khusus untuk klaster Aurora tetap tersedia untuk Anda gunakan. Untuk klaster DB Aurora, Anda juga dapat membuat titik akhir baca/tulis dan hanya-baca tambahan. Untuk informasi selengkapnya, lihat Ikhtisar titik akhir proksi.

Misalnya, Anda masih dapat terhubung ke titik akhir klaster untuk koneksi baca/tulis tanpa pengumpulan koneksi. Anda masih dapat terhubung ke titik akhir pembaca untuk koneksi keseimbangan beban hanya-baca. Anda masih dapat terhubung ke titik akhir instans untuk melakukan diagnosis dan memecahkan masalah instans DB tertentu dengan klaster. Jika Anda menggunakan layanan AWS lain seperti AWS Lambda untuk terhubung ke basis data RDS, ubah pengaturan koneksinya untuk menggunakan titik akhir proksi. Misalnya, tentukan titik akhir proksi untuk mengizinkan fungsi Lambda mengakses basis data Anda sekaligus memanfaatkan fungsionalitas Proksi RDS.

Setiap proksi berisi sebuah grup target. Grup target ini berisi klaster DB Aurora yang menjadi tujuan koneksi proksi. Untuk klaster Aurora, grup target dikaitkan secara default dengan semua instans DB dalam klaster tersebut. Dengan demikian, proksi dapat terhubung ke instans DB Aurora mana pun yang dipromosikan menjadi instans penulis dalam klaster. klaster DB Aurora yang terkait dengan proksi disebut sebagai target proksi tersebut. Untuk kemudahan, saat Anda membuat proksi melalui konsol, Proksi RDS juga membuat grup target yang sesuai dan mendaftarkan target terkait ini secara otomatis.

Keluarga mesin adalah serangkaian mesin basis data terkait yang menggunakan protokol DB yang sama. Anda dapat memilih keluarga mesin untuk setiap proksi yang Anda buat.

Pengumpulan koneksi

Setiap proksi melakukan pengumpulan koneksi untuk instans penulis dari DB Aurora terkaitnya. Pengumpulan koneksi adalah pengoptimalan yang menurunkan overhead yang terkait dengan pembukaan dan penutupan koneksi dan dengan menjaga banyak koneksi terbuka secara bersamaan. Overhead ini mencakup memori yang diperlukan untuk menangani setiap koneksi baru. Ini juga melibatkan overhead CPU untuk menutup setiap koneksi dan membuka koneksi yang baru.

Contohnya meliputi jabat tangan Keamanan Lapisan Pengangkutan/Lapisan Soket Aman (TLS/SSL), autentikasi, kemampuan negosiasi, dan sebagainya. Pengumpulan koneksi menyederhanakan logika aplikasi Anda. Anda tidak perlu menulis kode aplikasi untuk meminimalkan jumlah koneksi terbuka secara bersamaan.

Setiap proksi juga melakukan multipleks koneksi, yang juga dikenal sebagai penggunaan ulang koneksi. Dengan multiplexing, Proksi RDS melakukan semua operasi untuk transaksi menggunakan satu koneksi basis data acuan. RDS kemudian dapat menggunakan koneksi yang berbeda untuk transaksi berikutnya. Anda dapat membuka banyak koneksi ke proksi secara bersamaan, dan proksi akan mempertahankan koneksi terbuka dalam jumlah yang lebih kecil ke instans atau klaster DB. Tindakan ini akan lebih meminimalkan overhead memori untuk koneksi di server basis data. Teknik ini juga mengurangi kemungkinan kesalahan "terlalu banyak koneksi".

Keamanan Proksi RDS

Proksi RDS menggunakan mekanisme keamanan RDS yang sudah ada seperti TLS/SSL dan AWS Identity and Access Management (IAM). Untuk informasi umum tentang fitur keamanan tersebut, lihat Keamanan dalam Amazon Aurora. Selain itu, pastikan Anda benar-benar mengetahui bagaimana cara kerja Aurora dengan autentikasi, otorisasi, dan bidang keamanan lainnya.

Proksi RDS dapat bertindak sebagai lapisan keamanan tambahan antara aplikasi klien dan basis data acuan. Misalnya, Anda dapat terhubung ke proksi menggunakan TLS 1.2, meskipun instans DB acuan mendukung TLS versi yang lebih lama. Anda dapat terhubung ke proksi menggunakan peran IAM. Hal ini terjadi bahkan jika proksi terhubung ke basis data menggunakan pengguna native dan metode autentikasi kata sandi. Dengan menggunakan teknik ini, Anda dapat menerapkan persyaratan autentikasi yang kuat untuk aplikasi basis data tanpa sebuah upaya migrasi yang mahal untuk instans DB itu sendiri.

Anda menyimpan kredensial basis data yang digunakan oleh Proksi RDS dalam AWS Secrets Manager. Setiap pengguna basis data untuk sebuah klaster DB Aurora yang diakses oleh proksi harus memiliki rahasia yang sesuai di Secrets Manager. Anda juga dapat menyiapkan autentikasi IAM untuk pengguna Proksi RDS. Dengan melakukannya, Anda dapat menerapkan autentikasi IAM untuk akses basis data meskipun basis data menggunakan autentikasi kata sandi native. Sebaiknya gunakan fitur keamanan ini, alih-alih menyematkan kredensial basis data dalam kode aplikasi Anda.

Penggunaan TLS/SSL dengan Proksi RDS

Anda dapat terhubung ke Proksi RDS menggunakan protokol TLS/SSL.

> **Note**
>
> Proksi RDS menggunakan sertifikat dari AWS Certificate Manager (ACM). Jika Anda menggunakan Proksi RDS, Anda tidak perlu mengunduh sertifikat Amazon RDS atau memperbarui aplikasi yang menggunakan koneksi Proksi RDS.

Untuk menerapkan TLS untuk semua koneksi antara proksi dan basis data, Anda dapat menentukan pengaturan Wajibkan Keamanan Lapisan Pengangkutan saat membuat atau mengubah sebuah AWS Management Console.

Proksi RDS juga dapat memastikan agar sesi Anda menggunakan TLS/SSL antara klien Anda dan titik akhir Proksi RDS. Untuk membuat Proksi RDS melakukannya, tentukan persyaratan pada sisi klien. Variabel sesi SSL tidak diatur untuk koneksi SSL ke basis data menggunakan Proksi RDS.

- Untuk Aurora MySQL, tentukan persyaratan pada sisi klien dengan parameter `--ssl-mode` saat Anda menjalankan perintah `mysql`.
- Untuk dan Aurora PostgreSQL, tentukan `sslmode=require` sebagai bagian dari string `conninfo` saat Anda menjalankan perintah `psql`.

Proksi RDS mendukung protokol TLS versi 1.0, 1.1, dan 1.2. Anda dapat terhubung ke proksi menggunakan versi TLS yang lebih tinggi daripada yang digunakan dalam basis data acuan.

Secara default, program klien membangun koneksi terenkripsi dengan Proksi RDS, dengan ketersediaan kontrol lebih lanjut melalui opsi `--ssl-mode`. Dari sisi klien, Proksi RDS mendukung semua mode SSL.

Untuk klien, mode SSL adalah sebagai berikut:

DIUTAMAKAN

SSL adalah pilihan pertama, tetapi tidak diharuskan.

NONAKTIFKAN

Tidak ada SSL yang diperbolehkan.

DIPERLUKAN

Menerapkan SSL.

VERIFY_CA

Menerapkan SSL dan memverifikasi otoritas sertifikat (CA).

VERIFY_IDENTITY

Menerapkan SSL dan memverifikasi CA dan nama host CA.

Saat menggunakan sebuah klien dengan `--ssl-mode` `VERIFY_CA` atau `VERIFY_IDENTITY`, tentukan opsi `--ssl-ca` yang menunjuk ke CA dalam format `.pem`. Untuk file `.pem` yang akan digunakan, unduh semua CA PEM root dari Layanan Kepercayaan Amazon dan tempatkan ke dalam satu file `.pem`.

Proksi RDS menggunakan sertifikat wildcard, yang berlaku untuk domain dan subdomainnya. Jika Anda menggunakan klien `mysql` untuk terhubung dengan mode SSL `VERIFY_IDENTITY`, Anda kini harus menggunakan perintah `mysql` yang kompatibel dengan MySQL 8.0.

Failover

Failover adalah fitur ketersediaan tinggi yang menggantikan instans basis data dengan yang lain saat instans asli tidak tersedia. Failover dapat terjadi karena sebuah masalah pada instans basis data. Bisa juga bagian dari prosedur pemeliharaan normal, seperti saat upgrade basis data. Failover berlaku untuk klaster DB Aurora dengan satu instans pembaca atau lebih selain instans penulis.

Terhubung melalui proksi membuat aplikasi Anda lebih tangguh menghadapi failover basis data. Saat instans DB asli tidak tersedia, Proksi RDS akan terhubung ke basis data siaga tanpa kehilangan koneksi aplikasi yang idle. Hal ini dapat membantu mempercepat dan menyederhanakan proses failover. Dampaknya terhadap aplikasi juga lebih kecil dibandingkan masalah reboot atau basis data pada umumnya.

Tanpa Proksi RDS, failover dapat menyebabkan pemadaman singkat. Selama pemadaman, Anda tidak dapat melakukan operasi tulis pada basis data dengan failover. Koneksi semua basis data yang ada akan terganggu, dan aplikasi Anda harus membuka ulang koneksi tersebut. basis data akan tersedia untuk koneksi dan operasi tulis baru saat instans DB hanya-baca dipromosikan menggantikan yang tidak tersedia.

Selama failover DB, Proksi RDS terus menerima koneksi di alamat IP yang sama dan secara otomatis mengarahkan koneksi ke instans DB primer baru. Klien yang terhubung melalui Proksi RDS tidak rentan terhadap hal berikut:

- Penundaan propagasi Sistem Nama Domain (DNS) saat failover.
- Caching DNS lokal.
- Waktu koneksi habis.
- Ketidakpastian tentang instans DB mana yang merupakan instans penulis saat ini.
- Menunggu respons kueri dari penulis sebelumnya yang menjadi tidak tersedia tanpa menutup koneksi.

Untuk aplikasi yang mempertahankan kumpulan koneksinya sendiri, melewati Proksi RDS berarti sebagian besar koneksi tetap aktif selama failover atau gangguan lainnya. Hanya koneksi yang berada di tengah transaksi atau pernyataan SQL yang dibatalkan. Proksi RDS segera menerima koneksi baru. Saat penulis basis data tidak tersedia, Proksi RDS akan mengantrekan permintaan masuk.

Untuk aplikasi yang tidak mempertahankan kumpulan koneksinya sendiri, Proksi RDS menawarkan tingkat koneksi yang lebih cepat dan lebih banyak koneksi terbuka. Hal ini menyebabkan overhead yang mahal dikarenakan seringnya koneksi ulang dari basis data. Hal ini dilakukan dengan menggunakan kembali koneksi basis data yang dipertahankan dalam kumpulan koneksi Proksi RDS. Pendekatan ini sangatlah penting untuk koneksi TLS, yang memerlukan biaya penyiapan yang sangat besar.

Transaksi

Semua pernyataan dalam satu transaksi selalu menggunakan koneksi basis data acuan yang sama. Koneksi akan dapat digunakan oleh sesi yang berbeda saat transaksi berakhir. Penggunaan transaksi sebagai unit granularitas memiliki konsekuensi sebagai berikut:

- Penggunaan ulang koneksi bisa terjadi setelah setiap pernyataan jika pengaturan `autocommit` Aurora MySQL diaktifkan.
- Sebaliknya, jika pengaturan `autocommit` dinonaktifkan, pernyataan pertama yang Anda terbitkan dalam sesi akan memulai transaksi baru. Misalnya, Anda memasukkan urutan pernyataan `SELECT`, `INSERT`, `UPDATE`, dan bahasa manipulasi data (DML) lainnya. Dalam hal ini, penggunaan kembali koneksi tidak terjadi hingga Anda mengeluarkan `COMMIT`, `ROLLBACK`, atau mengakhiri transaksi.
- Memasukkan pernyataan bahasa definisi data (DDL) akan menyebabkan transaksi berakhir setelah pernyataan tersebut selesai.

Proksi RDS mendeteksi waktu saat transaksi berakhir melalui protokol jaringan yang digunakan oleh aplikasi klien basis data. Deteksi transaksi tidak bergantung pada kata kunci seperti `COMMIT` atau `ROLLBACK` yang muncul dalam teks pernyataan SQL.

Dalam beberapa kasus, Proksi RDS dapat mendeteksi permintaan basis data yang membuatnya tidak praktis untuk memindahkan sesi Anda ke koneksi yang berbeda. Dalam kasus ini, multiplexing dinonaktifkan untuk koneksi tersebut hingga sesi Anda berakhir. Aturan serupa berlaku jika Proksi RDS tidak dapat memastikan apakah multiplexing praktis untuk sesi ini. Operasi ini disebut pinning. Untuk mendeteksi dan meminimalkan pinning, lihat Menghindari penyematan.

Memulai dengan Proksi RDS

Di bagian berikut ini, Anda dapat menemukan cara menyiapkan dan mengelola Proksi RDS. Anda juga dapat menemukan cara mengatur opsi keamanan terkait. Opsi ini mengontrol siapa saja yang dapat mengakses setiap proksi dan cara setiap proksi terhubung ke instans DB.

Topik

- Menyiapkan prasyarat jaringan
- Menyiapkan kredensi database di AWS Secrets Manager
- Menyiapkan AWS Identity and Access Management kebijakan (IAM)
- Membuat Proksi RDS
- Melihat Proksi RDS
- Terhubung ke basis data melalui Proksi RDS

Menyiapkan prasyarat jaringan

Menggunakan Proksi RDS mengharuskan Anda memiliki cloud privat virtual (VPC) umum antara klaster DB Aurora dan Proksi RDS. VPC ini harus memiliki minimal dua subnet yang berada di Zona Ketersediaan yang berbeda. Akun Anda dapat memiliki subnet ini atau membagikannya dengan akun lain. Untuk informasi tentang berbagi VPC, lihat Bekerja dengan VPC bersama.

Sumber daya aplikasi klien seperti Amazon EC2, Lambda, atau Amazon ECS bisa berada di VPC yang sama dengan proksi. Atau sumber daya ini bisa berada di VPC terpisah dari proksi. Jika Anda berhasil terhubung ke klaster DB Aurora apa pun, berarti Anda sudah memiliki sumber daya jaringan yang diperlukan.

Topik

- Mendapatkan informasi tentang subnet Anda
- Perencanaan untuk kapasitas alamat IP

Mendapatkan informasi tentang subnet Anda

Jika Anda baru memulai dengan Aurora, Anda dapat mempelajari dasar-dasar menghubungkan ke database dengan mengikuti prosedur di. Menyiapkan lingkungan Anda untuk Amazon Aurora Anda juga dapat mengikuti tutorial dalam Memulai dengan Amazon Aurora.

Untuk membuat proxy, Anda harus menyediakan subnet dan VPC tempat proxy beroperasi di dalamnya. Contoh Linux berikut menunjukkan AWS CLI perintah yang memeriksa VPC dan subnet yang dimiliki oleh Anda. Akun AWS Khususnya, Anda meneruskan ID subnet sebagai parameter ketika Anda membuat proksi menggunakan CLI.

```
aws ec2 describe-vpcs
aws ec2 describe-internet-gateways
aws ec2 describe-subnets --query '*[].[VpcId,SubnetId]' --output text | sort
```

Contoh Linux berikut menunjukkan AWS CLI perintah untuk menentukan ID subnet yang sesuai dengan instance tertentu.

Untuk klaster Aurora, cari ID untuk salah satu instans DB terkait terlebih dahulu. Anda dapat mengekstrak ID subnet yang digunakan oleh instans DB tersebut. Untuk melakukannya, periksa kolom bertingkat dalam atribut `DBSubnetGroup` dan `Subnets` dan di output deskripsi untuk instans DB. Anda menentukan beberapa atau semua ID subnet tersebut saat menyiapkan proksi untuk server basis data tersebut.

```
$ # Find the ID of any DB instance in the cluster.
$  aws rds describe-db-clusters --db-cluster-identifier my_cluster_id --query '*[].
[DBClusterMembers]||[0]||[0][*].DBInstanceIdentifier' --output text
```

```
my_instance_id
instance_id_2
instance_id_3
```

Setelah menemukan ID instans DB, periksa VPC terkait untuk menemukan subnetnya. Contoh Linux berikut menunjukkan caranya.

```
$ #From the DB instance, trace through the DBSubnetGroup and Subnets to find the subnet
 IDs.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]|[0]|[0]|[Subnets]|[0]|[*].SubnetIdentifier' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
...
```

```
$ #From the DB instance, find the VPC.
$ aws rds describe-db-instances --db-instance-identifier my_instance_id --query '*[].
[DBSubnetGroup]|[0]|[0].VpcId' --output text
```

```
my_vpc_id
```

```
$ aws ec2 describe-subnets --filters Name=vpc-id,Values=my_vpc_id --query '*[].
[SubnetId]' --output text
```

```
subnet_id_1
subnet_id_2
subnet_id_3
subnet_id_4
subnet_id_5
subnet_id_6
```

Perencanaan untuk kapasitas alamat IP

Proksi RDS secara otomatis menyesuaikan kapasitasnya sesuai kebutuhan berdasarkan ukuran dan jumlah instans DB yang didaftarkan di proksi. Operasi tertentu mungkin juga memerlukan kapasitas proksi tambahan seperti menambah ukuran basis data yang terdaftar atau operasi pemeliharaan Proksi RDS internal. Selama operasi ini, proksi Anda mungkin memerlukan lebih banyak alamat IP untuk menyediakan kapasitas tambahan. Alamat tambahan ini memungkinkan proksi Anda diskalakan tanpa memengaruhi beban kerja Anda. Kurangnya alamat IP kosong di subnet Anda mencegah proksi dari menaikkan skala. Hal ini dapat menyebabkan latensi kueri yang lebih tinggi atau kegagalan koneksi klien. RDS memberi tahu Anda melalui peristiwa `RDS-EVENT-0243` ketika tidak ada cukup alamat IP kosong di subnet Anda. Untuk informasi tentang peristiwa ini, lihat Bekerja dengan peristiwa Proksi RDS.

Berikut adalah jumlah minimum alamat IP yang disarankan untuk dibiarkan kosong di subnet Anda untuk proksi Anda berdasarkan ukuran kelas instans DB.

Kelas instans DB	Alamat IP kosong minimum
db.*.xlarge atau lebih kecil	10
db.*.2xlarge	15
db.*.4xlarge	25
db.*.8xlarge	45
db.*.12xlarge	60
db.*.16xlarge	75
db.*.24xlarge	110

Jumlah alamat IP yang direkomendasikan ini adalah perkiraan untuk proksi dengan titik akhir default saja. Proksi dengan titik akhir tambahan atau replika baca mungkin memerlukan lebih banyak alamat IP kosong. Untuk setiap titik akhir tambahan, sebaiknya Anda mencadangkan tiga alamat IP lagi. Untuk setiap replika baca, sebaiknya Anda mencadangkan alamat IP tambahan seperti yang ditentukan dalam tabel berdasarkan ukuran replika baca tersebut.

> **Note**
>
> Proksi RDS tidak mendukung lebih dari 215 alamat IP di satu VPC.

Misalnya, Anda ingin memperkirakan alamat IP yang diperlukan untuk proksi yang terkait dengan klaster DB Aurora.

Dalam kasus ini, asumsikan hal berikut:

- Klaster DB Aurora Anda memiliki 1 instans penulis berukuran db.r5.8xlarge dan 1 instans pembaca berukuran db.r5.2xlarge.
- Proksi yang disematkan ke klaster DB ini memiliki titik akhir default dan 1 titik akhir kustom dengan peran hanya-baca.

Dalam hal ini, proksi memerlukan kira-kira 63 alamat IP kosong (45 untuk instans penulis, 15 untuk instans pembaca, dan 3 untuk titik akhir kustom tambahan).

Menyiapkan kredensi database di AWS Secrets Manager

Untuk setiap proksi yang Anda buat, pertama-tama Anda harus menggunakan layanan Secrets Manager untuk menyimpan kumpulan kredensial nama pengguna dan kata sandi. Anda dapat membuat rahasia Secrets Manager terpisah untuk setiap akun pengguna basis data yang terhubung ke klaster DB Aurora.

Dalam konsol Secrets Manager, Anda dapat membuat rahasia ini dengan nilai untuk kolom `username` dan `password`. Dengan melakukannya, proksi dapat terhubung ke pengguna basis data yang sesuai di klaster DB Aurora yang Anda kaitkan dengan proksi. Untuk melakukannya, Anda dapat menggunakan pengaturan Kredensial untuk basis data lain, Kredensial untuk basis data RDS, atau Jenis rahasia lainnya. Masukkan nilai yang sesuai di kolom Nama pengguna dan Kata sandi, dan nilai untuk kolom lain yang wajib diisi. Proksi akan mengabaikan kolom lain seperti Host dan Port jika ada dalam rahasia tersebut. Detail tersebut secara otomatis diisi oleh proksi.

Anda juga dapat memilih Jenis rahasia lainnya. Dalam kasus ini, Anda membuat rahasia dengan kunci bernama `username` dan `password`.

Karena rahasia yang digunakan oleh proksi Anda tidak terikat dengan server basis data tertentu, Anda dapat menggunakan kembali rahasia di beberapa proksi. Untuk melakukannya, gunakan kredensial yang sama di beberapa server basis data. Misalnya, Anda dapat menggunakan kredensial yang sama di seluruh server pengembangan dan pengujian.

Untuk terhubung melalui proksi sebagai pengguna basis data tertentu, pastikan kata sandi yang terkait dengan rahasia cocok dengan kata sandi basis data untuk pengguna tersebut. Jika ada ketidakcocokan, Anda dapat memperbarui rahasia terkait dalam Secrets Manager. Dalam kasus ini, Anda masih dapat terhubung ke akun lain yang memiliki kredensial rahasia dan kata sandi basis data yang cocok.

Saat Anda membuat proxy melalui AWS CLI atau RDS API, Anda menentukan Amazon Resource Names (ARN) dari rahasia yang sesuai. Anda melakukannya untuk semua akun pengguna DB yang dapat diakses proksi. Dalam AWS Management Console, Anda memilih rahasia dengan nama deskriptif mereka.

Untuk petunjuk tentang cara membuat rahasia di Secrets Manager, lihat halaman Membuat rahasia dalam dokumentasi Secrets Manager. Gunakan salah satu teknik berikut:

- Gunakan Secrets Manager di konsol.
- Untuk menggunakan CLI untuk membuat rahasia Secrets Manager untuk digunakan oleh Proksi RDS, gunakan perintah seperti berikut.

```
aws secretsmanager create-secret
  --name "secret_name"
  --description "secret_description"
  --region region_name
  --secret-string '{"username":"db_user","password":"db_user_password"}'
```

Misalnya, perintah berikut ini membuat rahasia Secrets Manager untuk dua pengguna basis data, yang satu bernama `admin` dan yang lain bernama `app-user`.

```
aws secretsmanager create-secret \
  --name admin_secret_name --description "db admin user" \
  --secret-string '{"username":"admin","password":"choose_your_own_password"}'

aws secretsmanager create-secret \
  --name proxy_secret_name --description "application user" \
  --secret-string '{"username":"app-user","password":"choose_your_own_password"}'
```

Untuk melihat rahasia yang dimiliki oleh AWS akun Anda, gunakan perintah seperti berikut ini.

```
aws secretsmanager list-secrets
```

Saat Anda membuat proksi menggunakan CLI, Anda meneruskan Amazon Resource Name (ARN) dari satu atau beberapa rahasia ke parameter `--auth`. Contoh Linux berikut menunjukkan cara menyiapkan laporan hanya dengan nama dan ARN dari setiap rahasia yang dimiliki oleh akun Anda AWS . Contoh ini menggunakan parameter `--output table` yang tersedia di AWS CLI versi 2. Jika Anda menggunakan AWS CLI versi 1, gunakan `--output text` sebagai gantinya.

```
aws secretsmanager list-secrets --query '*[].[Name,ARN]' --output table
```

Untuk memverifikasi bahwa Anda telah menyimpan kredensial yang benar dan dalam format yang benar dalam rahasia, gunakan perintah seperti berikut. Gantikan nama pendek atau ARN rahasia untuk *`your_secret_name`*.

```
aws secretsmanager get-secret-value --secret-id your_secret_name
```

Output harus mencakup baris yang menampilkan nilai yang dienkode JSON seperti berikut.

```
"SecretString": "{\"username\":\"your_username\",\"password\":\"your_password\"}",
```

Menyiapkan AWS Identity and Access Management kebijakan (IAM)

Setelah membuat rahasia di Secrets Manager, Anda dapat membuat kebijakan IAM yang dapat mengakses rahasia tersebut. Untuk informasi umum tentang cara menggunakan IAM, lihat Manajemen identitas dan akses untuk Amazon Aurora.

> **Tip**
>
> Prosedur berikut berlaku jika Anda menggunakan konsol IAM. Jika Anda menggunakan AWS Management Console for RDS, RDS dapat membuat kebijakan IAM untuk Anda secara otomatis. Dalam kasus ini, Anda dapat melewati prosedur berikut.

Untuk membuat kebijakan IAM yang mengakses rahasia Secrets Manager untuk digunakan oleh proksi Anda

1. Masuk ke konsol IAM. Ikuti proses Buat peran, seperti yang dijelaskan dalam Membuat peran IAM, memilih Membuat peran untuk mendelegasikan izin ke layanan. AWS

   Pilih Layanan AWS untuk Jenis entitas tepercaya. Di bagian Kasus penggunaan, pilih RDS dari menu drop-down Kasus penggunaan untuk layanan AWS . Pilih RDS – Tambahkan Peran ke basis data.

2. Untuk peran baru, lakukan langkah Tambahkan kebijakan sebaris. Gunakan prosedur umum yang sama seperti dalam Mengedit kebijakan IAM. Tempelkan JSON berikut ke dalam kotak teks JSON. Ganti ID akun Anda sendiri. Gantikan AWS Wilayah Anda untuk`us-east-2`. Ganti Amazon Resource Name (ARN) dengan rahasia yang Anda buat, lihat Menentukan kunci KMS dalam pernyataan kebijakan IAM. Untuk `kms:Decrypt` tindakan, gantikan ARN dari default AWS KMS key atau kunci KMS Anda sendiri. Mana yang Anda gunakan bergantung pada mana yang Anda gunakan untuk mengenkripsi rahasia Secrets Manager.

   ```
   {
       "Version": "2012-10-17",
       "Statement": [
           {
   ```

```
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource": [
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_1",
                "arn:aws:secretsmanager:us-east-2:account_id:secret:secret_name_2"
            ]
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "kms:Decrypt",
            "Resource": "arn:aws:kms:us-east-2:account_id:key/key_id",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": "secretsmanager.us-east-2.amazonaws.com"
                }
            }
        }
    ]
}
```

3. Edit kebijakan kepercayaan untuk peran IAM ini. Tempelkan JSON berikut ke dalam kotak teks JSON.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "rds.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Perintah berikut akan melakukan operasi yang sama melalui AWS CLI.

```
PREFIX=my_identifier
```

```
USER_ARN=$(aws sts get-caller-identity --query "Arn" --output text)

aws iam create-role --role-name my_role_name \
  --assume-role-policy-document '{"Version":"2012-10-17","Statement":
[{"Effect":"Allow","Principal":{"Service":
["rds.amazonaws.com"]},"Action":"sts:AssumeRole"}]}'

ROLE_ARN=arn:aws:iam::account_id:role/my_role_name

aws iam put-role-policy --role-name my_role_name \
  --policy-name $PREFIX-secret-reader-policy --policy-document
 '{"Version":"2012-10-17","Statement":
[{"Sid":"getsecretvalue","Effect":"Allow","Action":
["secretsmanager:GetSecretValue","kms:Decrypt"],"Resource":"*"}]}'

aws kms create-key --description "$PREFIX-test-key" --policy '{
  "Id":"$PREFIX-kms-policy",
  "Version":"2012-10-17",
  "Statement":
    [
      {
        "Sid":"Enable IAM User Permissions",
        "Effect":"Allow",
        "Principal":{"AWS":"arn:aws::iam:account_id:root"},
        "Action":"kms:*","Resource":"*"
      },
      {
        "Sid":"Allow access for Key Administrators",
        "Effect":"Allow",
        "Principal":
          {
            "AWS":
              ["$USER_ARN","arn:aws::iam:account_id:role/Admin"]
          },
        "Action":
          [
            "kms:Create*",
            "kms:Describe*",
            "kms:Enable*",
            "kms:List*",
            "kms:Put*",
            "kms:Update*",
            "kms:Revoke*",
            "kms:Disable*",
```

```
            "kms:Get*",
            "kms:Delete*",
            "kms:TagResource",
            "kms:UntagResource",
            "kms:ScheduleKeyDeletion",
            "kms:CancelKeyDeletion"
          ],
        "Resource":"*"
      },
      {
        "Sid":"Allow use of the key",
        "Effect":"Allow",
        "Principal":{"AWS":"$ROLE_ARN"},
        "Action":["kms:Decrypt","kms:DescribeKey"],
        "Resource":"*"
      }
    ]
}'
```

Membuat Proksi RDS

Untuk mengelola koneksi untuk klaster DB, buat proksi. Anda dapat mengaitkan sebuah proksi dengan klaster DB Aurora MySQL atau Aurora PostgreSQL.

AWS Management Console

Untuk membuat proksi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Proksi.
3. Pilih Buat proksi.
4. Pilih semua pengaturan untuk proksi Anda.

   Untuk Konfigurasi proksi, masukkan informasi untuk opsi berikut:

   - Keluarga mesin. Pengaturan ini menentukan protokol jaringan basis data mana yang dikenali proksi ketika menafsirkan lalu lintas jaringan ke dan dari basis data. Untuk Aurora MySQL, pilih MariaDB dan MySQL. Untuk Aurora PostgreSQL, pilih PostgreSQL.
   - ID proksi. Tentukan nama yang unik dalam ID AWS akun Anda dan AWS Wilayah saat ini.

- Batas waktu koneksi klien idle. Pilih periode waktu saat koneksi klien menjadi idle sebelum proksi menutupnya. Nilai default-nya adalah 1.800 detik (30 menit). Koneksi klien dianggap idle jika aplikasi tidak mengirimkan permintaan baru dalam waktu yang ditentukan setelah permintaan sebelumnya selesai. Koneksi basis data yang mendasarinya akan tetap terbuka dan dikembalikan ke kumpulan koneksi. Oleh karena itu, koneksi dapat digunakan kembali untuk koneksi klien baru.

  Jika Anda ingin proksi secara proaktif menghapus koneksi yang sudah tidak terpakai, kurangi batas waktu koneksi klien idle. Saat beban kerja meningkat, untuk menghemat biaya pembangunan koneksi, tambah batas waktu koneksi klien idle."

Untuk Konfigurasi grup target, masukkan informasi untuk opsi berikut:

- basis data. Pilih satu klaster DB Aurora yang akan diakses melalui proksi ini. Daftar ini hanya mencakup instans dan klaster DB dengan mesin basis data, versi mesin, dan pengaturan lainnya yang kompatibel. Jika daftar kosong, buat instans atau klaster DB baru yang kompatibel dengan Proksi RDS. Untuk melakukannya, ikuti prosedur dalam Membuat klaster DB Amazon Aurora. Lalu, coba buat proksi lagi.
- Koneksi maksimum kumpulan koneksi. Tentukan sebuah nilai dari 1 hingga 100. Pengaturan ini merepresentasikan persentase dari nilai `max_connections` yang dapat digunakan oleh Proksi RDS untuk koneksinya. Jika hanya ingin menggunakan satu proksi dengan instans atau klaster DB ini, Anda dapat mengatur nilai ini ke 100. Untuk detail tentang cara Proksi RDS menggunakan pengaturan ini, lihat MaxConnectionsPercent.
- Filter penyematan sesi. (Opsional) Opsi ini memungkinkan Anda untuk memaksa Proksi RDS untuk tidak memberi pin untuk jenis status sesi tertentu yang terdeteksi. Tindakan ini menghindari langkah-langkah keamanan default untuk me-multipleks koneksi basis data di seluruh koneksi klien. Saat ini, pengaturan tidak didukung untuk PostgreSQL. Satu-satunya pilihan adalah `EXCLUDE_VARIABLE_SETS`.

  Mengaktifkan pengaturan ini dapat menyebabkan variabel sesi dari satu koneksi memengaruhi koneksi lain. Hal ini dapat menyebabkan kesalahan atau masalah ketepatan jika kueri Anda bergantung pada nilai variabel sesi yang ditetapkan di luar transaksi saat ini. Pertimbangkan untuk menggunakan opsi ini setelah memastikan bahwa aplikasi Anda sudah bisa berbagi koneksi basis data dengan aman di seluruh koneksi klien.

  Pola berikut bisa dianggap aman:

  - Pernyataan SET di mana tidak ada perubahan pada nilai variabel sesi efektif, yaitu tidak ada perubahan pada variabel sesi.
  - Anda mengubah nilai variabel sesi dan mengeksekusi pernyataan dalam transaksi yang sama.

  Untuk informasi selengkapnya, lihat Menghindari penyematan.

- Batas waktu peminjaman koneksi. Dalam beberapa kasus, mungkin Anda berharap proksi tersebut sekali-sekali menggunakan semua koneksi basis data yang tersedia. Dalam kasus seperti itu, Anda dapat menentukan durasi proksi harus menunggu koneksi basis data menjadi tersedia sebelum menampilkan kesalahan batas waktu. Anda dapat menentukan periode hingga maksimum lima menit. Pengaturan ini hanya berlaku jika proksi memiliki koneksi terbuka maksimum dan semua koneksi sudah digunakan.
- Kueri inisialisasi. (Opsional) Anda dapat menentukan satu atau beberapa pernyataan SQL agar proksi berjalan saat membuka setiap koneksi basis data baru. Pengaturan ini biasanya digunakan dengan pernyataan SET untuk memastikan bahwa setiap koneksi memiliki pengaturan yang identik seperti zona waktu dan kumpulan karakter. Untuk beberapa pernyataan, gunakan titik koma sebagai pemisah. Anda juga dapat menyertakan beberapa variabel dalam satu pernyataan SET, seperti `SET x=1, y=2`.

Untuk Autentikasi, masukkan informasi untuk opsi berikut:

- Peran IAM. Pilih peran IAM yang memiliki izin untuk mengakses rahasia Secrets Manager yang Anda pilih sebelumnya. Atau, Anda dapat membuat peran IAM baru dari AWS Management Console.
- Rahasia Secrets Manager. Pilih setidaknya satu rahasia Secrets Manager terpisah yang berisi kredensial pengguna basis data yang memungkinkan proksi mengakses klaster DB Aurora.
- Jenis autentikasi klien. Pilih jenis autentikasi yang digunakan proksi untuk koneksi dari klien. Pilihan Anda berlaku untuk semua rahasia Secrets Manager yang Anda kaitkan dengan proksi ini. Jika Anda perlu menentukan jenis otentikasi klien yang berbeda untuk setiap rahasia, maka buat proxy Anda dengan menggunakan AWS CLI atau API sebagai gantinya.
- Autentikasi IAM. Pilih apakah akan mewajibkan , atau melarang autentikasi IAM untuk koneksi ke proksi Anda. Pilihan Anda berlaku untuk semua rahasia Secrets Manager yang Anda kaitkan dengan proksi ini. Jika Anda perlu menentukan otentikasi IAM yang berbeda untuk setiap rahasia, buat proxy Anda dengan menggunakan AWS CLI atau API sebagai gantinya.

Untuk Konektivitas, masukkan informasi untuk opsi berikut:

- Wajibkan Keamanan Lapisan Pengangkutan. Pilih pengaturan ini jika Anda ingin proksi menerapkan TLS/SSL untuk semua koneksi klien. Untuk koneksi terenkripsi atau tidak terenkripsi ke sebuah proksi, proksi menggunakan pengaturan enkripsi yang sama saat membuat koneksi ke basis data acuan.
- Subnet. Bidang ini telah diisi sebelumnya dengan semua subnet yang terkait dengan VPC Anda. Anda dapat menghapus subnet apa pun yang tidak diperlukan oleh proksi ini. Anda harus membiarkan setidaknya dua subnet.

Masukkan konfigurasi konektivitas tambahan:

- Grup keamanan VPC. Pilih grup keamanan VPC yang sudah ada. Atau, Anda dapat membuat grup keamanan baru dari AWS Management Console. Anda harus mengonfigurasi Aturan masuk untuk mengizinkan aplikasi Anda mengakses proksi. Anda juga harus mengonfigurasi Aturan keluar untuk mengizinkan lalu lintas dari target DB Anda.

  > **Note**
  >
  > Grup keamanan ini harus mengizinkan koneksi dari proksi ke basis data. Grup keamanan yang sama digunakan sebagai jalur masuk dari aplikasi ke proksi, dan jalur keluar dari proksi ke basis data. Misalnya, anggap saja Anda menggunakan grup keamanan yang sama untuk basis data dan proksi Anda. Dalam kasus ini, pastikan sumber daya dalam grup keamanan tersebut dapat berkomunikasi dengan sumber daya lain dalam grup keamanan yang sama.
  > Saat menggunakan VPC bersama, Anda tidak dapat menggunakan grup keamanan default untuk VPC, atau grup keamanan milik akun lain. Pilih grup keamanan milik akun Anda. Jika belum ada, buat satu. Untuk informasi selengkapnya tentang batasan ini, lihat Bekerja dengan VPC bersama.

  RDS mendeploy proksi di beberapa Zona Ketersediaan untuk memastikan ketersediaan yang tinggi. Untuk mengaktifkan komunikasi lintas-AZ untuk proksi semacam ini, daftar kontrol akses (ACL) untuk subnet proksi Anda harus mengizinkan jalan keluar khusus port mesin dan semua port untuk masuk. Untuk informasi selengkapnya tentang ACL jaringan, lihat Mengontrol lalu lintas ke subnet menggunakan ACL jaringan. Jika ACL jaringan untuk proksi

dan target Anda identik, Anda harus menambahkan aturan masuknya protokol TCP tempat Sumber diatur ke CIDR VPC. Anda juga harus menambahkan aturan keluar protokol TCP khusus port mesin tempat Sumber diatur ke CIDR VPC.

(Opsional) Masukkan konfigurasi lanjutan:

- Aktifkan pengelogan yang disempurnakan. Anda dapat mengaktifkan pengaturan ini untuk memecahkan masalah kompatibilitas proksi atau masalah performa.

  Jika pengaturan ini diaktifkan, Proksi RDS akan menyertakan informasi mendetail tentang pernyataan SQL dalam log-nya. Informasi ini membantu Anda untuk men-debug masalah yang melibatkan perilaku SQL atau performa serta skalabilitas koneksi proksi. Informasi debug mencakup teks dari pernyataan SQL yang Anda kirimkan melalui proksi. Oleh karena, hanya aktifkan pengaturan ini untuk debugging atau jika Anda memiliki prosedur keamanan untuk melindungi informasi sensitif apa pun yang muncul dalam log.

  Untuk meminimalkan overhead yang terkait dengan proksi Anda, Proksi RDS secara otomatis menonaktifkan pengaturan ini 24 jam setelah Anda mengaktifkannya. Aktifkan sementara untuk memecahkan masalah tertentu.

5. Pilih Buat Proksi.

AWS CLI

Untuk membuat proxy dengan menggunakan AWS CLI, panggil create-db-proxyperintah dengan parameter yang diperlukan berikut:

- `--db-proxy-name`
- `--engine-family`
- `--role-arn`
- `--auth`
- `--vpc-subnet-ids`

Nilai `--engine-family` ini bersifat peka huruf besar-kecil.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-proxy \
    --db-proxy-name proxy_name \
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } \
    --auth ProxyAuthenticationConfig_JSON_string \
    --role-arn iam_role \
    --vpc-subnet-ids space_separated_list \
    [--vpc-security-group-ids space_separated_list] \
    [--require-tls | --no-require-tls] \
    [--idle-client-timeout value] \
    [--debug-logging | --no-debug-logging] \
    [--tags comma_separated_list]
```

Untuk Windows:

```
aws rds create-db-proxy ^
    --db-proxy-name proxy_name ^
    --engine-family { MYSQL | POSTGRESQL | SQLSERVER } ^
    --auth ProxyAuthenticationConfig_JSON_string ^
    --role-arn iam_role ^
    --vpc-subnet-ids space_separated_list ^
    [--vpc-security-group-ids space_separated_list] ^
    [--require-tls | --no-require-tls] ^
    [--idle-client-timeout value] ^
    [--debug-logging | --no-debug-logging] ^
    [--tags comma_separated_list]
```

Berikut ini adalah contoh nilai JSON untuk opsi `--auth`. Contoh ini menerapkan jenis autentikasi klien yang berbeda untuk setiap rahasia.

```
[
  {
    "Description": "proxy description 1",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789123:secret/1234abcd-12ab-34cd-56ef-1234567890ab",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_SCRAM_SHA_256"
  },

  {
    "Description": "proxy description 2",
    "AuthScheme": "SECRETS",
```

```
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122223333:seret/1234abcd-12ab-34cd-56ef-1234567890cd",
    "IAMAuth": "DISABLED",
    "ClientPasswordAuthType": "POSTGRES_MD5"

  },

  {
    "Description": "proxy description 3",
    "AuthScheme": "SECRETS",
    "SecretArn": "arn:aws:secretsmanager:us-
west-2:111122221111:secret/1234abcd-12ab-34cd-56ef-1234567890ef",
    "IAMAuth": "REQUIRED"
  }

]
```

Tip

Jika Anda belum tahu ID subnet yang akan digunakan untuk parameter `--vpc-subnet-ids`, lihat Menyiapkan prasyarat jaringan untuk contoh tentang cara menemukannya.

Note

Grup keamanan ini harus mengizinkan akses ke basis data yang terhubung ke proksi. Grup keamanan yang sama digunakan sebagai jalur masuk dari aplikasi ke proksi, dan jalur keluar dari proksi ke basis data. Misalnya, anggap saja Anda menggunakan grup keamanan yang sama untuk basis data dan proksi Anda. Dalam kasus ini, pastikan sumber daya dalam grup keamanan tersebut dapat berkomunikasi dengan sumber daya lain dalam grup keamanan yang sama.

Saat menggunakan VPC bersama, Anda tidak dapat menggunakan grup keamanan default untuk VPC, atau grup keamanan milik akun lain. Pilih grup keamanan milik akun Anda. Jika belum ada, buat satu. Untuk informasi selengkapnya tentang batasan ini, lihat Bekerja dengan VPC bersama.

Untuk membuat asosiasi yang tepat untuk proxy, Anda juga menggunakan register-db-proxy-targetsperintah. Tentukan nama grup target `default`. Proksi RDS secara otomatis membuat grup target dengan nama ini saat Anda membuat setiap proksi.

```
aws rds register-db-proxy-targets
    --db-proxy-name value
    [--target-group-name target_group_name]
    [--db-instance-identifiers space_separated_list]  # rds db instances, or
    [--db-cluster-identifiers cluster_id]        # rds db cluster (all instances)
```

API RDS

Untuk membuat proksi RDS, panggil operasi API Amazon RDS CreateDBProxy. Anda melewati parameter dengan struktur AuthConfigdata.

Proksi RDS secara otomatis membuat grup target bernama `default` saat Anda membuat setiap proksi. Anda mengaitkan Aurora DB cluster dengan grup target dengan memanggil fungsi registerDB. ProxyTargets

Melihat Proksi RDS

Setelah membuat satu atau beberapa proksi RDS, Anda dapat melihat semuanya. Dengan begitu, Anda dapat memeriksa detail konfigurasinya dan memilih mana yang akan dimodifikasi, dihapus, dan sebagainya.

Agar aplikasi basis data dapat menggunakan proksi, Anda harus menyediakan titik akhir proksi dalam string koneksi.

AWS Management Console

Untuk melihat proksi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas AWS Management Console, pilih AWS Wilayah tempat Anda membuat Proxy RDS.
3. Di panel navigasi, pilih Proksi.
4. Pilih nama proksi RDS untuk menampilkan detailnya.
5. Pada halaman detail, bagian Grup target menunjukkan bagaimana proksi dikaitkan dengan klaster DB Aurora. Anda dapat mengikuti tautan ke halaman grup target default untuk melihat

detail selengkapnya tentang pengaitan antara proksi dan basis data. Halaman ini adalah tempat Anda melihat pengaturan yang Anda tentukan saat membuat proksi. Ini termasuk persentase koneksi maksimum, batas waktu peminjaman koneksi, keluarga mesin, dan filter penyematan sesi.

CLI

Untuk melihat proxy Anda menggunakan CLI, gunakan perintah. describe-db-proxies Secara default, ini menampilkan semua proxy yang dimiliki oleh akun Anda AWS . Untuk melihat detail dari satu proksi, masukkan namanya dengan parameter `--db-proxy-name`.

```
aws rds describe-db-proxies [--db-proxy-name proxy_name]
```

Untuk melihat informasi lain yang terkait dengan proksi, gunakan perintah berikut.

```
aws rds describe-db-proxy-target-groups  --db-proxy-name proxy_name

aws rds describe-db-proxy-targets --db-proxy-name proxy_name
```

Gunakan urutan perintah berikut untuk melihat detail selengkapnya tentang hal-hal yang terkait dengan proksi:

1. Untuk mendapatkan daftar proxy, jalankan. describe-db-proxies
2. Untuk menampilkan parameter koneksi seperti persentase maksimum koneksi yang dapat digunakan proxy, jalankan describe-db-proxy-target-groups`--db-proxy-name`. Gunakan nama proksi sebagai nilai parameter.
3. Untuk melihat detail cluster Aurora yang terkait dengan grup target yang dikembalikan, jalankan. describe-db-proxy-targets

API RDS

Untuk melihat proksi Anda menggunakan API RDS, gunakan operasi DescribeDBProxies. Operasi ini akan menampilkan nilai dari jenis data DBProxy.

Untuk melihat detail pengaturan koneksi untuk proxy, gunakan pengidentifikasi proxy dari nilai pengembalian ini dengan operasi ProxyTargetGroupsDescribeDB. Ia mengembalikan nilai-nilai dari tipe ProxyTargetGroup data DB.

Untuk melihat instance RDS atau cluster Aurora DB yang terkait dengan proxy, gunakan operasi DescribeDB. ProxyTargets Ia mengembalikan nilai-nilai dari tipe ProxyTarget data DB.

Terhubung ke basis data melalui Proksi RDS

Anda dapat terhubung ke klaster DB Aurora atau klaster yang menggunakan Aurora Serverless v2 melalui proksi seperti halnya Anda terhubung langsung ke basis data. Perbedaan utamanya adalah Anda menentukan titik akhir proksi, bukan titik akhir klaster. Secara default, semua koneksi proksi memiliki kemampuan baca/tulis dan menggunakan instans penulis. Jika Anda biasanya menggunakan titik akhir pembaca untuk koneksi hanya-baca, Anda dapat membuat titik akhir hanya-baca tambahan untuk proksi. Anda dapat menggunakan titik akhir tersebut dengan cara yang sama. Untuk informasi selengkapnya, lihat Ikhtisar titik akhir proksi.

Topik

- Terhubung ke sebuah proksi menggunakan autentikasi native
- Terhubung ke sebuah proksi menggunakan autentikasi IAM
- Pertimbangan untuk terhubung ke sebuah proksi dengan PostgreSQL

Terhubung ke sebuah proksi menggunakan autentikasi native

Gunakan langkah berikut untuk terhubung ke proksi menggunakan autentikasi native:

1. Temukan titik akhir proksi. Di AWS Management Console, Anda dapat menemukan titik akhir pada halaman detail untuk proxy yang sesuai. Dengan itu AWS CLI, Anda dapat menggunakan describe-db-proxiesperintah. Contoh berikut menunjukkan caranya.

```
# Add --output text to get output as a simple tab-separated list.
$ aws rds describe-db-proxies --query '*[*].
{DBProxyName:DBProxyName,Endpoint:Endpoint}'
[
    [
        {
            "Endpoint": "the-proxy.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy"
        },
        {
            "Endpoint": "the-proxy-other-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-other-secret"
```

```
        },
        {
            "Endpoint": "the-proxy-rds-secret.proxy-demo.us-
east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-rds-secret"
        },
        {
            "Endpoint": "the-proxy-t3.proxy-demo.us-east-1.rds.amazonaws.com",
            "DBProxyName": "the-proxy-t3"
        }
    ]
]
```

2. Tentukan titik akhir sebagai parameter host dalam string koneksi untuk aplikasi klien Anda. Misalnya, tentukan titik akhir proksi sebagai nilai untuk opsi `mysql -h` atau opsi `psql -h`.
3. Masukkan nama dan kata sandi pengguna basis data yang sama seperti biasanya.

Terhubung ke sebuah proksi menggunakan autentikasi IAM

Saat Anda menggunakan autentikasi IAM dengan Proksi RDS, siapkan pengguna basis data Anda untuk melakukan autentikasi dengan nama dan kata sandi pengguna reguler. Autentikasi IAM berlaku untuk Proksi RDS yang mengambil kredensial nama dan kata sandi pengguna dari Secrets Manager. Koneksi dari Proksi RDS ke basis data acuan tidak melewati IAM.

Untuk terhubung ke Proksi RDS menggunakan autentikasi IAM, gunakan prosedur koneksi umum yang sama seperti autentikasi IAM dengan klaster DB Aurora. Untuk informasi umum tentang cara menggunakan IAM, lihat Keamanan dalam Amazon Aurora.

Perbedaan utama dalam penggunaan IAM untuk Proksi RDS meliputi:

- Anda tidak dapat mengonfigurasi setiap pengguna basis data dengan plugin otorisasi. Pengguna basis data masih memiliki nama dan kata sandi pengguna reguler dalam basis data. Anda dapat menyiapkan rahasia Secrets Manager yang berisi nama dan kata sandi pengguna ini, dan mengotorisasi Proksi RDS untuk mengambil kredensial dari Secrets Manager.

  Autentikasi IAM berlaku untuk koneksi antara program klien Anda dan proksi. Proksi kemudian melakukan autentikasi ke basis data menggunakan kredensial nama dan kata sandi pengguna yang diambil dari Secrets Manager.
- Anda menentukan titik akhir proksi, bukan instans, klaster, atau titik akhir pembaca. Untuk detail tentang titik akhir proksi, lihat Menghubungkan ke instans DB menggunakan autentikasi IAM.

- Dalam kasus autentikasi IAM basis data langsung, Anda secara selektif memilih pengguna basis data dan mengonfigurasinya untuk diidentifikasi dengan plugin autentikasi khusus. Anda kemudian dapat terhubung ke pengguna tersebut menggunakan autentikasi IAM.

  Dalam kasus penggunaan proksi, Anda memberi proksi Rahasia yang berisi nama pengguna dan kata sandi pengguna tertentu (autentikasi native). Anda kemudian terhubung ke proksi menggunakan autentikasi IAM. Di sini, Anda melakukannya dengan membuat token autentikasi dengan titik akhir proksi, bukan titik akhir basis data. Anda juga menggunakan nama pengguna yang cocok dengan salah satu nama pengguna untuk rahasia yang Anda berikan.
- Pastikan Anda menggunakan Keamanan Lapisan Pengangkutan (TLS)/Lapisan Soket Aman (SSL) saat terhubung ke sebuah proksi menggunakan autentikasi IAM.

Anda dapat memberi pengguna tertentu akses ke proksi dengan mengubah kebijakan IAM. Berikut contohnya.

```
"Resource": "arn:aws:rds-db:us-east-2:1234567890:dbuser:prx-ABCDEFGHIJKL01234/db_user"
```

Pertimbangan untuk terhubung ke sebuah proksi dengan PostgreSQL

Untuk PostgreSQL, saat klien memulai koneksi ke basis data PostgreSQL, pesan startup akan dikirimkan. Pesan ini berisi pasangan nama parameter dan string nilai. Untuk detailnya, lihat `StartupMessage` dalam Format pesan PostgreSQL dalam dokumentasi PostgreSQL.

Saat terhubung melalui proksi RDS, pesan startup bisa menyertakan parameter yang dikenal saat ini sebagai berikut:

- `user`
- `database`
- `replication`

Pesan startup juga bisa menyertakan parameter runtime tambahan berikut:

- `application_name`
- `client_encoding`
- `DateStyle`
- `TimeZone`

- `extra_float_digits`

Untuk informasi selengkapnya tentang pesan PostgreSQL, lihat Protokol Frontend/Backend dalam dokumentasi PostgreSQL.

Untuk PostgreSQL, jika Anda menggunakan JDBC, sebaiknya lakukan tindakan berikut untuk menghindari penyematan:

- Atur parameter koneksi JDBC `assumeMinServerVersion` ke setidaknya `9.0` untuk menghindari penyematan. Tindakan ini dapat mencegah driver JDBC melakukan perjalanan roundtrip ekstra selama startup koneksi saat menjalankan `SET extra_float_digits = 3`.
- Atur parameter koneksi JDBC `ApplicationName` ke *`any/your-application-name`* untuk menghindari penyematan. Tindakan ini dapat mencegah driver JDBC melakukan roundtrip ekstra selama startup koneksi saat menjalankan `SET application_name = "PostgreSQL JDBC Driver"`. Perhatikan bahwa parameter JDBC adalah `ApplicationName`, tetapi parameter PostgreSQL `StartupMessage` adalah `application_name`.

Untuk informasi selengkapnya, lihat Menghindari penyematan. Untuk informasi selengkapnya tentang cara terhubung menggunakan JDBC, lihat Terhubung ke basis data dalam dokumentasi PostgreSQL.

Mengelola Proksi RDS

Bagian ini berisi informasi tentang cara mengelola operasi dan konfigurasi Proksi RDS. Prosedur ini membantu aplikasi Anda memaksimalkan koneksi basis data dan mencapai penggunaan ulang koneksi maksimum. Semakin banyak yang dapat Anda manfaatkan dari penggunaan ulang koneksi, semakin banyak overhead CPU dan memori yang bisa dihemat. Pada akhirnya, tindakan ini dapat mengurangi latensi untuk aplikasi Anda dan memungkinkan basis data untuk mendedikasikan lebih banyak sumber dayanya untuk memproses permintaan aplikasi.

Topik

- Mengubah Proksi RDS
- Menambahkan pengguna basis data baru
- Mengubah kata sandi untuk pengguna basis data
- Koneksi klien dan basis data
- Mengonfigurasi pengaturan koneksi

- Menghindari penyematan
- Menghapus Proksi RDS

Mengubah Proksi RDS

Anda dapat mengubah pengaturan spesifik yang terkait dengan proksi setelah Anda membuat proksi. Caranya adalah dengan mengubah proksi itu sendiri, grup target terkaitnya, atau keduanya. Setiap proksi memiliki satu grup target terkait.

AWS Management Console

> **Important**
>
> Nilai dalam kolom Jenis autentikasi klien dan Autentikasi IAM berlaku untuk semua rahasia Secrets Manager yang terkait dengan proksi ini. Untuk menentukan nilai yang berbeda untuk setiap rahasia, ubah proxy Anda dengan menggunakan AWS CLI atau API sebagai gantinya.

Untuk mengubah pengaturan proksi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Proksi.
3. Dalam daftar proksi, pilih proksi yang pengaturannya ingin diubah atau kunjungi halaman detailnya.
4. Untuk Tindakan, pilih Ubah.
5. Masukkan atau pilih properti yang akan diubah. Anda dapat mengubah opsi berikut:
   - ID proksi – Mengganti nama proksi dengan memasukkan ID baru.
   - Batas waktu koneksi klien idle – Masukkan periode waktu untuk batas waktu koneksi klien idle.
   - Peran IAM – Mengubah peran IAM yang digunakan untuk mengambil rahasia dari Secrets Manager.
   - Rahasia Secrets Manager – Menambahkan atau membuang rahasia Secrets Manager. Rahasia ini sesuai dengan nama dan kata sandi pengguna basis data.
   - Jenis autentikasi klien – (PostgreSQL saja) Mengubah jenis autentikasi untuk koneksi klien ke proksi.

- Autentikasi IAM – Mewajibkan atau melarang autentikasi IAM untuk koneksi ke proksi.
- Wajibkan Keamanan Lapisan Pengangkutan – Mengaktifkan atau menonaktifkan persyaratan untuk Keamanan Lapisan Pengangkutan (TLS).
- Grup keamanan VPC – Menambahkan atau menghapus grup keamanan VPC yang akan digunakan proksi.
- Aktifkan pencatatan log yang disempurnakan – Mengaktifkan atau menonaktifkan pencatatan log yang disempurnakan.

6. Pilih Ubah.

Jika pengaturan yang ingin diubah tidak tercantum, gunakan prosedur berikut untuk memperbarui grup target untuk proksi. Grup target yang terkait dengan proksi mengontrol pengaturan yang terkait dengan koneksi basis data fisik. Setiap proksi memiliki satu grup target terkait bernama `default`, yang dibuat secara otomatis dengan proksi.

Anda hanya dapat mengubah grup target dari halaman detail proksi, bukan dari daftar pada halaman Proksi.

Untuk mengubah pengaturan grup target proksi

1. Pada halaman Proksi, buka halaman detail proksi.
2. Untuk Grup target, pilih tautan `default`. Saat ini, semua proksi memiliki satu grup target bernama `default`.
3. Pada halaman detail grup target default, pilih Ubah.
4. Pilih pengaturan baru untuk properti yang dapat diubah:

   - Basis data — Pilih klaster Aurora.
   - Koneksi maksimum kumpulan koneksi – Sesuaikan persentase maksimum koneksi yang tersedia yang dapat digunakan proksi.
   - Filter penyematan sesi – (Opsional) Pilih filter penyematan sesi. Tindakan ini menghindari langkah-langkah keamanan default untuk me-multipleks koneksi basis data di seluruh koneksi klien. Saat ini, pengaturan tidak didukung untuk PostgreSQL. Satu-satunya pilihan adalah `EXCLUDE_VARIABLE_SETS`.

     Mengaktifkan pengaturan ini dapat menyebabkan variabel sesi dari satu koneksi memengaruhi koneksi lain. Hal ini dapat menyebabkan kesalahan atau masalah ketepatan jika kueri Anda bergantung pada nilai variabel sesi yang ditetapkan di luar transaksi saat ini. Pertimbangkan

untuk menggunakan opsi ini setelah memastikan bahwa aplikasi Anda sudah bisa berbagi koneksi basis data dengan aman di seluruh koneksi klien.

Pola berikut bisa dianggap aman:

- Pernyataan `SET` di mana tidak ada perubahan pada nilai variabel sesi efektif, yaitu tidak ada perubahan pada variabel sesi.
- Anda mengubah nilai variabel sesi dan mengeksekusi pernyataan dalam transaksi yang sama.

Untuk informasi selengkapnya, lihat Menghindari penyematan.

- Batas waktu peminjaman koneksi – Sesuaikan interval batas waktu peminjaman koneksi. Pengaturan ini berlaku saat jumlah maksimum koneksi sudah digunakan semua untuk proksi. Pengaturan ini menentukan seberapa lama proksi harus menunggu koneksi menjadi tersedia sebelum menampilkan sebuah kesalahan batas waktu.
- Kueri inisialisasi – (Opsional) Tambahkan kueri inisialisasi, atau ubah kueri inisialisasi ini. Anda dapat menentukan satu atau beberapa pernyataan SQL untuk proksi yang akan dijalankan saat membuka setiap koneksi basis data baru. Pengaturan ini biasanya digunakan dengan pernyataan `SET` untuk memastikan bahwa setiap koneksi memiliki pengaturan yang identik seperti zona waktu dan kumpulan karakter. Untuk beberapa pernyataan, gunakan titik koma sebagai pemisah. Anda juga dapat menyertakan beberapa variabel dalam satu pernyataan `SET`, seperti `SET x=1, y=2`.

Anda tidak dapat mengubah properti tertentu, seperti ID grup target dan mesin basis data.

5. Pilih Ubah grup target.

AWS CLI

Untuk memodifikasi proxy menggunakan AWS CLI, gunakan perintah modify-db-proxy, modify-db-proxy-target-group deregister-db-proxy-targets, dan register-db-proxy-targets.

Dengan perintah `modify-db-proxy`, Anda dapat mengubah properti seperti berikut:

- Kumpulan rahasia Secrets Manager yang digunakan proksi.
- Apakah TLS diperlukan.
- Batas waktu klien idle.
- Apakah harus mencatat informasi tambahan dari pernyataan SQL untuk debugging.

- Peran IAM yang digunakan untuk mengambil rahasia Secrets Manager.
- Grup keamanan yang digunakan proksi.

Contoh berikut menunjukkan cara mengganti nama proksi yang sudah ada.

```
aws rds modify-db-proxy --db-proxy-name the-proxy --new-db-proxy-name the_new_name
```

Untuk mengubah pengaturan terkait koneksi atau mengganti nama grup target, gunakan perintah `modify-db-proxy-target-group`. Saat ini, semua proksi memiliki satu grup target bernama `default`. Saat bekerja dengan grup target ini, Anda menentukan nama proksi dan `default` untuk nama grup target.

Contoh berikut ini menunjukkan cara memeriksa pengaturan `MaxIdleConnectionsPercent` untuk proksi terlebih dahulu dan kemudian mengubahnya menggunakan grup target.

```
aws rds describe-db-proxy-target-groups --db-proxy-name the-proxy

{
    "TargetGroups": [
        {
            "Status": "available",
            "UpdatedDate": "2019-11-30T16:49:30.342Z",
            "ConnectionPoolConfig": {
                "MaxIdleConnectionsPercent": 50,
                "ConnectionBorrowTimeout": 120,
                "MaxConnectionsPercent": 100,
                "SessionPinningFilters": []
            },
            "TargetGroupName": "default",
            "CreatedDate": "2019-11-30T16:49:27.940Z",
            "DBProxyName": "the-proxy",
            "IsDefault": true
        }
    ]
}

aws rds modify-db-proxy-target-group --db-proxy-name the-proxy --target-group-name
 default --connection-pool-config '
{ "MaxIdleConnectionsPercent": 75 }'

{
```

```
    "DBProxyTargetGroup": {
        "Status": "available",
        "UpdatedDate": "2019-12-02T04:09:50.420Z",
        "ConnectionPoolConfig": {
            "MaxIdleConnectionsPercent": 75,
            "ConnectionBorrowTimeout": 120,
            "MaxConnectionsPercent": 100,
            "SessionPinningFilters": []
        },
        "TargetGroupName": "default",
        "CreatedDate": "2019-11-30T16:49:27.940Z",
        "DBProxyName": "the-proxy",
        "IsDefault": true
    }
}
```

Dengan perintah `deregister-db-proxy-targets` dan `register-db-proxy-targets`, Anda dapat mengubah klaster DB Aurora mana yang dikaitkan dengan proksi melalui grup targetnya. Saat ini, setiap proksi dapat terhubung ke satu klaster DB Aurora. Grup target melacak detail koneksi untuk semua semua instans DB dalam klaster Aurora.

Contoh berikut dimulai dengan proksi yang dikaitkan dengan klaster Aurora MySQL bernama `cluster-56-2020-02-25-1399`. Contoh ini menunjukkan cara mengubah proksi sehingga dapat terhubung ke klaster lainnya yang bernama `provisioned-cluster`.

Saat menggunakan klaster DB Aurora, Anda dapat menentukan opsi `--db-cluster-identifier`.

Contoh berikut mengubah proksi Aurora MySQL. Proksi Aurora PostgreSQL memiliki port 5432.

```
aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": [
        {
            "Endpoint": "instance-9814.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-9814"
        },
        {
            "Endpoint": "instance-8898.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
```

```
            "RdsResourceId": "instance-8898"
        },
        {
            "Endpoint": "instance-1018.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-1018"
        },
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "cluster-56-2020-02-25-1399"
        },
        {
            "Endpoint": "instance-4330.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "instance-4330"
        }
    ]
}

aws rds deregister-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 cluster-56-2020-02-25-1399

aws rds describe-db-proxy-targets --db-proxy-name the-proxy

{
    "Targets": []
}

aws rds register-db-proxy-targets --db-proxy-name the-proxy --db-cluster-identifier
 provisioned-cluster

{
    "DBProxyTargets": [
        {
            "Type": "TRACKED_CLUSTER",
            "Port": 0,
            "RdsResourceId": "provisioned-cluster"
        },
        {
            "Endpoint": "gkldje.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
```

```
            "Port": 3306,
            "RdsResourceId": "gkldje"
        },
        {
            "Endpoint": "provisioned-1.demo.us-east-1.rds.amazonaws.com",
            "Type": "RDS_INSTANCE",
            "Port": 3306,
            "RdsResourceId": "provisioned-1"
        }
    ]
}
```

API RDS

Untuk memodifikasi proxy menggunakan RDS API, Anda menggunakan operasi ModifyDBProxy, ModifyDB, DeregisterDB, dan operasi RegisterDB. ProxyTargetGroup ProxyTargets ProxyTargets

Dengan `ModifyDBProxy`, Anda dapat mengubah properti seperti berikut:

- Kumpulan rahasia Secrets Manager yang digunakan proksi.
- Apakah TLS diperlukan.
- Batas waktu klien idle.
- Apakah harus mencatat informasi tambahan dari pernyataan SQL untuk debugging.
- Peran IAM yang digunakan untuk mengambil rahasia Secrets Manager.
- Grup keamanan yang digunakan proksi.

Dengan `ModifyDBProxyTargetGroup`, Anda dapat mengubah pengaturan terkait koneksi atau mengganti nama grup target. Saat ini, semua proksi memiliki satu grup target bernama `default`. Saat bekerja dengan grup target ini, Anda menentukan nama proksi dan `default` untuk nama grup target.

Dengan `DeregisterDBProxyTargets` dan `RegisterDBProxyTargets`, Anda dapat mengubah klaster Aurora mana yang dikaitkan dengan proksi melalui grup targetnya. Saat ini, setiap proksi dapat terhubung ke satu klaster Aurora. Grup target melacak detail koneksi untuk instans DB dalam klaster Aurora.

Menambahkan pengguna basis data baru

Dalam beberapa kasus, Anda dapat menambahkan pengguna basis data baru ke klaster Aurora yang terkait dengan proksi. Jika demikian, tambahkan atau ganti tujuan sebuah rahasia Secrets Manager untuk menyimpan kredensial dari pengguna tersebut. Untuk melakukan ini, pilih salah satu opsi berikut:

1. Buat rahasia Secrets Manager yang baru, dengan menggunakan prosedur yang dijelaskan dalam Menyiapkan kredensi database di AWS Secrets Manager.
2. Perbarui peran IAM untuk memberi Proksi RDS akses ke rahasia Secrets Manager baru. Untuk melakukannya, perbarui bagian sumber daya dari kebijakan peran IAM.
3. Ubah Proksi RDS untuk menambahkan rahasia Secrets Manager baru di bagian Rahasia Secrets Manager.
4. Jika pengguna baru menggantikan yang sudah ada, perbarui kredensial yang tersimpan dalam rahasia Secrets Manager proksi untuk pengguna yang sudah ada.

Menambahkan pengguna basis data baru ke basis data PostgreSQL

Saat menambahkan pengguna baru ke database PostgreSQL Anda, jika Anda telah menjalankan perintah berikut:

```
REVOKE CONNECT ON DATABASE postgres FROM PUBLIC;
```

Berikan hak istimewa `CONNECT` kepada pengguna `rdsproxyadmin` sehingga pengguna dapat memantau koneksi pada basis data target.

```
GRANT CONNECT ON DATABASE postgres TO rdsproxyadmin;
```

Anda juga dapat mengizinkan pengguna basis data target lainnya untuk melakukan pemeriksaan kondisi dengan mengubah `rdsproxyadmin` ke pengguna basis data dalam perintah di atas.

Mengubah kata sandi untuk pengguna basis data

Dalam beberapa kasus, Anda dapat mengubah kata sandi untuk pengguna basis data dalam instans DB RDS yang terkait dengan proksi. Jika demikian, perbarui rahasia Secrets Manager yang sesuai dengan kata sandi baru.

Koneksi klien dan basis data

Koneksi dari aplikasi Anda ke Proksi RDS dikenal sebagai koneksi klien. Koneksi dari proxy ke basis data adalah koneksi basis data. Saat menggunakan Proksi RDS, koneksi klien berakhir di proksi sementara koneksi basis data dikelola dalam Proksi RDS.

Pengumpulan koneksi sisi aplikasi dapat memberikan manfaat untuk mengurangi pembuatan koneksi berulang antara aplikasi Anda dan Proksi RDS.

Pertimbangkan aspek konfigurasi berikut sebelum menerapkan kumpulan koneksi sisi aplikasi:

- Masa pakai maks koneksi klien: Proksi RDS menerapkan masa pakai maksimum koneksi klien selama 24 jam. Nilai ini tidak dapat dikonfigurasi. Konfigurasikan kumpulan Anda dengan masa pakai koneksi maksimum kurang dari 24 jam guna menghindari penurunan koneksi klien yang tidak terduga.
- Batas waktu idle koneksi klien: Proksi RDS menerapkan waktu idle maksimum untuk koneksi klien. Konfigurasikan kumpulan Anda dengan batas waktu koneksi idle dengan nilai yang lebih rendah dari pengaturan batas waktu idle koneksi klien untuk Proksi RDS guna menghindari penurunan koneksi yang tidak terduga.

Jumlah maksimum koneksi klien yang dikonfigurasi dalam kumpulan koneksi sisi aplikasi Anda tidak harus dibatasi pada pengaturan max_connections untuk Proksi RDS.

Pengumpulan koneksi klien menghasilkan masa pakai koneksi klien yang lebih lama. Jika koneksi Anda mengalami penyematan, pengumpulan koneksi klien dapat mengurangi efisiensi multiplexing. Koneksi klien yang disematkan tetapi idle dalam kumpulan koneksi sisi aplikasi terus berpegang pada koneksi basis data dan mencegah koneksi basis data digunakan kembali oleh koneksi klien lainnya. Tinjau log proksi untuk memeriksa apakah koneksi Anda mengalami penyematan.

Mengonfigurasi pengaturan koneksi

Untuk menyesuaikan pengumpulan koneksi Proksi RDS, Anda dapat mengubah pengaturan berikut:

- IdleClientTimeout
- MaxConnectionsPercent
- MaxIdleConnectionsPercent
- ConnectionBorrowTimeout

IdleClientTimeout

Anda dapat menentukan berapa lama koneksi klien bisa berada dalam status idle sebelum proksi menutupnya. Nilai default-nya adalah 1.800 detik (30 menit).

Koneksi klien dianggap idle jika aplikasi tidak mengirimkan permintaan baru dalam waktu yang ditentukan setelah permintaan sebelumnya selesai. Koneksi basis data yang mendasarinya akan tetap terbuka dan dikembalikan ke kumpulan koneksi. Oleh karena itu, koneksi dapat digunakan kembali untuk koneksi klien baru. Jika Anda ingin proksi secara proaktif menghapus koneksi yang sudah tidak terpakai, turunkan batas waktu koneksi klien yang idle. Jika beban kerja Anda sering terhubung dengan proksi, maka naikkan batas waktu koneksi klien yang idle untuk menghemat biaya pembangunan koneksi.

Pengaturan ini diwakili oleh bidang batas waktu koneksi klien Idle di konsol RDS dan `IdleClientTimeout` pengaturan di AWS CLI dan API. Untuk mempelajari cara mengubah nilai kolom Batas waktu koneksi klien idle di konsol RDS, lihat AWS Management Console. Untuk mempelajari cara mengubah nilai `IdleClientTimeout` setelan, lihat perintah CLI modify-db-proxyatau operasi API ModifyDBProxy.

MaxConnectionsPercent

Anda dapat membatasi jumlah koneksi yang dapat dibuat oleh Proksi RDS dengan basis data target. Anda menentukan batas dalam bentuk persentase koneksi maksimum yang tersedia untuk basis data Anda. Pengaturan ini diwakili oleh bidang koneksi maksimum Connection pool di konsol RDS dan `MaxConnectionsPercent` pengaturan di AWS CLI dan API.

Nilai `MaxConnectionsPercent` dinyatakan sebagai persentase dari pengaturan `max_connections` untuk klaster DB Aurora yang digunakan oleh grup target. Proksi tidak membuat semua koneksi ini di depan. Pengaturan ini memungkinkan proksi membuat koneksi ini karena beban kerja membutuhkannya.

Misalnya, untuk target basis data terdaftar dengan `max_connections` diatur ke 1000, dan `MaxConnectionsPercent` diatur ke 95, Proksi RDS menetapkan 950 koneksi sebagai batas atas koneksi bersamaan ke target basis data tersebut.

Efek samping umum beban kerja yang mencapai jumlah maksimum koneksi basis data yang diizinkan adalah peningkatan latensi kueri secara keseluruhan, disertai peningkatan metrik `DatabaseConnectionsBorrowLatency`. Anda dapat memantau koneksi basis data yang saat ini digunakan dan total koneksi basis data yang diizinkan dengan membandingkan metrik `DatabaseConnections` dan `MaxDatabaseConnectionsAllowed`.

Saat mengatur parameter ini, perhatikan praktik terbaik berikut:

- Izinkan headroom koneksi yang cukup untuk perubahan pola beban kerja. Sebaiknya atur parameter ini setidaknya 30% di atas penggunaan maksimum yang dipantau baru-baru ini. Karena Proksi RDS mendistribusikan ulang kuota koneksi basis data di beberapa simpul, perubahan kapasitas internal mungkin memerlukan setidaknya 30% headroom untuk koneksi tambahan guna menghindari peningkatan latensi pinjaman.
- Proksi RDS mencadangkan jumlah koneksi tertentu untuk pemantauan aktif guna mendukung failover cepat, perutean lalu lintas, dan operasi internal. Metrik `MaxDatabaseConnectionsAllowed` tidak mencakup koneksi yang dicadangkan ini. Metrik ini mewakili jumlah koneksi yang tersedia untuk melayani beban kerja, dan bisa lebih rendah dari nilai yang berasal dari pengaturan `MaxConnectionsPercent`.

  Nilai `MaxConnectionsPercent` minimum yang direkomendasikan adalah:

  - db.t3.small: 100
  - db.t3.medium: 55
  - db.t3.large: 35
  - db.r3.large atau lebih: 20

  Jika beberapa instans target didaftarkan di Proksi RDS seperti klaster Aurora dengan simpul pembaca, tetapkan nilai minimum berdasarkan instans terkecil yang terdaftar.

Untuk mempelajari cara mengubah nilai kolom Batas waktu maksimum kumpulan koneksi di konsol RDS, lihat AWS Management Console. Untuk mempelajari cara mengubah nilai `MaxConnectionsPercent` setelan, lihat perintah CLI modify-db-proxy-target-group atau operasi API ModifyDB. ProxyTargetGroup

> **Important**
>
> Jika klaster DB adalah bagian dari basis data global dengan penerusan tulis yang diaktifkan, kurangi nilai `MaxConnectionsPercent` proksi dengan kuota yang dialokasikan untuk penerusan tulis. Kuota penerusan tulis diatur dalam parameter klaster DB `aurora_fwd_writer_max_connections_pct`. Untuk informasi tentang penerusan tulis, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.

Untuk informasi tentang batas koneksi basis data, lihat Koneksi maksimum ke instans DB Aurora MySQL dan Koneksi maksimum ke instans DB Aurora PostgreSQL.

MaxIdleConnectionsPercent

Anda dapat mengontrol jumlah koneksi basis data idle yang dapat disimpan oleh Proksi RDS di kumpulan koneksi. Secara default, Proksi RDS menganggap koneksi basis data di kumpulannya menjadi idle jika tidak ada aktivitas pada koneksi selama lima menit.

Anda menentukan batas dalam bentuk persentase koneksi maksimum yang tersedia untuk basis data Anda. Nilai default-nya adalah 50 persen dari `MaxConnectionsPercent`, dan batas atasnya adalah nilai `MaxConnectionsPercent`. Dengan nilai tinggi, proksi membiarkan koneksi basis data idle dengan persentase yang tinggi tetap terbuka. Dengan nilai rendah, proksi menutup koneksi basis data idle dengan persentase yang tinggi. Jika beban kerja Anda tidak dapat diprediksi, pertimbangkan untuk mengatur nilai tinggi untuk `MaxIdleConnectionsPercent`. Jika Anda melakukannya, Proksi RDS dapat mengakomodasi lonjakan aktivitas tanpa membuka banyak koneksi basis data baru.

Pengaturan ini diwakili oleh `MaxIdleConnectionsPercent` pengaturan `DBProxyTargetGroup` di AWS CLI dan API. Untuk mempelajari cara mengubah nilai `MaxIdleConnectionsPercent` setelan, lihat perintah CLI modify-db-proxy-target-group atau operasi API ModifyDB. ProxyTargetGroup

Untuk informasi tentang batas koneksi basis data, lihat Koneksi maksimum ke instans DB Aurora MySQL dan Koneksi maksimum ke instans DB Aurora PostgreSQL.

ConnectionBorrowTimeout

Anda dapat memilih berapa lama Proksi RDS menunggu koneksi basis data dalam kumpulan koneksi menjadi tersedia untuk digunakan sebelum menampilkan eror batas waktu. Periode default-nya adalah 120 detik. Pengaturan ini berlaku saat jumlah koneksi mencapai titik maksimum, sehingga tidak ada koneksi yang tersedia dalam kumpulan koneksi. Pengaturan ini juga berlaku ketika tidak ada instans basis data yang tersedia untuk menangani permintaan, seperti saat operasi failover sedang berlangsung. Dengan pengaturan ini, Anda dapat mengatur periode tunggu terbaik untuk aplikasi Anda tanpa mengubah batas waktu kueri dalam kode aplikasi Anda.

Pengaturan ini diwakili oleh bidang batas waktu pinjam koneksi di konsol RDS atau `ConnectionBorrowTimeout` pengaturan `DBProxyTargetGroup` di API atau. AWS CLI Untuk mempelajari cara mengubah nilai kolom Batas waktu peminjaman koneksi di konsol RDS, lihat AWS Management Console. Untuk mempelajari cara mengubah nilai `ConnectionBorrowTimeout`

setelan, lihat perintah CLI modify-db-proxy-target-group atau operasi API ModifyDB. ProxyTargetGroup

Menghindari penyematan

Multiplexing akan lebih efisien saat permintaan basis data tidak bergantung pada informasi status dari permintaan sebelumnya. Dalam kasus ini, Proksi RDS dapat menggunakan kembali koneksi saat setiap transaksi selesai. Contoh informasi status tersebut mencakup sebagian besar variabel dan parameter konfigurasi yang dapat diubah melalui pernyataan `SET` atau `SELECT`. Secara default, transaksi SQL pada koneksi klien dapat bermultipleks antar-koneksi basis data acuan.

Koneksi ke proksi dapat memasukkan status yang disebut sebagai penyematan. Saat koneksi disematkan, setiap transaksi berikutnya akan menggunakan koneksi basis data acuan yang sama hingga sesi berakhir. Koneksi klien lainnya juga tidak dapat menggunakan kembali koneksi basis data tersebut hingga sesi berakhir. Sesi berakhir saat koneksi klien terputus.

Proksi RDS secara otomatis menyematkan koneksi klien ke koneksi DB tertentu saat mendeteksi perubahan sebuah status sesi yang tidak sesuai untuk sesi lainnya. Penyematan mengurangi efektivitas penggunaan kembali koneksi. Jika semua atau hampir semua koneksi Anda mengalami penyematan, pertimbangkan untuk mengubah kode aplikasi atau beban kerja untuk mengurangi kondisi yang menyebabkan penyematan.

Misalnya, aplikasi Anda mengubah variabel sesi atau parameter konfigurasi. Dalam hal ini, pernyataan selanjutnya dapat mengandalkan variabel atau parameter baru yang berlaku. Oleh karena itu, saat Proksi RDS memproses permintaan untuk mengubah variabel sesi atau pengaturan konfigurasi, sesi ini akan disematkan ke koneksi DB. Dengan demikian, tahap sesi tetap berfungsi untuk semua transaksi berikutnya dalam sesi yang sama.

Untuk beberapa mesin basis data, aturan ini tidak berlaku untuk semua parameter yang dapat Anda atur. Proksi RDS melacak pernyataan dan variabel tertentu. Oleh karena itu, Proksi RDS tidak menyematkan sesi saat Anda mengubahnya. Dalam kasus ini, Proksi RDS hanya menggunakan kembali koneksi untuk sesi lain yang memiliki nilai yang sama untuk pengaturan tersebut. Untuk daftar pernyataan dan variabel yang dilacak untuk Aurora MySQL, lihat Apa yang dilacak Proksi RDS untuk basis data Aurora MySQL.

Apa yang dilacak Proksi RDS untuk basis data Aurora MySQL

Berikut ini adalah pernyataan MySQL yang dilacak Proksi RDS:

- DROP DATABASE

- DROP SCHEMA
- USE

Berikut ini adalah variabel MySQL yang dilacak Proksi RDS:

- AUTOCOMMIT
- AUTO_INCREMENT_INCREMENT
- CHARACTER SET (or CHAR SET)
- CHARACTER_SET_CLIENT
- CHARACTER_SET_DATABASE
- CHARACTER_SET_FILESYSTEM
- CHARACTER_SET_CONNECTION
- CHARACTER_SET_RESULTS
- CHARACTER_SET_SERVER
- COLLATION_CONNECTION
- COLLATION_DATABASE
- COLLATION_SERVER
- INTERACTIVE_TIMEOUT
- NAMES
- NET_WRITE_TIMEOUT
- QUERY_CACHE_TYPE
- SESSION_TRACK_SCHEMA
- SQL_MODE
- TIME_ZONE
- TRANSACTION_ISOLATION (or TX_ISOLATION)
- TRANSACTION_READ_ONLY (or TX_READ_ONLY)
- WAIT_TIMEOUT

Meminimalkan penyematan

Penyetelan performa untuk Proksi RDS meliputi upaya memaksimalkan penggunaan kembali koneksi tingkat transaksi (multiplexing) dengan meminimalkan penyematan.

Anda dapat meminimalkan penyematan dengan melakukan hal berikut:

- Hindari permintaan basis data yang tidak perlu yang dapat menyebabkan penyematan.
- Atur variabel dan pengaturan konfigurasi secara konsisten di semua koneksi. Dengan demikian, sesi berikutnya cenderung menggunakan kembali koneksi yang memiliki pengaturan tertentu tersebut.

  Akan tetapi, untuk pengaturan PostgreSQL, sebuah variabel bisa menimbulkan penyematan sesi.
- Untuk basis data keluarga MySQL, terapkan sebuah filter penyematan sesi ke proksi. Anda dapat mengecualikan jenis operasi tertentu dari penyematan sesi jika Anda mengetahui bahwa tindakan ini tidak memengaruhi operasi yang benar aplikasi Anda.
- Lihat seberapa sering penyematan terjadi dengan memantau CloudWatch metrik `DatabaseConnectionsCurrentlySessionPinned` Amazon. Untuk informasi tentang ini dan CloudWatch metrik lainnya, lihatMemantau metrik Proxy RDS dengan Amazon CloudWatch.
- Jika menggunakan pernyataan `SET` untuk melakukan inisialisasi yang identik untuk setiap koneksi klien, Anda dapat melakukannya sekaligus mempertahankan multiplexing tingkat transaksi. Dalam kasus ini, Anda memindahkan pernyataan yang menyiapkan status sesi awal ke dalam kueri inisialisasi yang digunakan oleh proksi. Properti ini adalah string yang berisi satu atau beberapa pernyataan SQL, yang dipisahkan oleh titik koma.

  Misalnya, Anda dapat menentukan kueri inisialisasi untuk proksi yang menetapkan parameter konfigurasi tertentu. Kemudian, Proksi RDS menerapkan pengaturan tersebut setiap kali koneksi baru untuk proksi itu disiapkan. Anda dapat menghapus pernyataan `SET` yang sesuai dari kode aplikasi, sehingga tidak mengganggu multiplexing tingkat transaksi.

  Untuk metrik tentang seberapa sering penyematan terjadi pada proksi, lihat Memantau metrik Proxy RDS dengan Amazon CloudWatch.

Kondisi yang menyebabkan penyematan untuk semua keluarga mesin

Proksi menyematkan sesi ke koneksi saat ini dalam situasi berikut ketika multiplexing dapat menyebabkan perilaku yang tidak terduga:

- Pernyataan apa pun dengan ukuran teks lebih dari 16 KB bisa menyebabkan proksi menyematkan sesi.

Kondisi yang menyebabkan penyematan untuk Aurora MySQL

Untuk MySQL, interaksi berikut juga menyebabkan penyematan:

- Pernyataan kunci tabel eksplisit `LOCK TABLE`, `LOCK TABLES`, atau `FLUSH TABLES WITH READ LOCK` menyebabkan proksi menyematkan sesi.
- Membuat kunci bernama dengan menggunakan `GET_LOCK` menyebabkan proksi menyematkan sesi.
- Menetapkan variabel pengguna atau variabel sistem (dengan beberapa pengecualian) menyebabkan proksi menyematkan sesi. Jika situasi ini terlalu banyak mengurangi frekuensi penggunaan kembali koneksi Anda, pilih operasi `SET` agar tidak menyebabkan penyematan. Untuk informasi tentang cara melakukannya dengan mengatur properti filter penyematan sesi, lihat Membuat Proksi RDS dan Mengubah Proksi RDS.
- Membuat tabel sementara menyebabkan proksi menyematkan sesi. Dengan begitu, konten tabel sementara dipertahankan sepanjang sesi, terlepas dari batasan transaksi.
- Memanggil fungsi `ROW_COUNT`, `FOUND_ROWS`, dan `LAST_INSERT_ID` terkadang menyebabkan penyematan.

  Keadaan ketika fungsi ini menyebabkan penyematan mungkin berbeda antara versi Aurora MySQL yang kompatibel dengan MySQL 5.7.
- Pernyataan yang disiapkan menyebabkan proksi menyematkan sesi. Aturan ini berlaku terlepas dari apakah pernyataan yang disiapkan menggunakan teks SQL maupun protokol biner.
- Proksi RDS tidak menyematkan koneksi saat Anda menggunakan SET LOCAL.
- Memanggil prosedur tersimpan dan fungsi tersimpan tidak menyebabkan pinning. Proksi RDS tidak mendeteksi perubahan status sesi apa pun yang terjadi akibat perintah tersebut. Pastikan aplikasi Anda tidak mengubah status sesi di dalam rutinitas tersimpan jika Anda mengandalkan status sesi tersebut untuk bertahan di seluruh transaksi. Misalnya, Proxy RDS saat ini tidak kompatibel dengan prosedur tersimpan yang membuat tabel sementara yang bertahan di semua transaksi.

Jika memiliki pengetahuan mendalam tentang perilaku aplikasi, Anda dapat menangani perilaku penyematan untuk pernyataan aplikasi tertentu. Untuk melakukannya, pilih opsi Filter penyematan sesi saat membuat proksi. Saat ini, Anda dapat memilih untuk tidak menggunakan penyematan sesi untuk pengaturan variabel sesi dan pengaturan konfigurasi.

Kondisi yang menyebabkan penyematan untuk Aurora PostgreSQL

Untuk PostgreSQL, interaksi berikut juga menyebabkan penyematan:

- Menggunakan `SET` perintah.
- Menggunakan`PREPARE`,`DISCARD`,`DEALLOCATE`, atau `EXECUTE` perintah untuk mengelola pernyataan yang disiapkan.
- Membuat urutan sementara, tabel, atau tampilan.
- Mendeklarasikan kursor.
- Membuang status sesi.
- Mendengarkan di saluran notifikasi.
- Memuat modul perpustakaan seperti`auto_explain`.
- Memanipulasi urutan menggunakan fungsi seperti `nextval` dan. `setval`
- Berinteraksi dengan kunci menggunakan fungsi seperti `pg_advisory_lock` dan`pg_try_advisory_lock`.
- Mengatur parameter, atau mengatur ulang parameter ke defaultnya. Secara khusus, menggunakan `SET` dan `set_config` perintah untuk menetapkan nilai default ke variabel sesi.
- Memanggil prosedur tersimpan dan fungsi tersimpan tidak menyebabkan pinning. Proksi RDS tidak mendeteksi perubahan status sesi apa pun yang terjadi akibat perintah tersebut. Pastikan aplikasi Anda tidak mengubah status sesi di dalam rutinitas tersimpan jika Anda mengandalkan status sesi tersebut untuk bertahan di seluruh transaksi. Misalnya, Proxy RDS saat ini tidak kompatibel dengan prosedur tersimpan yang membuat tabel sementara yang bertahan di semua transaksi.

Menghapus Proksi RDS

Anda dapat menghapus proksi saat tidak lagi membutuhkannya. Atau Anda dapat menghapus proksi jika Anda merasa instans atau klaster DB yang terkait dengannya sedang dalam perbaikan.

AWS Management Console

Untuk menghapus proksi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Proksi.
3. Pilih proksi yang akan dihapus dari daftar.

4. Pilih Hapus Proksi.

AWS CLI

Untuk menghapus proxy DB, gunakan AWS CLI perintah delete-db-proxy. Untuk menghapus asosiasi terkait, gunakan juga deregister-db-proxy-targetsperintah.

```
aws rds delete-db-proxy --name proxy_name
```

```
aws rds deregister-db-proxy-targets
    --db-proxy-name proxy_name
    [--target-group-name target_group_name]
    [--target-ids comma_separated_list]       # or
    [--db-instance-identifiers instance_id]       # or
    [--db-cluster-identifiers cluster_id]
```

API RDS

Untuk menghapus proksi DB, panggil fungsi Amazon RDS API DeleteDBProxy. Untuk menghapus item dan asosiasi terkait, Anda juga memanggil fungsi DeleteDB ProxyTargetGroup dan DeregisterDB. ProxyTargets

Bekerja dengan titik akhir Proksi Amazon RDS

Pelajari titik akhir untuk Proksi RDS dan cara menggunakannya. Dengan titik akhir proksi, Anda dapat memanfaatkan kemampuan berikut:

- Anda dapat menggunakan beberapa titik akhir dengan proksi untuk memantau dan memecahkan masalah koneksi dari aplikasi yang berbeda secara independen.
- Anda dapat menggunakan titik akhir pembaca dengan klaster DB Aurora untuk meningkatkan skalabilitas baca dan ketersediaan tinggi untuk aplikasi Anda yang memiliki banyak kueri.
- Anda dapat menggunakan titik akhir lintas-VPC untuk mengizinkan akses ke basis data dalam satu VPC dari sumber daya seperti instans Amazon EC2 di VPC yang berbeda.

Topik

- Ikhtisar titik akhir proksi
- Menggunakan titik akhir pembaca dengan klaster Aurora

- Mengakses basis data Aurora di seluruh VPC
- Membuat titik akhir proksi
- Melihat titik akhir proksi
- Mengubah titik akhir proksi
- Menghapus titik akhir proksi
- Batasan untuk titik akhir proksi

Ikhtisar titik akhir proksi

Bekerja dengan titik akhir Proksi RDS melibatkan jenis prosedur yang sama seperti dengan klaster DB Aurora dan titik akhir pembaca. Jika Anda belum terbiasa dengan titik akhir Aurora, temukan informasi selengkapnya di Manajemen koneksi Amazon Aurora.

Secara default, titik akhir yang Anda hubungkan saat Anda menggunakan Proksi RDS dengan klaster Aurora memiliki kapabilitas baca/tulis. Akibatnya, titik akhir ini mengirimkan semua permintaan ke instans penulis klaster. Semua koneksi tersebut dihitung terhadap nilai `max_connections` untuk instans penulis. Jika proksi Anda dikaitkan dengan klaster DB Aurora, Anda dapat membuat titik akhir baca/tulis atau titik akhir hanya-baca tambahan untuk proksi itu.

Anda dapat menggunakan titik akhir hanya baca dengan proksi Anda untuk kueri hanya baca. Anda dapat melakukannya seperti saat Anda menggunakan titik akhir pembaca untuk klaster yang disediakan Aurora. Tindakan ini membantu Anda untuk memanfaatkan skalabilitas baca klaster Aurora dengan satu atau beberapa instans DB pembaca. Anda dapat menjalankan lebih banyak kueri simultan dan membuat lebih banyak koneksi simultan dengan menggunakan titik akhir hanya-baca dan menambahkan lebih banyak instans DB pembaca untuk klaster Aurora Anda sesuai kebutuhan.

> **Tip**
>
> Ketika membuat proksi untuk klaster Aurora menggunakan AWS Management Console, Anda dapat memaksa Proksi RDS membuat titik akhir pembaca secara otomatis. Untuk informasi tentang manfaat titik akhir pembaca, lihat Menggunakan titik akhir pembaca dengan klaster Aurora.

Untuk titik akhir proksi yang Anda buat, Anda juga dapat mengaitkan titik akhir tersebut dengan cloud privat virtual (VPC) yang berbeda dengan yang digunakan oleh proksi itu sendiri. Dengan demikian,

Anda dapat terhubung ke proksi dari VPC yang berbeda, misalnya VPC yang digunakan oleh aplikasi lain dalam organisasi Anda.

Untuk informasi tentang batas yang terkait dengan titik akhir proksi, lihat Batasan untuk titik akhir proksi.

Dalam log Proksi RDS, setiap entri diawali dengan nama titik akhir proksi terkait. Nama ini bisa berupa nama yang Anda tentukan untuk titik akhir yang ditentukan pengguna. Atau, bisa menjadi nama khusus `default` untuk titik akhir default proksi yang melakukan permintaan baca/tulis.

Setiap titik akhir proxy memiliki kumpulan CloudWatch metriknya sendiri. Anda dapat memantau metrik untuk semua titik akhir proksi. Anda juga dapat memantau metrik untuk titik akhir tertentu, atau untuk semua titik akhir baca/tulis atau hanya-baca dari proksi. Untuk informasi selengkapnya, lihat Memantau metrik Proxy RDS dengan Amazon CloudWatch.

Titik akhir proksi menggunakan mekanisme autentikasi yang sama seperti proksi yang terkait. Proksi RDS secara otomatis menyiapkan izin dan otorisasi untuk titik akhir yang ditentukan pengguna, yang konsisten dengan properti proksi terkait.

Untuk mempelajari cara kerja titik akhir proksi untuk klaster DB dalam Aurora global basis data, lihat Cara kerja titik akhir Proksi RDS dengan basis data global.

Menggunakan titik akhir pembaca dengan klaster Aurora

Anda dapat membuat dan menyambung ke titik akhir hanya-baca yang disebut titik akhir pembaca jika Anda menggunakan Proksi RDS dengan klaster Aurora. Titik akhir pembaca ini membantu meningkatkan skalabilitas baca aplikasi padat kueri. Titik akhir pembaca juga membantu meningkatkan ketersediaan koneksi Anda jika instans DB pembaca di klaster Anda menjadi tidak tersedia.

> **Note**
>
> Jika Anda menentukan bahwa titik akhir yang baru adalah titik akhir hanya-baca, Proksi RDS mengharuskan klaster Aurora memiliki satu atau beberapa instans DB pembaca. Dalam beberapa kasus, Anda dapat mengubah target proksi ke klaster Aurora yang hanya berisi klaster Aurora penulis tunggal atau multi-penulis. Jika Anda melakukannya, permintaan apa pun ke titik akhir pembaca akan gagal dengan kesalahan. Permintaan juga gagal jika target proksi adalah instans RDS, bukan klaster Aurora.
> Jika klaster Aurora memiliki instans pembaca, tetapi instans tersebut tidak tersedia, Proksi RDS akan menunggu untuk mengirim permintaan, bukan serta-merta menampilkan

kesalahan. Jika tidak ada instans pembaca yang tersedia dalam periode batas waktu peminjaman koneksi, permintaan gagal akan dengan kesalahan.

Cara titik akhir pembaca membantu ketersediaan aplikasi

Dalam beberapa kasus, satu atau beberapa instans pembaca dalam klaster Anda mungkin menjadi tidak tersedia. Jika demikian, koneksi yang menggunakan titik akhir pembaca proksi DB dapat pulih lebih cepat daripada koneksi yang menggunakan titik akhir pembaca Aurora. Proksi RDS hanya merutekan koneksi ke instans pembaca yang tersedia dalam klaster. Tidak ada penundaan dikarenakan caching DNS saat instans menjadi tidak tersedia.

Jika koneksi di-multipleks, Proksi RDS mengarahkan kueri berikutnya ke instans DB pembaca yang berbeda tanpa mengganggu aplikasi. Selama switchover otomatis ke instans pembaca baru, Proksi RDS memeriksa keterlambatan replikasi instans pembaca lama dan baru. Proksi RDS memastikan bahwa instans pembaca baru diperbarui dengan perubahan yang sama seperti instans pembaca sebelumnya. Dengan begitu, aplikasi Anda tidak pernah melihat data usang ketika Proksi RDS beralih dari satu instans DB pembaca ke instans DB pembaca lainnya.

Jika koneksi disematkan, kueri berikutnya pada koneksi akan menampilkan kesalahan. Namun, aplikasi Anda dapat langsung terhubung kembali ke titik akhir yang sama. Proksi RDS merutekan koneksi ke instans DB pembaca yang berbeda yang ada dalam status `available`. Ketika Anda terhubung kembali secara manual, Proksi RDS tidak memeriksa keterlambatan replikasi antara instans pembaca lama dan baru.

Jika klaster Aurora Anda tidak memiliki instans pembaca yang tersedia, Proksi RDS akan memeriksa apakah kondisi ini sementara atau permanen. Perilaku dalam setiap kasusnya adalah sebagai berikut:

- Misalkan klaster Anda memiliki satu atau beberapa instans DB pembaca, tetapi tidak satu pun berada dalam status `Available`. Misalnya, semua instans pembaca mungkin di-reboot atau menghadapi masalah. Dalam hal ini, upaya untuk terhubung ke titik akhir pembaca menunggu instans pembaca menjadi tersedia. Jika tidak ada instans pembaca yang tersedia dalam periode batas waktu peminjaman koneksi, upaya koneksi akan gagal. Jika instans pembaca menjadi tersedia, upaya koneksi berhasil.
- Misalkan klaster Anda tidak memiliki instans DB pembaca. Dalam hal ini, Proksi RDS segera menampilkan kesalahan jika Anda mencoba untuk terhubung ke titik akhir pembaca. Untuk

mengatasi masalah ini, tambahkan satu atau beberapa instans pembaca ke klaster Anda sebelum terhubung ke titik akhir pembaca.

Cara titik akhir pembaca membantu skalabilitas kueri

Titik akhir pembaca untuk proksi membantu skalabilitas kueri Aurora dengan cara berikut:

- Saat Anda menambahkan instans pembaca ke klaster Aurora, Proksi RDS dapat merutekan koneksi baru ke setiap titik akhir pembaca ke instans pembaca yang berbeda. Dengan begitu, kueri yang dilakukan dengan menggunakan satu koneksi titik akhir pembaca tidak memperlambat kueri yang dilakukan menggunakan koneksi titik akhir pembaca lain. Kueri berjalan pada instans DB terpisah. Setiap instans DB memiliki sumber daya komputasinya sendiri, buffer cache, dan sebagainya.
- Jika praktis, Proksi RDS menggunakan instans DB pembaca yang sama untuk semua masalah kueri yang menggunakan koneksi titik akhir pembaca tertentu. Dengan demikian, satu kumpulan kueri terkait pada tabel yang sama dapat memanfaatkan caching, optimasi rencana, dan sebagainya, pada instans DB tertentu.
- Jika instans DB pembaca tidak tersedia, pengaruh pada aplikasi Anda bergantung pada apakah sesi di-multipleks atau disematkan. Jika sesi di-multipleks, Proksi RDS merutekan setiap kueri berikutnya untuk instans DB pembaca yang berbeda tanpa campur tangan Anda. Jika sesi disematkan, aplikasi Anda mengalami kesalahan dan harus dihubungkan kembali. Anda dapat segera terhubung kembali ke titik akhir pembaca dan Proksi RDS merutekan koneksi ke instans DB pembaca yang tersedia. Untuk informasi selengkapnya tentang cara memultipleks dan menyematkan untuk sesi proksi, lihat Ikhtisar konsep Proksi RDS.
- Semakin banyak instans DB pembaca yang Anda miliki dalam klaster, semakin banyak koneksi simultan yang dapat Anda buat menggunakan titik akhir pembaca. Misalnya, anggaplah bahwa klaster Anda memiliki empat instans DB pembaca, masing-masing dikonfigurasi untuk mendukung 200 koneksi simultan. Misalkan proksi Anda dikonfigurasi untuk menggunakan 50% dari koneksi maksimum. Di sini, jumlah maksimum koneksi yang dapat Anda buat melalui titik akhir pembaca di proksi ini adalah 100 (50% dari 200) untuk pembaca 1. Juga 100 koneksi untuk pembaca 2, dan seterusnya, dengan total 400. Jika Anda menggandakan jumlah instans DB pembaca klaster menjadi delapan, maka jumlah maksimum koneksi melalui titik akhir pembaca juga berlipat ganda, menjadi 800.

Contoh penggunaan titik akhir pembaca

Contoh Linux berikut menunjukkan cara mengonfirmasi bahwa Anda terhubung ke klaster Aurora MySQL melalui titik akhir pembaca. Pengaturan konfigurasi `innodb_read_only` diaktifkan. Upaya untuk melakukan operasi tulis seperti pernyataan `CREATE DATABASE` gagal dengan kesalahan. Dan Anda dapat mengonfirmasi bahwa Anda terhubung ke instans DB pembaca dengan memeriksa nama instans DB menggunakan variabel `aurora_server_id`.

> **Tip**
>
> Jangan hanya mengandalkan pemeriksaan nama instans DB untuk menentukan apakah koneksi itu baca/tulis atau hanya-baca. Ingat bahwa instans DB dalam klaster Aurora dapat mengubah peran antara penulis dan pembaca ketika failover terjadi.

```
$ mysql -h endpoint-demo-reader.endpoint.proxy-demo.us-east-1.rds.amazonaws.com -u
 admin -p
...
mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+
mysql> create database shouldnt_work;
ERROR 1290 (HY000): The MySQL server is running with the --read-only option so it
 cannot execute this statement

mysql> select @@aurora_server_id;
+--------------------------------------+
| @@aurora_server_id                   |
+--------------------------------------+
| proxy-reader-endpoint-demo-instance-3 |
+--------------------------------------+
```

Contoh berikut menunjukkan bagaimana koneksi Anda ke titik akhir pembaca proksi dapat tetap bekerja bahkan ketika instans DB pembaca dihapus. Dalam contoh ini, klaster Aurora memiliki dua instans pembaca, `instance-5507` dan `instance-7448`. Koneksi ke titik akhir pembaca mulai menggunakan salah satu instans pembaca. Dalam contoh, instans pembaca ini dihapus oleh

perintah `delete-db-instance`. Proksi RDS beralih ke instans pembaca yang berbeda untuk kueri berikutnya.

```
$ mysql -h reader-demo.endpoint.proxy-demo.us-east-1.rds.amazonaws.com
  -u my_user -p
...
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-5507      |
+--------------------+

mysql> select @@innodb_read_only;
+--------------------+
| @@innodb_read_only |
+--------------------+
|                  1 |
+--------------------+

mysql> select count(*) from information_schema.tables;
+----------+
| count(*) |
+----------+
|      328 |
+----------+
```

Ketika sesi `mysql` masih berjalan, perintah berikut menghapus instans pembaca yang terhubung ke titik akhir pembaca.

```
aws rds delete-db-instance --db-instance-identifier instance-5507 --skip-final-snapshot
```

Kueri dalam sesi `mysql` terus bekerja tanpa perlu menghubungkan kembali. Proksi RDS beralih otomatis ke instans DB pembaca yang berbeda.

```
mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-7448      |
+--------------------+
```

```
mysql> select count(*) from information_schema.TABLES;
+----------+
| count(*) |
+----------+
|      328 |
+----------+
```

Mengakses basis data Aurora di seluruh VPC

Secara default, semua komponen tumpukan teknologi Aurora Anda berada dalam Amazon VPC yang sama. Misalnya, anggaplah bahwa aplikasi yang berjalan di instans Amazon EC2 terhubung ke klaster DB Aurora. Dalam hal ini, server aplikasi dan basis data keduanya harus berada dalam VPC yang sama.

Dengan Proksi RDS, Anda dapat menyiapkan akses ke klaster DB Aurora di satu VPC dari sumber daya di VPC lain, seperti instans EC2. Misalnya, organisasi Anda mungkin memiliki beberapa aplikasi yang mengakses sumber daya basis data yang sama. Setiap aplikasi mungkin berada dalam VPC-nya sendiri.

Untuk mengaktifkan akses lintas-VPC, Anda membuat titik akhir baru untuk proksi tersebut. Proksi itu sendiri berada di VPC yang sama dengan klaster DB Aurora. Namun, titik akhir lintas-VPC berada di VPC lain, bersama dengan sumber daya lain seperti instans EC2. Titik akhir lintas-VPC dikaitkan dengan subnet dan grup keamanan dari VPC yang sama sebagai EC2 dan sumber daya lainnya. Pengaitan ini memungkinkan Anda terhubung ke titik akhir dari aplikasi yang jika sebaliknya tidak dapat mengakses basis data dikarenakan pembatasan VPC.

Langkah-langkah berikut menjelaskan cara membuat dan mengakses titik akhir lintas-VPC melalui Proksi RDS:

1. Buat dua VPC, atau pilih dua VPC yang sudah Anda gunakan untuk pekerjaan Aurora . Setiap VPC harus memiliki sumber daya jaringan yang terkaitnya sendiri seperti gateway internet, tabel rute, subnet, dan grup keamanan. Jika Anda hanya memiliki satu VPC, Anda dapat melihat Memulai dengan Amazon Aurora untuk mengetahui langkah-langkah penyiapan VPC lain agar berhasil menggunakan Aurora. Anda juga dapat memeriksa VPC yang ada di konsol Amazon EC2 untuk melihat jenis sumber daya yang dapat dihubungkan bersama.
2. Buat proksi DB yang terkait dengan klaster DB Aurora yang ingin Anda hubungkan. Ikuti prosedur di Membuat Proksi RDS.
3. Pada halaman Detail untuk proksi Anda di konsol RDS, di bagian Titik akhir proksi, pilih Buat titik akhir. Ikuti prosedur di Membuat titik akhir proksi.

4. Pilih apakah lintas-titik akhir VPC akan bersifat baca/tulis atau hanya-baca.
5. Alih-alih menerima pengaturan default VPC yang sama dengan klaster DB Aurora, pilih VPC yang berbeda. VPC ini harus berada di Wilayah AWS yang sama dengan VPC tempat proksi berada.
6. Sekarang, alih-alih menerima pengaturan default untuk subnet dan grup keamanan dari VPC yang sama dengan klaster DB Aurora, buatlah pilihan baru. Buat pilihan ini berdasarkan subnet dan grup keamanan dari VPC yang Anda pilih.
7. Anda tidak perlu mengubah pengaturan apa pun untuk rahasia Secrets Manager. Kredensial yang sama dapat digunakan untuk semua titik akhir proksi Anda, terlepas dari VPC tempat setiap titik akhir berada.
8. Tunggu sampai titik akhir baru untuk mencapai status Tersedia.
9. Buat catatan nama titik akhir lengkap. Ini adalah nilai yang berakhiran *`Region_name`*`.rds.amazonaws.com` yang Anda berikan sebagai bagian dari string koneksi untuk aplikasi basis data Anda.
10. Akses titik akhir baru dari sumber daya di VPC yang sama dengan titik akhir. Cara mudah untuk menguji proses ini adalah membuat instans EC2 baru di VPC ini. Kemudian, masuk ke instans EC2 dan jalankan perintah `mysql` atau `psql` untuk terhubung dengan menggunakan nilai titik akhir dalam string koneksi Anda.

Membuat titik akhir proksi

Konsol

Untuk membuat titik akhir proksi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Proksi.
3. Klik nama proksi yang titik akhir barunya ingin dibuat.

   Halaman detail untuk proksi tersebut muncul.
4. Di bagian Titik akhir proksi, pilih Buat titik akhir proksi.

   Jendela Buat titik akhir proksi akan muncul.
5. Untuk Nama titik akhir proksi, masukkan nama deskriptif pilihan Anda.

6. Untuk Peran target, pilih apakah akan membuat titik akhir lintas-VPC bersifat baca/tulis atau hanya-baca.

   Koneksi yang menggunakan titik akhir baca/tulis dapat melakukan segala jenis operasi, seperti pernyataan bahasa definisi data (DDL), pernyataan bahasa manipulasi data (DML), dan kueri. Titik akhir ini terhubung ke instans primer klaster Aurora. Anda dapat menggunakan titik akhir baca/tulis untuk operasi basis data umum jika Anda hanya menggunakan titik akhir tunggal dalam aplikasi Anda. Anda juga dapat menggunakan endpoint baca/tulis untuk operasi administratif, aplikasi pemrosesan transaksi online (OLTP), dan extract-transform-load (ETL) pekerjaan.

   Koneksi yang menggunakan titik akhir hanya-baca hanya dapat melakukan kueri. Jika ada beberapa instans pembaca dalam klaster Aurora, Proksi RDS dapat menggunakan instans pembaca yang berbeda untuk setiap koneksi ke titik akhir. Dengan begitu, aplikasi intensif kueri dapat memanfaatkan kemampuan pengklasteran Aurora. Anda dapat menambahkan lebih banyak kapasitas kueri ke klaster dengan menambahkan lebih banyak instans pembaca DB. Koneksi hanya-baca ini tidak membebankan overhead apa pun pada instans primer klaster. Dengan begitu, kueri pelaporan dan analisis Anda tidak akan memperlambat operasi menulis aplikasi OLTP Anda.

7. Untuk Cloud Privat Virtual (VPC), pilih default untuk mengakses titik akhir dari instans EC2 yang sama atau sumber daya lain yang biasanya digunakan untuk mengakses proksi atau basis data terkaitnya. Untuk menyiapkan akses lintas-VPC untuk proksi ini, pilih VPC selain default. Untuk informasi selengkapnya tentang akses lintas-VPC, lihat Mengakses basis data Aurora di seluruh VPC.

8. Untuk Subnet, Proksi RDS mengisi subnet yang sama dengan proksi terkait secara default. Untuk membatasi akses ke titik akhir agar hanya sebagian dari rentang alamat VPC dapat terhubung, hapus satu subnet atau lebih.

9. Untuk Grup keamanan VPC, Anda dapat memilih grup keamanan yang sudah ada atau membuat grup keamanan baru. Proksi RDS mengisi grup keamanan atau beberapa grup keamanan yang sama sebagai proksi terkait secara default. Jika aturan masuk dan keluar proksi sesuai untuk titik akhir ini, pertahankan pilihan default.

   Jika Anda memilih untuk membuat grup keamanan baru, tentukan nama grup keamanan tersebut di halaman ini. Kemudian, edit pengaturan grup keamanan dari konsol EC2 nanti.

10. Pilih Buat titik akhir proksi.

AWS CLI

Untuk membuat titik akhir proxy, gunakan AWS CLI create-db-proxy-endpointperintah.

Sertakan parameter-parameter yang diperlukan berikut:

- `--db-proxy-name` *`value`*
- `--db-proxy-endpoint-name` *`value`*
- `--vpc-subnet-ids` *`list_of_ids`*. Pisahkan ID subnet dengan spasi. Anda tidak menentukan ID dari VPC itu sendiri.

Anda juga dapat menentukan parameter opsional berikut:

- `--target-role { READ_WRITE | READ_ONLY }`. Pengaturan default parameter ini adalah `READ_WRITE`. Nilai `READ_ONLY` hanya mempengaruhi klaster yang disediakan Aurora yang berisi satu atau beberapa instans DB pembaca. Ketika proksi dikaitkan dengan klaster Aurora yang hanya berisi instans DB penulis, Anda tidak dapat menentukan `READ_ONLY`. Untuk informasi selengkapnya tentang tujuan penggunaan titik akhir hanya-baca dengan klaster Aurora, lihat Menggunakan titik akhir pembaca dengan klaster Aurora .
- `--vpc-security-group-ids` *`value`*. Pisahkan ID grup keamanan dengan spasi. Jika Anda menghilangkan parameter ini, Proksi RDS menggunakan grup keamanan default untuk VPC. Proksi RDS menentukan VPC berdasarkan ID subnet yang Anda tentukan untuk parameter `--vpc-subnet-ids`.

Example

Contoh berikut membuat titik akhir proksi bernama `my-endpoint`.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-proxy-endpoint \
  --db-proxy-name my-proxy \
  --db-proxy-endpoint-name my-endpoint \
  --vpc-subnet-ids subnet_id subnet_id subnet_id ... \
  --target-role READ_ONLY \
  --vpc-security-group-ids security_group_id ]
```

Untuk Windows:

```
aws rds create-db-proxy-endpoint ^
  --db-proxy-name my-proxy ^
  --db-proxy-endpoint-name my-endpoint ^
  --vpc-subnet-ids subnet_id_1 subnet_id_2 subnet_id_3 ... ^
  --target-role READ_ONLY ^
  --vpc-security-group-ids security_group_id
```

API RDS

Untuk membuat titik akhir proxy, gunakan tindakan ProxyEndpointCreateDB API RDS.

Melihat titik akhir proksi

Konsol

Untuk melihat detail titik akhir proksi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Proksi.
3. Dalam daftar, pilih proksi yang titik akhirnya ingin dilihat. Klik nama proksi untuk melihat halaman detailnya.
4. Di bagian Titik akhir proksi, pilih titik akhir yang ingin dilihat. Klik namanya untuk melihat halaman detailnya.
5. Periksa parameter yang nilainya Anda minati. Anda dapat mengubah properti seperti berikut ini:
   - Apakah titik akhir ini baca/tulis atau hanya-baca.
   - Alamat titik akhir yang Anda gunakan dalam string koneksi basis data.
   - VPC, subnet, dan grup keamanan yang terkait dengan titik akhir.

AWS CLI

Untuk melihat satu atau beberapa titik akhir proxy, gunakan AWS CLI describe-db-proxy-endpointsperintah.

Anda dapat menyertakan parameter opsional berikut:

- `--db-proxy-endpoint-name`

- `--db-proxy-name`

Contoh berikut menjelaskan titik akhir proksi `my-endpoint`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-proxy-endpoints \
  --db-proxy-endpoint-name my-endpoint
```

Untuk Windows:

```
aws rds describe-db-proxy-endpoints ^
  --db-proxy-endpoint-name my-endpoint
```

API RDS

Untuk mendeskripsikan satu atau beberapa titik akhir proxy, gunakan operasi RDS API ProxyEndpointsDescribeDB.

Mengubah titik akhir proksi

Konsol

Untuk mengubah satu atau beberapa titik akhir proksi

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Proksi.
3. Dalam daftar, pilih proksi yang titik akhirnya ingin diubah. Klik nama proksi untuk melihatnya
4. Di bagian Titik akhir proksi, pilih titik akhir yang ingin diubah. Anda dapat memilihnya dalam daftar, atau mengeklik namanya untuk melihat halaman detail.
5. Di halaman detail proksi, di bagian Titik akhir proksi, pilih Edit. Atau, di halaman detail titik akhir proksi, untuk Tindakan, pilih Edit.
6. Ubah nilai parameter yang ingin dimodifikasi.

7. Pilih Simpan perubahan.

AWS CLI

Untuk memodifikasi titik akhir proxy, gunakan AWS CLI modify-db-proxy-endpointperintah dengan parameter yang diperlukan berikut:

- `--db-proxy-endpoint-name`

Tentukan perubahan ke properti titik akhir dengan menggunakan satu atau beberapa parameter berikut:

- `--new-db-proxy-endpoint-name`
- `--vpc-security-group-ids`. Pisahkan ID grup keamanan dengan spasi.

Contoh berikut mengganti nama titik akhir proksi `my-endpoint` menjadi `new-endpoint-name`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint \
  --new-db-proxy-endpoint-name new-endpoint-name
```

Untuk Windows:

```
aws rds modify-db-proxy-endpoint ^
  --db-proxy-endpoint-name my-endpoint ^
  --new-db-proxy-endpoint-name new-endpoint-name
```

API RDS

Untuk memodifikasi titik akhir proxy, gunakan operasi RDS API ProxyEndpointModifyDB.

Menghapus titik akhir proksi

Anda dapat menghapus titik akhir proksi menggunakan konsol seperti yang dijelaskan berikut ini.

> **Note**
>
> Anda tidak dapat menghapus titik akhir proksi default yang dibuat secara otomatis oleh Proksi RDS untuk setiap proksi.
> Saat Anda menghapus proksi, Proksi RDS secara otomatis menghapus semua titik akhir terkait.

Konsol

Untuk menghapus titik akhir proksi menggunakan AWS Management Console

1. Di panel navigasi, pilih Proksi.
2. Dalam daftar, pilih proksi yang titik akhirnya ingin dihapus. Klik nama proksi untuk melihat halaman detailnya.
3. Di bagian Titik akhir proksi, pilih titik akhir yang ingin dihapus. Anda dapat memilih satu atau beberapa titik akhir dalam daftar, atau mengeklik nama titik akhir tunggal untuk melihat halaman detail.
4. Di halaman detail proksi, di bagian Titik akhir proksi, pilih Hapus. Atau, di halaman detail titik akhir proksi, untuk Tindakan, pilih Hapus.

AWS CLI

Untuk menghapus titik akhir proxy, jalankan delete-db-proxy-endpointperintah dengan parameter yang diperlukan berikut:

- `--db-proxy-endpoint-name`

Perintah berikut akan menghapus titik akhir proksi bernama `my-endpoint`.

Untuk Linux, macOS, atau Unix:

```
aws rds delete-db-proxy-endpoint \
  --db-proxy-endpoint-name my-endpoint
```

Untuk Windows:

```
aws rds delete-db-proxy-endpoint ^
```

```
  --db-proxy-endpoint-name my-endpoint
```

API RDS

Untuk menghapus titik akhir proxy dengan RDS API, jalankan operasi ProxyEndpointDeleteDB. Tentukan nama titik akhir proksi untuk parameter `DBProxyEndpointName`.

Batasan untuk titik akhir proksi

Titik akhir Proksi RDS memiliki batasan berikut:

- Setiap proksi memiliki titik akhir default yang dapat dimodifikasi, tetapi tidak dapat dibuat atau dihapus.
- Jumlah maksimum titik akhir yang ditentukan pengguna untuk proksi adalah 20. Dengan demikian, proksi dapat memiliki hingga 21 titik akhir: titik akhir default, ditambah 20 titik akhir yang Anda buat.
- Jika Anda mengaitkan titik akhir tambahan dengan proksi, Proksi RDS secara otomatis menentukan instans DB dalam klaster Anda yang digunakan untuk setiap titik akhir. Anda tidak dapat memilih instans tertentu seperti yang Anda lakukan dengan titik akhir kustom Aurora.
- Titik akhir pembaca tidak tersedia untuk klaster multi-penulis Aurora.

Memantau metrik Proxy RDS dengan Amazon CloudWatch

Anda dapat memantau Proxy RDS dengan menggunakan Amazon CloudWatch. CloudWatch mengumpulkan dan memproses data mentah dari proxy menjadi metrik yang dapat dibaca. near-real-time Untuk menemukan metrik ini di CloudWatch konsol, pilih Metrik, lalu pilih RDS, dan pilih Metrik Per-Proxy. Untuk informasi selengkapnya, lihat Menggunakan CloudWatch metrik Amazon di Panduan CloudWatch Pengguna Amazon.

> **Note**
>
> RDS menerbitkan metrik ini untuk setiap instans Amazon EC2 yang mendasarinya yang terkait dengan proksi. Satu proksi dapat dilayani oleh lebih dari satu instans EC2. Gunakan CloudWatch statistik untuk menggabungkan nilai untuk proxy di semua instance terkait. Beberapa metrik ini mungkin tidak terlihat hingga koneksi berhasil disambungkan terlebih dahulu oleh proksi.

Dalam log Proksi RDS, setiap entri diawali dengan nama titik akhir proksi terkait. Nama ini dapat berupa nama yang Anda tentukan untuk titik akhir yang ditetapkan pengguna, atau nama khusus `default` untuk titik akhir default proksi yang melakukan permintaan baca/tulis.

Semua metrik Proksi RDS berada di dalam grup `proxy`.

Setiap titik akhir proxy memiliki CloudWatch metriknya sendiri. Anda dapat memantau penggunaan setiap titik akhir proksi secara independen. Untuk informasi selengkapnya tentang titik akhir proksi, lihat Bekerja dengan titik akhir Proksi Amazon RDS.

Anda dapat menggabungkan beberapa nilai untuk setiap metrik menggunakan salah satu kumpulan dimensi berikut. Misalnya, dengan menggunakan kumpulan dimensi `ProxyName`, Anda dapat menganalisis semua lalu lintas untuk proksi tertentu. Dengan menggunakan kumpulan dimensi lainnya, Anda dapat membagi beberapa metrik dengan cara yang berbeda. Anda dapat membagi metrik berdasarkan titik akhir atau basis data target yang berbeda dari setiap proksi, atau lalu lintas baca/tulis dan hanya baca ke setiap basis data.

- Kumpulan dimensi 1: `ProxyName`
- Kumpulan dimensi 2: `ProxyName`, `EndpointName`
- Kumpulan dimensi 3: `ProxyName`, `TargetGroup`, `Target`
- Kumpulan dimensi 4: `ProxyName`, `TargetGroup`, `TargetRole`

Metrik	Deskripsi	Periode yang valid	CloudWatch set dimensi
`AvailabilityPercentage`	Persentase waktu saat grup target tersedia dalam peran yang diindikasikan oleh dimensi. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Average`.	1 menit	Dimension set 4

Metrik	Deskripsi	Periode yang valid	CloudWatch set dimensi
`ClientConnections`	Jumlah koneksi klien saat ini. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit	Dimension set 1, Dimension set 2
`ClientConnectionsClosed`	Jumlah koneksi klien yang ditutup. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit ke atas	Dimension set 1, Dimension set 2
`ClientConnectionsNoTLS`	Jumlah koneksi klien saat ini tanpa Keamanan Lapisan Pengangkutan (TLS). Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit ke atas	Dimension set 1, Dimension set 2
`ClientConnectionsReceived`	Jumlah permintaan koneksi klien yang diterima. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit ke atas	Dimension set 1, Dimension set 2

Metrik	Deskripsi	Periode yang valid	CloudWatch set dimensi
`ClientConnectionsSetupFailedAuth`	Jumlah upaya koneksi klien yang gagal karena konfigurasi autentikasi atau TLS yang salah. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit ke atas	Dimension set 1, Dimension set 2
`ClientConnectionsSetupSucceeded`	Jumlah koneksi klien yang berhasil dibangun dengan mekanisme autentikasi apa pun dengan atau tanpa TLS. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit ke atas	Dimension set 1, Dimension set 2
`ClientConnectionsTLS`	Jumlah koneksi klien saat ini dengan TLS. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit ke atas	Dimension set 1, Dimension set 2
`DatabaseConnectionRequests`	Jumlah permintaan untuk membuat koneksi basis data. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit ke atas	Dimension set 1, Dimension set 3, Dimension set 4

Metrik	Deskripsi	Periode yang valid	CloudWatch set dimensi
`DatabaseConnectionRequestsWithTLS`	Jumlah permintaan untuk membuat koneksi basis data dengan TLS. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit ke atas	Dimension set 1, Dimension set 3, Dimension set 4
`DatabaseConnections`	Jumlah koneksi basis data saat ini. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit	Dimension set 1, Dimension set 3, Dimension set 4
`DatabaseConnectionsBorrowLatency`	Waktu dalam mikrodetik yang dibutuhkan bagi proksi yang dipantau untuk mendapatkan koneksi basis data. Statistik yang paling berguna untuk metrik ini adalah `Average`.	1 menit ke atas	Dimension set 1, Dimension set 2
`DatabaseConnectionsCurrentlyBorrowed`	Jumlah koneksi basis data saat ini yang berada dalam status meminjam. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit	Dimension set 1, Dimension set 3, Dimension set 4

Metrik	Deskripsi	Periode yang valid	CloudWatch set dimensi
`DatabaseConnectionsCurrentlyInTransaction`	Jumlah koneksi basis data saat ini dalam transaksi. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit	Dimension set 1, Dimension set 3, Dimension set 4
`DatabaseConnectionsCurrentlySessionPinned`	Jumlah koneksi basis data saat ini yang saat ini disematkan karena adanya operasi dalam permintaan klien yang mengubah status sesi. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit	Dimension set 1, Dimension set 3, Dimension set 4
`DatabaseConnectionsSetupFailed`	Jumlah permintaan koneksi basis data yang gagal. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit ke atas	Dimension set 1, Dimension set 3, Dimension set 4

Metrik	Deskripsi	Periode yang valid	CloudWatch set dimensi
`DatabaseConnectionsSetupSucceeded`	Jumlah koneksi basis data yang berhasil dibangun dengan atau tanpa TLS. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit ke atas	Dimension set 1, Dimension set 3, Dimension set 4
`DatabaseConnectionsWithTLS`	Jumlah koneksi basis data saat ini dengan TLS. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit	Dimension set 1, Dimension set 3, Dimension set 4
`MaxDatabaseConnectionsAllowed`	Jumlah maksimum koneksi basis data yang diperbolehkan. Metrik ini dilaporkan setiap menit. Statistik yang paling berguna untuk metrik ini adalah `Sum`.	1 menit	Dimension set 1, Dimension set 3, Dimension set 4
`QueryDatabaseResponseLatency`	Waktu dalam mikrodetik yang dibutuhkan basis data untuk merespons kueri. Statistik yang paling berguna untuk metrik ini adalah `Average`.	1 menit ke atas	Dimension set 1, Dimension set 2, Dimension set 3, Dimension set 4

Metrik	Deskripsi	Periode yang valid	CloudWatch set dimensi
QueryRequests	Jumlah kueri yang diterima. Kueri yang mencakup beberapa pernyataan yang dihitung sebagai satu kueri. Statistik yang paling berguna untuk metrik ini adalah Sum.	1 menit ke atas	Dimension set 1, Dimension set 2
QueryRequestsNoTLS	Jumlah kueri yang diterima dari koneksi non-TLS. Kueri yang mencakup beberapa pernyataan yang dihitung sebagai satu kueri. Statistik yang paling berguna untuk metrik ini adalah Sum.	1 menit ke atas	Dimension set 1, Dimension set 2
QueryRequestsTLS	Jumlah kueri yang diterima dari koneksi TLS. Kueri yang mencakup beberapa pernyataan yang dihitung sebagai satu kueri. Statistik yang paling berguna untuk metrik ini adalah Sum.	1 menit ke atas	Dimension set 1, Dimension set 2

Metrik	Deskripsi	Periode yang valid	CloudWatch set dimensi
`QueryResponseLatency`	Waktu dalam mikrodetik antara saat mendapatkan permintaan kueri dan saat proksi meresponsnya. Statistik yang paling berguna untuk metrik ini adalah `Average`.	1 menit ke atas	Dimension set 1, Dimension set 2

Anda dapat menemukan log aktivitas Proxy RDS CloudWatch di bawah. AWS Management Console Setiap proksi memiliki entri di halaman Grup log.

> **Important**
>
> Log ini ditujukan untuk konsumsi manusia untuk tujuan pemecahan masalah dan bukan untuk akses terprogram. Format dan konten log dapat berubah sewaktu-waktu.
> Khususnya, log lama tidak berisi awalan yang menunjukkan titik akhir untuk setiap permintaan. Dalam log yang lebih baru, setiap entri diawali dengan nama titik akhir proksi terkait. Nama ini dapat berupa nama yang Anda tentukan untuk titik akhir yang ditentukan pengguna, atau nama khusus `default` untuk permintaan yang menggunakan titik akhir default proksi.

Bekerja dengan peristiwa Proksi RDS

Peristiwa menunjukkan perubahan dalam lingkungan seperti lingkungan AWS atau layanan atau aplikasi dari mitra perangkat lunak sebagai layanan (SaaS). Atau bisa berupa salah satu aplikasi atau layanan kustom Anda sendiri. Misalnya, Amazon Aurora menghasilkan peristiwa saat Anda membuat atau memodifikasi Proksi RDS. Amazon Aurora mengirimkan acara ke CloudWatch Acara dan EventBridge Amazon dalam waktu hampir nyata. Temukan daftar peristiwa Proksi RDS yang dapat dijadikan langganan dan contoh peristiwa Proksi RDS di bawah.

Berikut informasi selengkapnya tentang cara bekerja dengan peristiwa:

- Untuk petunjuk tentang cara melihat peristiwa menggunakan AWS Management Console, AWS CLI, atau API RDS, lihat Melihat peristiwa Amazon RDS.
- Untuk mempelajari cara mengonfigurasi Amazon Aurora untuk mengirim acara EventBridge, lihat. Membuat aturan yang dipicu pada peristiwa Amazon Aurora

Peristiwa Proksi RDS

Tabel berikut menunjukkan kategori peristiwa dan daftar peristiwa saat Proksi RDS berupa jenis sumber.

Kategori	ID peristiwa RDS	Pesan	Catatan
perubahan konfigurasi	RDS-EVENT-0204	RDS memodifikasi proksi DB *name*.	
perubahan konfigurasi	RDS-EVENT-0207	RDS memodifikasi titik akhir proksi DB *name*.	
perubahan konfigurasi	RDS-EVENT-0213	RDS mendeteksi penambahan instans DB dan secara otomatis menambahkannya ke grup target proksi DB *name*.	
perubahan konfigurasi	RDS-EVENT-0213	RDS mendeteksi pembuatan instans DB *name* dan secara otomatis menambahkannya ke grup target *name* proksi DB *name*.	
perubahan konfigurasi	RDS-EVENT-0214	RDS mendeteksi penghapusan instans DB *name* dan secara otomatis menghapusnya dari grup target *name* proksi DB *name*.	

Kategori	ID peristiwa RDS	Pesan	Catatan
perubahan konfigurasi	RDS-EVENT-0215	RDS mendeteksi penghapusan klaster DB *name* dan secara otomatis menghapusnya dari grup target *name* proksi DB *name*.	
pembuatan	RDS-EVENT-0203	RDS membuat proksi DB *name*.	
pembuatan	RDS-EVENT-0206	RDS membuat titik akhir *name* untuk proksi DB *name*.	
penghapusan	RDS-EVENT-0205	RDS menghapus proksi DB *name*.	
penghapusan	RDS-EVENT-0208	RDS menghapus titik akhir *name* untuk proksi DB *name*.	
kegagalan	RDS-EVENT-0243	RDS gagal menyediakan kapasitas untuk proksi *name* karena tidak ada alamat IP yang cukup yang tersedia di subnet Anda: *name*. Untuk memperbaiki masalah ini, pastikan subnet Anda memiliki jumlah minimum alamat IP yang tidak digunakan seperti yang direkomendasikan dalam dokumentasi Proksi RDS.	Untuk menentukan jumlah yang direkomendasikan untuk kelas instans Anda, lihat Perencanaan untuk kapasitas alamat IP.

Kategori	ID peristiwa RDS	Pesan	Catatan
kegagalan	RDS-EVENT-0275	*RDS membatasi beberapa koneksi ke nama proxy DB.* Jumlah permintaan koneksi simultan dari klien ke proxy telah melampaui batas.	

Berikut adalah contoh peristiwa Proksi RDS dalam format JSON. Peristiwa ini menunjukkan bahwa RDS memodifikasi titik akhir bernama `my-endpoint` dari Proksi RDS bernama `my-rds-proxy`. ID peristiwa ini adalah RDS-EVENT-0207.

```
{
  "version": "0",
  "id": "68f6e973-1a0c-d37b-f2f2-94a7f62ffd4e",
  "detail-type": "RDS DB Proxy Event",
  "source": "aws.rds",
  "account": "123456789012",
  "time": "2018-09-27T22:36:43Z",
  "region": "us-east-1",
  "resources": [
     "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy"
  ],
  "detail": {
    "EventCategories": [
      "configuration change"
    ],
    "SourceType": "DB_PROXY",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:db-proxy:my-rds-proxy",
    "Date": "2018-09-27T22:36:43.292Z",
    "Message": "RDS modified endpoint my-endpoint of DB Proxy my-rds-proxy.",
    "SourceIdentifier": "my-endpoint",
    "EventID": "RDS-EVENT-0207"
  }
}
```

Contoh baris perintah Proksi RDS

Untuk melihat cara kombinasi perintah koneksi dan pernyataan SQL berinteraksi dengan Proksi RDS, perhatikan contoh berikut.

Contoh

- Preserving Connections to a MySQL Database Across a Failover
- Adjusting the max_connections Setting for an Aurora DB Cluster

Example Mencadangkan koneksi ke basis data MySQL di seluruh failover

Contoh MySQL ini menunjukkan bagaimana koneksi terbuka terus berfungsi selama failover. Contohnya adalah ketika Anda me-reboot basis data atau basis data tidak tersedia karena terjadi masalah. Contoh ini menggunakan proksi bernama `the-proxy` dan klaster DB Aurora dengan instans DB `instance-8898` dan `instance-9814`. Saat Anda menjalankan perintah `failover-db-cluster` dari baris perintah Linux, instans penulis yang terhubung ke proksi berubah ke instans DB yang berbeda. Anda dapat melihat bahwa instans DB yang terkait dengan proksi ini berubah saat koneksi masih terbuka.

```
$ mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p
Enter password:
...

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                 mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ # Initially, instance-9814 is the writer.
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-8898 is the writer.
$ fg
```

```
mysql -h the-proxy.proxy-demo.us.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-8898      |
+--------------------+
1 row in set (0.01 sec)

mysql>
[1]+  Stopped                 mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com
 -u admin_user -p
$ aws rds failover-db-cluster --db-cluster-identifier cluster-56-2019-11-14-1399
JSON output
$ # After a short time, the console shows that the failover operation is complete.
$ # Now instance-9814 is the writer again.
$ fg
mysql -h the-proxy.proxy-demo.us-east-1.rds.amazonaws.com -u admin_user -p

mysql> select @@aurora_server_id;
+--------------------+
| @@aurora_server_id |
+--------------------+
| instance-9814      |
+--------------------+
1 row in set (0.01 sec)
+---------------+---------------+
| Variable_name | Value         |
+---------------+---------------+
| hostname      | ip-10-1-3-178 |
+---------------+---------------+
1 row in set (0.02 sec)
```

Example Menyesuaikan pengaturan max_connections untuk klaster DB Aurora

Contoh ini menunjukkan cara menyesuaikan pengaturan `max_connections` untuk klaster DB Aurora MySQL. Untuk melakukannya, buat grup parameter klaster DB Anda sendiri berdasarkan pengaturan parameter default untuk klaster yang kompatibel dengan MySQL 5.7. Tentukan nilai untuk pengaturan `max_connections`, dengan mengganti formula yang menetapkan nilai default. Kaitkan grup parameter klaster DB dengan klaster DB Anda.

```
export REGION=us-east-1
```

```
export CLUSTER_PARAM_GROUP=rds-proxy-mysql-57-max-connections-demo
export CLUSTER_NAME=rds-proxy-mysql-57

aws rds create-db-parameter-group --region $REGION \
  --db-parameter-group-family aurora-mysql5.7 \
  --db-parameter-group-name $CLUSTER_PARAM_GROUP \
  --description "Aurora MySQL 5.7 cluster parameter group for RDS Proxy demo."

aws rds modify-db-cluster --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP

echo "New cluster param group is assigned to cluster:"
aws rds describe-db-clusters --region $REGION \
  --db-cluster-identifier $CLUSTER_NAME \
  --query '*[*].{DBClusterParameterGroup:DBClusterParameterGroup}'

echo "Current value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
  --output text | grep "^max_connections"

echo -n "Enter number for max_connections setting: "
read answer

aws rds modify-db-cluster-parameter-group --region $REGION --db-cluster-parameter-
group-name $CLUSTER_PARAM_GROUP \
  --parameters "ParameterName=max_connections,ParameterValue=$
$answer,ApplyMethod=immediate"

echo "Updated value for max_connections:"
aws rds describe-db-cluster-parameters --region $REGION \
  --db-cluster-parameter-group-name $CLUSTER_PARAM_GROUP \
  --query '*[*].{ParameterName:ParameterName,ParameterValue:ParameterValue}' \
  --output text | grep "^max_connections"
```

Pemecahan masalah untuk Proksi RDS

Setelah itu, Anda dapat menemukan ide pemecahan masalah untuk beberapa masalah Proxy RDS umum dan informasi tentang CloudWatch log untuk RDS Proxy.

Dalam log Proksi RDS, setiap entri diawali dengan nama titik akhir proksi terkait. Nama ini bisa berupa nama yang Anda tentukan untuk titik akhir yang ditentukan pengguna. Atau, bisa menjadi nama khusus `default` untuk titik akhir default proksi yang melakukan permintaan baca/tulis. Untuk informasi selengkapnya tentang titik akhir proksi, lihat Bekerja dengan titik akhir Proksi Amazon RDS.

Topik

- Memverifikasi konektivitas untuk proksi
- Masalah dan solusi umum

Memverifikasi konektivitas untuk proksi

Anda dapat menggunakan perintah berikut untuk memverifikasi bahwa semua komponen seperti proksi, basis data, dan instans komputasi dalam koneksi dapat berkomunikasi satu sama lain.

Periksa proxy itu sendiri menggunakan describe-db-proxiesperintah. Periksa juga kelompok target terkait menggunakan perintah describe-db-proxy-target-groups. Periksa apakah detail target cocok dengan klaster Aurora yang ingin dikaitkan dengan proksi. Gunakan perintah seperti berikut ini.

```
aws rds describe-db-proxies --db-proxy-name $DB_PROXY_NAME
aws rds describe-db-proxy-target-groups --db-proxy-name $DB_PROXY_NAME
```

Untuk mengonfirmasi bahwa proxy dapat terhubung ke database yang mendasarinya, periksa target yang ditentukan dalam grup target menggunakan describe-db-proxy-targetsperintah. Gunakan perintah seperti berikut.

```
aws rds describe-db-proxy-targets --db-proxy-name $DB_PROXY_NAME
```

Output dari describe-db-proxy-targetsperintah termasuk `TargetHealth` bidang. Anda dapat memeriksa kolom `State`, `Reason`, dan `Description` di dalam `TargetHealth` untuk memeriksa apakah proksi dapat berkomunikasi dengan instans DB acuan.

- Nilai `State` dari `AVAILABLE` mengindikasikan bahwa proksi dapat terhubung ke instans DB.
- Nilai `State` dari `UNAVAILABLE` mengindikasikan masalah koneksi sementara atau permanen. Dalam kasus ini, periksa kolom `Reason` dan `Description`. Misalnya, jika `Reason` memiliki nilai `PENDING_PROXY_CAPACITY`, coba hubungkan lagi setelah proksi menyelesaikan operasi penskalaan. Jika `Reason` memiliki nilai `UNREACHABLE`, `CONNECTION_FAILED`, atau

`AUTH_FAILURE`, gunakan penjelasan dari kolom `Description` untuk membantu Anda mendiagnosis masalah.

- Kolom `State` muungkin memiliki nilai `REGISTERING` dalam waktu yang singkat sebelum berubah ke `AVAILABLE` atau `UNAVAILABLE`.

Jika perintah Netcat berikut (`nc`) berhasil, Anda dapat mengakses titik akhir proksi dari instans EC2 atau sistem lain tempat Anda masuk. Perintah ini melaporkan kegagalan jika Anda tidak berada di VPC yang sama dengan proksi dan basis data terkait. Anda mungkin dapat masuk secara langsung ke basis data tanpa berada di VPC yang sama. Akan tetapi, Anda tidak dapat masuk ke proksi kecuali Anda berada di VPC yang sama.

```
nc -zx MySQL_proxy_endpoint 3306

nc -zx PostgreSQL_proxy_endpoint 5432
```

Anda dapat menggunakan perintah berikut untuk memastikan bahwa instans EC2 Anda memiliki properti yang dibutuhkan. Khususnya, VPC untuk instans EC2 harus sama dengan VPC untuk instans DB RDS klaster Aurora yang terhubung dengan proksi.

```
aws ec2 describe-instances --instance-ids your_ec2_instance_id
```

Periksa rahasia Secrets Manager yang digunakan oleh proksi.

```
aws secretsmanager list-secrets
aws secretsmanager get-secret-value --secret-id your_secret_id
```

Pastikan kolom `SecretString` yang ditampilkan oleh `get-secret-value` dienkode sebagai string JSON yang mencakup kolom `username` dan `password`. Contoh berikut menunjukkan format kolom `SecretString`.

```
{
  "ARN": "some_arn",
  "Name": "some_name",
  "VersionId": "some_version_id",
  "SecretString": '{"username":"some_username","password":"some_password"}',
  "VersionStages": [ "some_stage" ],
  "CreatedDate": some_timestamp
}
```

Masalah dan solusi umum

Bagian ini menjelaskan beberapa masalah umum dan solusi potensial saat menggunakan Proksi RDS.

Setelah menjalankan perintah `aws rds describe-db-proxy-targets` CLI, jika deskripsi `TargetHealth` menyatakan `Proxy does not have any registered credentials`, verifikasi hal-hal berikut:

- Ada kredensial yang terdaftar bagi pengguna untuk mengakses proksi.
- Peran IAM untuk mengakses rahasia Secrets Manager yang digunakan oleh proksi valid.

Anda mungkin mengalami peristiwa RDS berikut saat membuat atau terhubung ke proksi DB.

Kategori	ID peristiwa RDS	Deskripsi
kegagalan	RDS-EVENT-0243	RDS gagal menyediakan kapasitas untuk proksi karena tidak ada alamat IP yang cukup yang tersedia di subnet Anda. Untuk memperbai ki masalah ini, pastikan subnet Anda memiliki jumlah minimum alamat IP yang tidak digunakan. Untuk menentuka n jumlah yang direkomen dasikan untuk kelas instans Anda, lihat Perencanaan untuk kapasitas alamat IP.
kegagalan	RDS-EVENT-0275	*`RDS membatasi beberapa koneksi ke nama proxy DB.`* Jumlah permintaan koneksi simultan dari klien ke proxy telah melampaui batas.

Anda mungkin mengalami masalah berikut saat membuat proksi baru atau terhubung ke proksi.

Kesalahan	Penyebab atau solusi
`403: The security token included in the request is invalid`	Pilih peran IAM yang sudah ada, bukan memilih untuk membuat peran baru.

Anda mungkin mengalami masalah berikut saat terhubung ke proksi MySQL.

Kesalahan	Penyebab atau solusi
`ERROR 1040 (HY000): Connections rate limit exceeded (`*`limit_value`*` )`	Tingkat permintaan koneksi dari klien ke proksi telah melampaui batas.
`ERROR 1040 (HY000): IAM authentication rate limit exceeded`	Jumlah permintaan simultan dengan autentikasi IAM dari klien ke proksi telah melampaui batas.
`ERROR 1040 (HY000): Number simultaneous connections exceeded (`*`limit_value`*` )`	Jumlah permintaan koneksi simultan dari klien ke proksi telah melampaui batas.
`ERROR 1045 (28000): Access denied for user`	Rahasia Secrets Manager yang digunakan oleh proksi tidak cocok dengan nama pengguna dan kata sandi dari pengguna basis data yang sudah ada. Perbarui kredensial dalam rahasia Secrets Manager, atau

Kesalahan	Penyebab atau solusi
`'DB_USER'@'%' (usi password: YES)`	pastikan pengguna basis data ada dan memiliki kata sandi yang sama dengan yang ada dalam rahasia.
`ERROR 1105 (HY000): Unknown error`	Terjadi kesalahan yang tidak diketahui.
`ERROR 1231 (42000): Variable ''charact er_set_cl ient'' can't be set to the value of value`	Nilai yang diatur untuk parameter `character_set_client` tidak valid. Misalnya, nilai `ucs2` tidak valid karena dapat merusak server MySQL Anda.
`ERROR 3159 (HY000): This RDS Proxy requires TLS connections.`	Anda mengaktifkan pengaturan Wajibkan Keamanan Lapisan Pengangkutan dalam proksi, tetapi koneksi Anda menyertakan parameter `ssl-mode=DISABLED` dalam klien MySQL. Lakukan salah satu dari langkah berikut: • Nonaktifkan pengaturan Wajibkan Keamanan Lapisan Pengangkutan untuk proksi. • Hubungkan ke basis data menggunakan pengaturan minimum `ssl-mode=REQUIRED` dalam klien MySQL.
`ERROR 2026 (HY000): SSL connection error: Internal Server Error`	Proses jabat tangan TLS dengan proksi gagal. Beberapa kemungkinan alasannya meliputi: • SSL dibutuhkan, tetapi server tidak mendukungnya. • Terjadi kesalahan server internal. • Terjadi jabat tangan yang buruk.

Kesalahan	Penyebab atau solusi
`ERROR 9501 (HY000): Timed-out waiting to acquire database connection`	Waktu tunggu proksi untuk memperoleh koneksi basis data habis. Beberapa kemungkinan alasannya meliputi: • Proksi tidak dapat membangun koneksi basis data karena koneksi maksimum sudah tercapai • Proksi tidak dapat membangun koneksi basis data karena basis data tidak tersedia.

Anda mungkin mengalami masalah berikut saat terhubung ke proksi PostgreSQL.

Kesalahan	Penyebab	Solusi
`IAM authentication is allowed only with SSL connections.`	Pengguna mencoba terhubung ke basis data menggunakan autentikasi IAM dengan pengaturan `sslmode=disable` dalam klien PostgreSQL.	Pengguna harus terhubung ke basis data menggunakan pengaturan minimum `sslmode=require` dalam klien PostgreSQL. Untuk informasi lebih lanjut, lihat dokumentasi Dukungan PostgreSQL SSL.
`This RDS Proxy requires TLS connections.`	Pengguna mengaktifkan opsi Wajibkan Keamanan Lapisan Pengangkutan, tetapi mencoba untuk terhubung dengan `sslmode=disable` dalam klien PostgreSQL.	Untuk memperbaiki kesalahan ini, lakukan salah satu tindakan berikut: • Nonaktifkan opsi Wajibkan Keamanan Lapisan Pengangkutan proksi. • Hubungkan ke basis data menggunakan pengaturan minimum `sslmode=allow` dalam klien PostgreSQL.

Kesalahan	Penyebab	Solusi
`IAM authentication failed for user user_name . Check the IAM token for this user and try again.`	Kesalahan ini mungkin terjadi karena alasan berikut: • Klien memberikan nama pengguna IAM yang salah. • Klien memberikan token otorisasi IAM yang salah untuk pengguna. • Klien menggunakan kebijakan IAM yang tidak memiliki izin yang dibutuhkan. • Klien memberikan token otorisasi IAM yang kedaluwarsa untuk pengguna.	Untuk memperbaiki kesalahan ini, lakukan tindakan berikut: 1. Pastikan pengguna IAM yang disediakan sudah ada. 2. Pastikan token otorisasi IAM adalah milik pengguna IAM yang disediakan. 3. Pastikan kebijakan IAM memiliki izin yang memadai untuk RDS. 4. Periksa validitas token otorisasi IAM yang digunakan.
`This RDS proxy has no credentials for the role role_name . Check the credentials for this role and try again.`	Tidak ada rahasia Secrets Manager untuk peran ini.	Tambahkan rahasia Secrets Manager untuk peran ini. Untuk informasi selengkapnya, lihat Menyiapkan AWS Identity and Access Management kebijakan (IAM).
`RDS supports only IAM, MD5, or SCRAM authentication.`	Klien basis data yang digunakan untuk terhubung ke proksi menggunakan mekanisme autentikasi yang saat ini tidak didukung oleh proksi.	Jika Anda tidak menggunakan autentikasi IAM, gunakan autentikasi kata sandi MD5 atau SCRAM.

Kesalahan	Penyebab	Solusi
`A user name is missing from the connection startup packet. Provide a user name for this connection.`	Klien basis data yang digunakan untuk terhubung ke proksi tidak mengirimkan nama pengguna saat mencoba membangun koneksi.	Pastikan untuk menentukan sebuah nama pengguna saat menyiapkan koneksi ke proksi menggunakan klien PostgreSQL pilihan Anda.
`Feature not supported: RDS Proxy supports only version 3.0 of the PostgreSQL messaging protocol.`	Klien PostgreSQL yang digunakan untuk terhubung ke proksi menggunakan protokol yang lebih lama dari 3.0.	Gunakan klien PostgreSQL yang lebih baru yang mendukung protokol olah pesan 3.0. Jika Anda menggunakan CLI `psql` PostgreSQL, gunakan versi yang lebih baru atau yang setara dengan 7.4.
`Feature not supported: RDS Proxy currently doesn't support streaming replication mode.`	Klien PostgreSQL yang digunakan untuk terhubung ke proksi mencoba menggunakan mode replikasi streaming, yang saat ini tidak didukung oleh Proksi RDS.	Nonaktifkan mode replikasi streaming dalam klien PostgreSQL yang digunakan untuk menghubungkan.
`Feature not supported: RDS Proxy currently doesn't support the option` *`option_name`* `.`	Melalui pesan startup, klien PostgreSQL yang digunakan untuk terhubung ke proksi meminta opsi yang saat ini tidak didukung oleh Proksi RDS.	Nonaktifkan opsi yang ditampilkan sebagai tidak didukung dari pesan di atas dalam klien PostgreSQL yang digunakan untuk menghubungkan.
`The IAM authentication failed because of too many competing requests.`	Jumlah permintaan simultan dengan autentikasi IAM dari klien ke proksi telah melampaui batas.	Kurangi tingkat pembangunan koneksi menggunakan autentikasi IAM dari klien PostgreSQL.

Kesalahan	Penyebab	Solusi
`The maximum number of client connections to the proxy exceeded` *`number_value`* `.`	Jumlah permintaan koneksi simultan dari klien ke proksi telah melampaui batas.	Kurangi jumlah koneksi aktif dari klien PostgreSQL ke proksi RDS ini.
`Rate of connectio n to proxy exceeded` *`number_value`* `.`	Tingkat permintaan koneksi dari klien ke proksi telah melampaui batas.	Kurangi tingkat pembangunan koneksi dari klien PostgreSQL.
`The password that was provided for the role` *`role_name`* `is wrong.`	Kata sandi untuk peran ini tidak cocok dengan rahasia Secrets Manager.	Periksa rahasia peran ini dalam Secrets Manager untuk melihat apakah kata sandi sama dengan yang digunakan dalam klien PostgreSQL Anda.
`The IAM authentic ation failed for the role` *`role_name`* `. Check the IAM token for this role and try again.`	Terjadi masalah dengan token IAM yang digunakan untuk autentikasi IAM.	Buat token autentikasi baru dan gunakan dalam koneksi baru.
`IAM is allowed only with SSL connections.`	Klien mencoba untuk terhubung menggunakan autentikasi IAM, tetapi SSL tidak diaktifkan.	Aktifkan SSL dalam klien PostgreSQL.
`Unknown error.`	Terjadi kesalahan yang tidak diketahui.	Hubungi Dukungan AWS untuk menyelidiki masalah ini.

Kesalahan	Penyebab	Solusi
`Timed-out waiting to acquire database connection.`	Waktu tunggu proksi untuk memperoleh koneksi basis data habis. Beberapa kemungkinan alasannya meliputi: • Proksi tidak dapat membangun koneksi basis data karena koneksi maksimum sudah tercapai. • Proksi tidak dapat membangun koneksi basis data karena basis data tidak tersedia.	Kemungkinan solusinya sebagai berikut: • Periksa target instans DB RDSklaster Aurora untuk melihat apakah basis data tidak tersedia. • Periksa apakah ada eksekusi transaksi dan/atau kueri yang berjalan lama. Transaksi atau kueri ini bisa menggunakan koneksi basis data dari kumpulan koneksi dalam waktu yang lama.
`Request returned an error: database_error .`	Koneksi basis data yang dibangun dari proksi menampilkan kesalahan.	Solusinya bergantung dari kesalahan basis data tertentu. Salah satu contohnya adalah: `Request returned an error: database "your-database-name" does not exist`. Ini berarti bahwa nama basis data yang ditentukan tidak ada di server basis data. Atau, nama pengguna yang digunakan sebagai sebuah nama basis data (jika nama basis data tidak ditentukan) tidak ada di server.

Penggunaan Proksi RDS dengan AWS CloudFormation

Anda dapat menggunakan Proksi RDS dengan AWS CloudFormation. Hal ini membantu Anda membuat grup sumber daya terkait. Grup semacam ini bisa mencakup proksi yang dapat terhubung ke klaster DB Aurora. Dukungan Proksi RDS dalam AWS CloudFormation melibatkan dua jenis registri baru: `DBProxy` dan `DBProxyTargetGroup`.

Daftar berikut menunjukkan contoh templat AWS CloudFormation untuk Proksi RDS.

```
Resources:
 DBProxy:
   Type: AWS::RDS::DBProxy
   Properties:
     DBProxyName: CanaryProxy
     EngineFamily: MYSQL
     RoleArn:
      Fn::ImportValue: SecretReaderRoleArn
     Auth:
       - {AuthScheme: SECRETS, SecretArn: !ImportValue ProxySecret, IAMAuth: DISABLED}
     VpcSubnetIds:
       Fn::Split: [",", "Fn::ImportValue": SubnetIds]

 ProxyTargetGroup:
   Type: AWS::RDS::DBProxyTargetGroup
   Properties:
     DBProxyName: CanaryProxy
     TargetGroupName: default
     DBInstanceIdentifiers:
       - Fn::ImportValue: DBInstanceName
   DependsOn: DBProxy
```

Untuk informasi selengkapnya tentang sumber daya dalam sampel ini, lihat `DBProxy`dan `DBProxyTargetGroup`.

Untuk informasi selengkapnya tentang sumber daya yang dapat dibuat menggunakan AWS CloudFormation, lihat Referensi jenis sumber daya RDS.

Menggunakan Proksi RDS dengan basis data global Aurora

Basis data global Aurora adalah basis data tunggal yang mencakup beberapa Wilayah AWS, yang memungkinkan pembacaan global latensi rendah dan pemulihan bencana dari pemadaman tingkat

Wilayah. Basis data ini menyediakan toleransi kesalahan bawaan untuk deployment karena instans DB tidak mengandalkan satu Wilayah AWS, tetapi mengandalkan beberapa Wilayah dan Zona Ketersediaan yang berbeda. Untuk informasi selengkapnya, lihat Menggunakan basis data global Amazon Aurora.

Anda dapat menggunakan Proksi RDS dengan klaster DB apa pun dalam basis data global Aurora. Sebelum mulai menggunakan fitur-fitur ini bersama-sama, pastikan Anda benar-benar memahami informasi berikut.

> **Important**
>
> Jika klaster DB adalah bagian dari basis data global dengan penerusan tulis yang diaktifkan, kurangi nilai `MaxConnectionsPercent` proksi dengan kuota yang dialokasikan untuk penerusan tulis. Kuota penerusan tulis diatur dalam parameter klaster DB `aurora_fwd_writer_max_connections_pct`. Untuk informasi tentang penerusan tulis, lihat Menggunakan penerusan menulis dalam basis data global Amazon Aurora.

Batasan untuk Proksi RDS dengan basis data global

Jika penerusan tulis klaster DB Aurora diaktifkan, Proksi RDS tidak mendukung nilai `SESSION` untuk variabel `aurora_replica_read_consistency`. Menetapkan nilai ini dapat menimbulkan perilaku tak terduga.

Cara kerja titik akhir Proksi RDS dengan basis data global

Jika Anda memahami cara kerja titik akhir Proksi RDS dengan basis data global, Anda akan lebih mudah mengelola aplikasi yang menggunakan basis data Aurora dengan kedua fitur ini.

Untuk proksi dengan klaster primer basis data global sebagai target terdaftar, titik akhir proksi berfungsi seperti halnya dengan klaster DB Aurora apa pun. Titik akhir baca/tulis proksi mengirim semua permintaan ke instans penulis klaster. Titik akhir hanya baca proksi mengirim semua permintaan ke instans pembaca. Jika pembaca tidak tersedia saat koneksi terbuka, Proksi RDS akan mengalihkan kueri berikutnya pada koneksi ke instans pembaca lain. Untuk proksi dengan klaster sekunder sebagai target terdaftar, permintaan yang dikirim ke titik akhir hanya baca proksi juga dikirim ke instans pembaca. Karena klaster tidak memiliki instans penulis, permintaan yang dikirim ke titik akhir baca/tulis gagal dengan kesalahan "`The target group doesn't have any associated read/write instances`".

Operasi failover dan switchover basis data global melibatkan peralihan peran antara klaster DB primer dan salah satu klaster DB sekunder. Ketika klaster sekunder yang dipilih menjadi klaster primer baru, salah satu instans pembacanya ditingkatkan menjadi instans penulis. Instans DB ini kini menjadi instans penulis baru untuk klaster global. Pastikan untuk mengalihkan operasi tulis aplikasi ke titik akhir baca/tulis yang sesuai dari proksi yang terkait dengan klaster primer baru. Titik akhir proksi ini mungkin merupakan titik akhir default atau titik akhir baca/tulis kustom.

Proksi RDS mengantrekan semua permintaan melalui titik akhir baca/tulis dan mengirimkannya ke instans penulis dari klaster primer baru segera setelah tersedia. Ini dilakukan terlepas dari apakah operasi switchover atau failover telah selesai. Selama switchover atau failover, titik akhir default proksi untuk klaster primer lama masih menerima operasi tulis. Namun, segera setelah klaster itu menjadi klaster sekunder, semua operasi tulis gagal. Untuk mempelajari bagaimana dan kapan harus melakukan tugas switchover atau failover global tertentu, lihat topik berikut:

- Switchover basis data global – Melakukan switchover untuk basis data global Amazon Aurora
- Failover basis data global – Memulihkan basis data global Amazon Aurora dari pemadaman yang tidak direncanakan

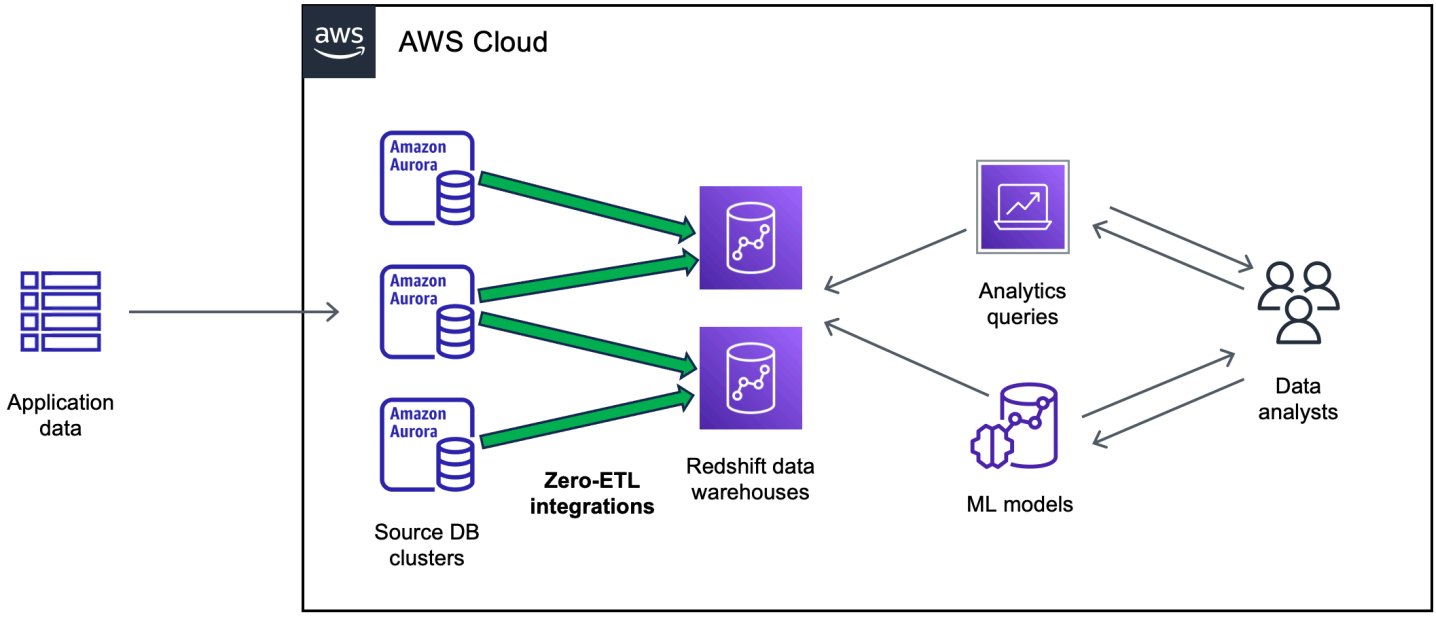
Menggunakan integrasi nol-ETL Aurora dengan Amazon Redshift

Integrasi nol-ETL Aurora dengan Amazon Redshift memungkinkan analisis hampir waktu nyata dan machine learning (ML) menggunakan Amazon Redshift pada data transaksional berukuran petabyte dari Aurora. Ini adalah solusi yang dikelola sepenuhnya untuk membuat data transaksional tersedia di Amazon Redshift setelah ditulis ke Aurora DB cluster. Extract, transform, and load (ETL) adalah proses menggabungkan data dari berbagai sumber ke dalam repositori pusat yang besar.

Integrasi nol-ETL membuat data dalam DB tersedia di Amazon Redshift dalam waktu dekat. Setelah data tersebut berada di Amazon Redshift, Anda dapat memberi daya pada beban kerja analitik, ML, dan AI menggunakan kemampuan bawaan Amazon Redshift, seperti pembelajaran mesin, tampilan terwujud, berbagi data, akses gabungan ke beberapa penyimpanan data dan data lake, serta integrasi dengan Amazon, Amazon, dan lainnya. SageMaker QuickSight Layanan AWS

Untuk membuat integrasi Nol-ETL, Anda menentukan sebagai sumbernya, dan gudang data Amazon Redshift sebagai target. Integrasi ini mereplikasi data dari basis data sumber ke gudang data target.

Diagram berikut menggambarkan fungsi ini:



Integrasi memantau kondisi pipeline data dan memulihkan dari masalah jika memungkinkan. Anda dapat membuat integrasi dari beberapa klaster Aurora DB ke dalam satu namespace Amazon Redshift, memungkinkan Anda memperoleh wawasan di beberapa aplikasi.

Untuk informasi tentang harga integrasi nol-ETL, lihat Harga Amazon Aurora dan Harga Amazon Redshift.

Topik

- Manfaat
- Konsep utama
- Batasan
- Kuota
- Wilayah yang Didukung
- Mulai menggunakan integrasi nol-ETL Aurora dengan Amazon Redshift
- Membuat integrasi nol-ETL Aurora dengan Amazon Redshift
- Pemfilteran data untuk integrasi Zero-ETL dengan Amazon Redshift
- Menambahkan data ke sumber cluster Aurora DB dan menanyakannya di Amazon Redshift
- Melihat dan memantau integrasi nol-ETL Aurora dengan Amazon Redshift
- Memodifikasi integrasi Zero-ETL dengan Amazon Redshift
- Menghapus integrasi nol-ETL Aurora dengan Amazon Redshift
- Memecahkan masalah integrasi nol-ETL Aurora dengan Amazon Redshift

Manfaat

Integrasi nol-ETL Aurora dengan Amazon Redshift memiliki manfaat berikut:

- Membantu Anda memperoleh wawasan menyeluruh dari berbagai sumber data.
- Menghilangkan kebutuhan untuk membangun dan memelihara pipeline data yang kompleks yang melakukan operasi extract, transform, and load (ETL). Integrasi nol-ETL menghilangkan tantangan yang muncul dalam membangun dan mengelola pipeline dengan menyediakan dan mengelolanya untuk Anda.
- Mengurangi beban dan biaya operasional, serta membantu Anda fokus pada peningkatan aplikasi Anda.
- Memungkinkan Anda memanfaatkan analitik Amazon Redshift dan kemampuan ML untuk memperoleh wawasan dari data transaksional dan data lainnya, guna merespons secara efektif peristiwa kritis dan sensitif terhadap waktu.

Konsep utama

Saat mulai menggunakan integrasi nol-ETL, pertimbangkan konsep berikut ini:

Integrasi

Pipa data yang dikelola sepenuhnya yang secara otomatis mereplikasi data dan skema transaksional dari DB cluster ke gudang data Amazon Redshift.

sumber DB cluster

Aurora DB cluster tempat data direplikasi. Untuk Aurora MySQL, Anda dapat menentukan klaster DB yang menggunakan instans DB terprovisi atau instans DB Aurora Serverless v2 sebagai sumbernya. Untuk Aurora PostgreSQL pratinjau, Anda hanya dapat menentukan klaster yang menggunakan instans DB yang disediakan. Beberapa sumber DB cluster dapat menulis ke target yang sama. Ada beberapa batasan pengaturan untuk cluster DB sumber, yang diuraikan dalamthe section called “Batasan”.

Gudang data target

Gudang data Amazon Redshift tempat tujuan data direplikasi. Ada dua jenis gudang data: gudang data klaster terprovisi dan gudang data nirserver. Gudang data klaster terprovisi adalah kumpulan sumber daya komputasi yang disebut simpul, yang diatur ke dalam grup yang disebut klaster. Gudang data nirserver terdiri dari grup kerja yang menyimpan sumber daya komputasi, serta ruang nama yang menampung objek basis data dan pengguna. Kedua gudang data ini menjalankan mesin Amazon Redshift dan berisi satu atau beberapa basis data.

Untuk informasi selengkapnya, lihat Arsitektur sistem gudang data dalam Panduan Developer Amazon Redshift.

Batasan

Batasan berikut berlaku pada integrasi nol-ETL Aurora dengan Amazon Redshift.

Topik

- Batasan umum
- Batasan Aurora MySQL
- Batasan Aurora PostgreSQL pratinjau
- Batasan Amazon Redshift

Batasan umum

- Cluster DB sumber harus berada di Wilayah yang sama dengan gudang data Amazon Redshift target.
- Anda tidak dapat mengganti nama cluster DB atau instance-nya jika memiliki integrasi yang ada.
- Anda tidak dapat menghapus cluster DB yang memiliki integrasi yang ada. Anda harus menghapus semua integrasi yang terkait terlebih dahulu.
- Jika Anda menghentikan cluster DB sumber, beberapa transaksi terakhir mungkin tidak direplikasi ke gudang data target sampai Anda melanjutkan cluster .
- Integrasi nol-ETL saat ini tidak mendukung pemfilteran data.
- Jika kluster Anda adalah sumber penerapan biru/hijau, lingkungan biru dan hijau tidak dapat memiliki integrasi nol-ETL selama peralihan. Anda harus menghapus integrasi tersebut terlebih dahulu dan beralih, lalu membuat ulang integrasi.
- Jika klaster sumber Anda adalah klaster DB primer dalam basis data global Aurora dan melakukan failover ke salah satu klaster sekundernya, integrasi menjadi tidak aktif. Anda harus menghapus dan membuat ulang integrasi.
- Saat Anda pertama kali membuat integrasi, atau ketika tabel sedang disinkronkan ulang, seeding data dari sumber ke target dapat memakan waktu 20-25 menit atau lebih tergantung ukuran basis data sumber. Penundaan ini dapat menyebabkan peningkatan lag replika.
- Beberapa jenis data tidak didukung. Untuk daftar jenis data yang didukung, lihat the section called “Perbedaan jenis data”.
- Referensi kunci asing dengan pembaruan tabel yang telah ditentukan sebelumnya tidak didukung. Secara khusus, `ON DELETE` dan `ON UPDATE` aturan tidak didukung dengan`CASCADE`,`SET NULL`, dan `SET DEFAULT` tindakan. Mencoba membuat atau memperbarui tabel dengan referensi tersebut ke tabel lain akan menempatkan tabel ke dalam keadaan gagal.
- Transaksi XA tidak didukung.
- Pengidentifikasi objek (termasuk nama basis data, nama tabel, nama kolom, dan lainnya) hanya dapat berisi karakter alfanumerik, angka, $, dan _ (garis bawah).

Batasan Aurora MySQL

- Cluster DB sumber Anda harus menjalankan Aurora MySQL versi 3.05 (kompatibel dengan MySQL 8.0.32) atau lebih tinggi.

- Integrasi nol-ETL mengandalkan pencatatan log biner MySQL (binlog) untuk mengambil perubahan data yang sedang berlangsung. Sebaiknya jangan gunakan pemfilteran data berbasis binlog, karena hal ini dapat menyebabkan inkonsistensi data antara basis data sumber dan target.
- Tabel sistem, tabel sementara, dan tampilan Aurora MySQL tidak direplikasi ke Amazon Redshift.
- Integrasi nol-ETL didukung hanya untuk basis data yang dikonfigurasi untuk menggunakan mesin penyimpanan InnoDB.
- Operasi partisi `ALTER TABLE` menyebabkan tabel Anda melakukan sinkronisasi ulang untuk memuat ulang data dari Aurora ke Amazon Redshift. Tabel tidak akan tersedia untuk kueri saat disinkronkan ulang.

Batasan Aurora PostgreSQL pratinjau

> **Important**
>
> Integrasi nol-ETL dengan fitur Amazon Redshift untuk Aurora PostgreSQL sedang dalam rilis pratinjau. Dokumentasi dan fitur dapat berubah. Anda dapat menggunakan fitur ini hanya dalam lingkungan pengujian, bukan dalam lingkungan produksi. Untuk syarat dan ketentuan pratinjau, lihat Beta dan Pratinjau dalam Persyaratan Layanan AWS.

- Klaster DB sumber Anda harus menjalankan Aurora PostgreSQL (kompatibel dengan PostgreSQL 15.4 dan Dukungan Nol-ETL).
- Anda dapat membuat dan mengelola integrasi nol-ETL untuk Aurora PostgreSQL hanya di Lingkungan Pratinjau Database Amazon RDS, di Timur AS (Ohio) (us-timur-2). Wilayah AWS Anda dapat menggunakan lingkungan pratinjau untuk menguji versi beta, versi kandidat rilis, dan versi produksi awal perangkat lunak mesin basis data PostgreSQL.
- Anda dapat membuat dan mengelola integrasi untuk Aurora PostgreSQL hanya menggunakan AWS Management Console. Anda tidak dapat menggunakan AWS Command Line Interface (AWS CLI), Amazon RDS API, atau AWS SDK mana pun.
- Saat Anda membuat klaster DB sumber, grup parameter yang Anda pilih harus sudah mengonfigurasi nilai parameter klaster DB yang diperlukan. Anda tidak dapat membuat grup parameter baru setelahnya lalu mengaitkannya dengan klaster. Untuk daftar parameter yang diperlukan, lihat the section called “Langkah 1: Buat grup parameter klaster DB kustom”.
- Anda tidak dapat memodifikasi integrasi setelah Anda membuatnya. Jika perlu mengubah pengaturan tertentu, Anda harus menghapus dan membuat ulang integrasi.

- Saat ini, klaster DB Aurora PostgreSQL yang merupakan sumber integrasi tidak melakukan pengumpulan sampah data replikasi logis.
- Semua basis data yang dibuat dalam klaster DB Aurora PostgreSQL sumber harus menggunakan pengenkodean UTF-8.
- Nama kolom tidak dapat berisi salah satu karakter berikut: koma (,), titik koma (;), tanda kurung (), kurung kurawal {}, baris baru (\n), tab (\ t), tanda sama dengan (=), dan spasi.
- Integrasi nol-ETL dengan Aurora PostgreSQL tidak mendukung hal berikut ini:
  - Instans DB Aurora Serverless v2. Klaster DB sumber Anda harus menggunakan instans DB terprovisi.
  - Jenis data kustom atau jenis data yang dibuat oleh ekstensi.
  - Subtransaksi pada klaster DB sumber.
  - Mengubah nama skema atau basis data dalam klaster DB sumber.
  - Memulihkan dari snapshot klaster DB atau menggunakan kloning Aurora untuk membuat klaster DB sumber. Jika Anda ingin menambahkan data yang ada ke dalam klaster pratinjau, Anda harus menggunakan utilitas `pg_dump` atau `pg_restore`.
  - Pembuatan slot replikasi logis pada instans penulis dari klaster DB sumber.
  - Nilai bidang besar yang memerlukan The Oversized-Attribute Storage Technique (TOAST).
  - Operasi partisi `ALTER TABLE`. Operasi ini dapat menyebabkan tabel Anda disinkronkan kembali dan akhirnya beralih ke status `Failed`. Jika sebuah tabel gagal, maka Anda harus menghapus dan membuatnya kembali.

Batasan Amazon Redshift

Untuk mengetahui daftar batasan Amazon Redshift yang terkait dengan integrasi nol-ETL, lihat Pertimbangan dalam Panduan Manajemen Amazon Redshift.

Kuota

Akun Anda memiliki kuota berikut yang terkait dengan integrasi nol-ETL Aurora dengan Amazon Redshift. Kecuali ditentukan lain, masing-masing kuota ditentukan untuk setiap Wilayah.

Nama	Default	Deskripsi
Integrasi	100	Jumlah total integrasi dalam sebuah Akun AWS.

Nama	Default	Deskripsi
Integrasi per gudang data target	50	Jumlah integrasi yang mengirim data ke satu gudang data Amazon Redshift target.
Integrasi per klaster sumber	5 untuk Aurora MySQL, 1 untuk Aurora PostgreSQL	Jumlah integrasi yang mengirimkan data dari klaster DB sumber tunggal.

Selain itu, Amazon Redshift menempatkan batasan tertentu pada jumlah tabel yang diizinkan di setiap instans DB atau simpul klaster. Untuk informasi selengkapnya, lihat Kuota dan batasan di Amazon Redshift dalam Panduan Manajemen Amazon Redshift.

Wilayah yang Didukung

Integrasi Aurora Zero-ETL dengan Amazon Redshift tersedia dalam subset. Wilayah AWS Untuk mengetahui daftar Wilayah yang didukung, lihat the section called “Integrasi nol-ETL ”.

Mulai menggunakan integrasi nol-ETL Aurora dengan Amazon Redshift

Sebelum Anda membuat integrasi nol-ETL dengan Amazon Redshift, konfigurasikan DB dan gudang data Amazon Redshift Anda dengan parameter dan izin yang diperlukan. Selama pengaturan, Anda akan menyelesaikan langkah-langkah berikut:

1. Buat grup parameter klaster DB kustom.
2. Buat sumber DB cluster.
3. Buat gudang data Amazon Redshift target.

Setelah Anda menyelesaikan tugas-tugas ini, lanjutkan ke the section called “Membuat integrasi nol-ETL”.

Anda dapat menggunakan AWS SDK untuk mengotomatiskan proses penyiapan untuk Anda. Untuk informasi selengkapnya, lihat the section called “Siapkan integrasi menggunakan AWS SDK (hanya Aurora MySQL)”.

Langkah 1: Buat grup parameter klaster DB kustom

Integrasi nol-ETL Aurora dengan Amazon Redshift memerlukan nilai spesifik untuk parameter klaster DB yang mengontrol replikasi. Secara khusus, Aurora MySQL memerlukan binlog yang ditingkatkan (`aurora_enhanced_binlog`), dan Aurora PostgreSQL memerlukan replikasi logis yang ditingkatkan (`aurora.enhanced_logical_replication`).

Untuk mengonfigurasi pencatatan log biner atau replikasi logis, Anda harus membuat grup parameter klaster DB kustom terlebih dahulu, lalu mengaitkannya dengan klaster DB sumber.

Buat grup parameter klaster DB kustom dengan pengaturan berikut bergantung pada mesin DB sumber Anda. Untuk petunjuk cara membuat grup parameter, lihat the section called “Bekerja dengan grup parameter klaster DB”.

Aurora MySQL (rangkaian aurora-mysql8.0):

- `aurora_enhanced_binlog=1`
- `binlog_backup=0`
- `binlog_format=ROW`
- `binlog_replication_globaldb=0`
- `binlog_row_image=full`
- `binlog_row_metadata=full`

Selain itu, pastikan bahwa parameter `binlog_transaction_compression` tidak ditetapkan ke `ON`, dan bahwa parameter `binlog_row_value_options` tidak diatur ke `PARTIAL_JSON`.

Untuk informasi selengkapnya tentang binlog yang ditingkatkan Aurora MySQL, lihat the section called “Menyiapkan binlog yang ditingkatkan”.

Aurora PostgreSQL (rangkaian aurora-postgresql15):

> **Note**
>
> Untuk klaster DB Aurora PostgreSQL, Anda harus membuat grup parameter kustom dalam Lingkungan Pratinjau Basis Data Amazon RDS, di Wilayah AWS AS Timur (Ohio) (us-east-2).

- `rds.logical_replication=1`
- `aurora.enhanced_logical_replication=1`
- `aurora.logical_replication_backup=0`
- `aurora.logical_replication_globaldb=0`

Mengaktifkan replikasi logis yang disempurnakan (`aurora.enhanced_logical_replication`) akan secara otomatis menetapkan parameter `REPLICA IDENTITY` ke `FULL`. Hal ini berarti bahwa semua nilai kolom ditulis ke write ahead log (WAL). Hal ini akan meningkatkan IOPS untuk klaster DB sumber Anda.

Langkah 2: Buat cluster DB sumber

Setelah Anda membuat grup parameter cluster DB kustom, buat cluster. Cluster ini akan menjadi sumber replikasi data ke Amazon Redshift.

Kluster harus menjalankan , atau Aurora PostgreSQL (kompatibel dengan PostgreSQL 15.4 dan Zero-ETL Support). Untuk petunjuk cara membuat cluster , lihat.

> **Note**
>
> Anda harus membuat klaster DB Aurora PostgreSQL dalam Lingkungan Pratinjau Basis Data Amazon RDS, di Wilayah AWS AS Timur (Ohio) (us-east-2).

Di bagian Konfigurasi tambahan, ubah grup parameter klaster DB default ke grup parameter kustom yang Anda buat pada langkah sebelumnya.

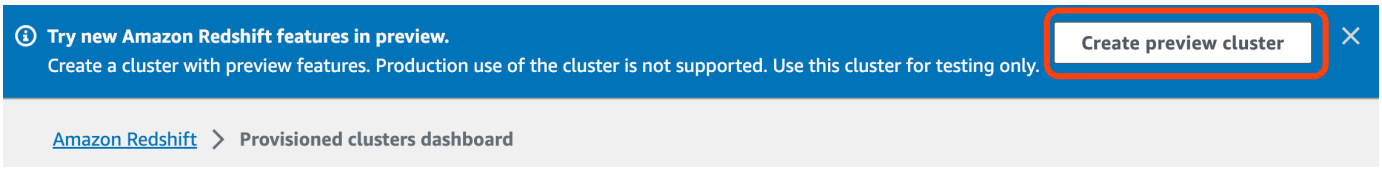
> **Note**
>
> Untuk Aurora MySQL, jika Untuk petunjuk, lihat the section called “Mem-boot ulang instans atau klaster DB Aurora”.

Selama rilis pratinjau integrasi nol-ETL Aurora PostgreSQL dengan Amazon Redshift, Anda harus mengaitkan klaster dengan grup parameter klaster DB kustom saat membuat klaster. Anda tidak dapat melakukan tindakan ini setelah klaster DB sumber sudah dibuat. Jika tidak, Anda harus menghapus dan membuat ulang klaster.

Langkah 3: Buat gudang data Amazon Redshift

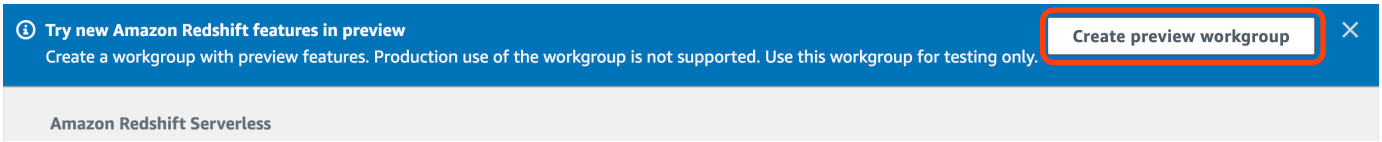
Setelah membuat cluster DB sumber, Anda harus membuat dan mengonfigurasi gudang data target di Amazon Redshift. Gudang data harus memenuhi persyaratan berikut:

- Dibuat dalam pratinjau (hanya untuk sumber Aurora PostgreSQL). Untuk sumber Aurora MySQL, Anda harus membuat klaster produksi dan grup kerja.
  - Untuk membuat klaster terprovisi dalam pratinjau, pilih Buat klaster pratinjau dari banner di dasbor klaster terprovisi. Untuk informasi selengkapnya, lihat Membuat klaster pratinjau.



    Saat membuat klaster, tetapkan Jalur pratinjau ke `preview_2023`.

  - Untuk membuat grup kerja Redshift Nirserver dalam pratinjau, pilih Buat grup kerja pratinjau dari banner di dasbor Nirserver. Untuk informasi selengkapnya, lihat Membuat grup kerja pratinjau.



- Menggunakan jenis simpul RA3 (`ra3.16xlarge`, `ra3.4xlarge`, atau `ra3.xlplus`) dengan setidaknya dua node, atau Redshift Nirserver
- Terenkripsi (jika menggunakan klaster yang disediakan). Untuk informasi selengkapnya, lihat Enkripsi basis data Amazon Redshift.

Untuk petunjuk cara membuat gudang data, lihat Membuat klaster untuk klaster terprovisi, atau Membuat grup kerja dengan ruang nama untuk Redshift Nirserver.

Aktifkan kepekaan huruf besar/kecil di gudang data

Agar integrasi berhasil, parameter kepekaan huruf besar/kecil (`enable_case_sensitive_identifier`) harus diaktifkan untuk gudang data. Secara default, kepekaan huruf besar/kecil dinonaktifkan di semua klaster terprovisi dan grup kerja Redshift Nirserver.

Untuk mengaktifkan kepekaan huruf besar/kecil, lakukan langkah-langkah berikut bergantung pada jenis gudang data Anda:

- Klaster terprovisi – Untuk mengaktifkan kepekaan huruf besar/kecil pada klaster terprovisi, buat grup parameter kustom dengan parameter `enable_case_sensitive_identifier` diaktifkan. Kemudian, hubungkan grup parameter dengan klaster. Untuk petunjuknya, lihat Mengelola grup parameter menggunakan konsol atau Mengonfigurasi nilai parameter menggunakan AWS CLI.

  > **Note**
  >
  > Ingatlah untuk mem-boot ulang klaster setelah Anda mengaitkan grup parameter kustom dengannya.

- Grup kerja Nirserver – Untuk mengaktifkan kepekaan huruf besar/kecil di grup kerja Redshift Nirserver, Anda harus menggunakan AWS CLI. Konsol Amazon Redshift saat ini tidak mendukung modifikasi nilai parameter Redshift Nirserver. Kirim permintaan update-workgroup berikut:

  ```
  aws redshift-serverless update-workgroup \
    --workgroup-name target-workgroup \
    --config-parameters
   parameterKey=enable_case_sensitive_identifier,parameterValue=true
  ```

  Anda tidak perlu mem-boot ulang grup kerja setelah Anda mengubah nilai parameternya.

Konfigurasikan otorisasi untuk gudang data

Setelah Anda membuat gudang data, Anda harus mengkonfigurasi sumber Aurora DB cluster sebagai sumber integrasi resmi. Untuk petunjuknya, lihat Mengonfigurasi otorisasi untuk gudang data Amazon Redshift Anda.

Siapkan integrasi menggunakan AWS SDK (hanya Aurora MySQL)

Daripada menyiapkan setiap sumber daya secara manual, Anda dapat menjalankan skrip Python berikut untuk menyiapkan sumber daya yang diperlukan secara otomatis. Contoh kode ini menggunakan AWS SDK for Python (Boto3) untuk membuat klaster DB Aurora MySQL dan gudang data Amazon Redshift target, masing-masing dengan nilai parameter yang diperlukan. Kemudian, kode ini akan menunggu klaster tersedia sebelum membuat integrasi nol-ETL di antara klaster. Anda dapat menerapkan "comment out" pada berbagai fungsi tergantung pada sumber daya yang perlu Anda atur.

Untuk menginstal dependensi yang diperlukan, jalankan perintah berikut:

```
pip install boto3
pip install time
```

Dalam skrip, secara opsional, Anda dapat memodifikasi nama grup sumber, target, dan parameter. Fungsi akhir akan membuat integrasi bernama `my-integration` setelah sumber daya disiapkan.

Contoh kode Python

```
import boto3
import time

# Build the client using the default credential configuration.
# You can use the CLI and run 'aws configure' to set access key, secret
# key, and default Region.

rds = boto3.client('rds')
redshift = boto3.client('redshift')
sts = boto3.client('sts')

source_cluster_name = 'my-source-cluster' # A name for the source cluster
source_param_group_name = 'my-source-param-group' # A name for the source parameter
 group
target_cluster_name = 'my-target-cluster' # A name for the target cluster
target_param_group_name = 'my-target-param-group' # A name for the target parameter
 group

def create_source_cluster(*args):
    """Creates a source Aurora MySQL DB cluster"""

    response = rds.create_db_cluster_parameter_group(
```

```
        DBClusterParameterGroupName=source_param_group_name,
        DBParameterGroupFamily='aurora-mysql8.0',
        Description='For Aurora MySQL zero-ETL integrations'
    )
    print('Created source parameter group: ' + response['DBClusterParameterGroup']
['DBClusterParameterGroupName'])

    response = rds.modify_db_cluster_parameter_group(
        DBClusterParameterGroupName=source_param_group_name,
        Parameters=[
            {
                'ParameterName': 'aurora_enhanced_binlog',
                'ParameterValue': '1',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_backup',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_format',
                'ParameterValue': 'ROW',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_replication_globaldb',
                'ParameterValue': '0',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_row_image',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
            },
            {
                'ParameterName': 'binlog_row_metadata',
                'ParameterValue': 'full',
                'ApplyMethod': 'pending-reboot'
            }
        ]
    )
    print('Modified source parameter group: ' +
 response['DBClusterParameterGroupName'])
```

```
    response = rds.create_db_cluster(
        DBClusterIdentifier=source_cluster_name,
        DBClusterParameterGroupName=source_param_group_name,
        Engine='aurora-mysql',
        EngineVersion='8.0.mysql_aurora.3.05.2',
        DatabaseName='myauroradb',
        MasterUsername='username',
        MasterUserPassword='Password01**'
    )
    print('Creating source cluster: ' + response['DBCluster']['DBClusterIdentifier'])
    source_arn = (response['DBCluster']['DBClusterArn'])
    create_target_cluster(target_cluster_name, source_arn, target_param_group_name)

    response = rds.create_db_instance(
        DBInstanceClass='db.r6g.2xlarge',
        DBClusterIdentifier=source_cluster_name,
        DBInstanceIdentifier=source_cluster_name + '-instance',
        Engine='aurora-mysql'
    )
    return(response)

def create_target_cluster(target_cluster_name, source_arn, target_param_group_name):
    """Creates a target Redshift cluster"""

    response = redshift.create_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        ParameterGroupFamily='redshift-1.0',
        Description='For Aurora MySQL zero-ETL integrations'
    )
    print('Created target parameter group: ' + response['ClusterParameterGroup']
['ParameterGroupName'])

    response = redshift.modify_cluster_parameter_group(
        ParameterGroupName=target_param_group_name,
        Parameters=[
            {
                'ParameterName': 'enable_case_sensitive_identifier',
                'ParameterValue': 'true'
            }
        ]
    )
    print('Modified target parameter group: ' + response['ParameterGroupName'])
```

```
    response = redshift.create_cluster(
        ClusterIdentifier=target_cluster_name,
        NodeType='ra3.4xlarge',
        NumberOfNodes=2,
        Encrypted=True,
        MasterUsername='username',
        MasterUserPassword='Password01**',
        ClusterParameterGroupName=target_param_group_name
    )
    print('Creating target cluster: ' + response['Cluster']['ClusterIdentifier'])

    # Retrieve the target cluster ARN
    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name
    )
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Retrieve the current user's account ID
    response = sts.get_caller_identity()
    account_id = response['Account']

    # Create a resource policy specifying cluster ARN and account ID
    response = redshift.put_resource_policy(
        ResourceArn=target_arn,
        Policy='''
        {
            \"Version\":\"2012-10-17\",
            \"Statement\":[
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"Service\":\"redshift.amazonaws.com\"
                },
                \"Action\":[\"redshift:AuthorizeInboundIntegration\"],
                \"Condition\":{
                    \"StringEquals\":{
                        \"aws:SourceArn\":\"%s\"}
                    }
                },
                {\"Effect\":\"Allow\",
                \"Principal\":{
                    \"AWS\":\"arn:aws:iam::%s:root\"},
                \"Action\":\"redshift:CreateInboundIntegration\"}
            ]
        }
```

```
        ''' % (source_arn, account_id)
    )
    return(response)

def wait_for_cluster_availability(*args):
    """Waits for both clusters to be available"""

    print('Waiting for clusters to be available...')

    response = rds.describe_db_clusters(
        DBClusterIdentifier=source_cluster_name,
    )
    source_status = response['DBClusters'][0]['Status']
    source_arn = response['DBClusters'][0]['DBClusterArn']

    response = rds.describe_db_instances(
        DBInstanceIdentifier=source_cluster_name + '-instance',
    )
    source_instance_status = response['DBInstances'][0]['DBInstanceStatus']

    response = redshift.describe_clusters(
        ClusterIdentifier=target_cluster_name,
    )
    target_status = response['Clusters'][0]['ClusterStatus']
    target_arn = response['Clusters'][0]['ClusterNamespaceArn']

    # Every 60 seconds, check whether the clusters are available.
    if source_status != 'available' or target_status != 'available' or
 source_instance_status != 'available':
        time.sleep(60)
        response = wait_for_cluster_availability(
            source_cluster_name, target_cluster_name)
    else:
        print('Clusters available. Ready to create zero-ETL integration.')
        create_integration(source_arn, target_arn)
        return

def create_integration(source_arn, target_arn):
    """Creates a zero-ETL integration using the source and target clusters"""

    response = rds.create_integration(
        SourceArn=source_arn,
        TargetArn=target_arn,
        IntegrationName='my-integration'
```

```
    )
    print('Creating integration: ' + response['IntegrationName'])

def main():
    """main function"""
    create_source_cluster(source_cluster_name, source_param_group_name)
    wait_for_cluster_availability(source_cluster_name, target_cluster_name)

if __name__ == "__main__":
    main()
```

Langkah selanjutnya

Dengan sumber cluster Aurora DB dan gudang data target Amazon Redshift, Anda sekarang dapat membuat integrasi nol-ETL dan mereplikasi data. Untuk petunjuk, lihat the section called “Membuat integrasi nol-ETL”.

Membuat integrasi nol-ETL Aurora dengan Amazon Redshift

Saat membuat integrasi Aurora Zero-ETL, Anda menentukan cluster Aurora DB instans dan gudang data Amazon Redshift target. Anda juga dapat menyesuaikan pengaturan enkripsi dan menambahkan tag. Aurora menciptakan integrasi antara cluster DB database dan targetnya. Setelah integrasi aktif, data apa pun yang Anda masukkan ke dalam cluster DB sumber akan direplikasi ke target Amazon Redshift yang dikonfigurasi.

Topik

- Prasyarat
- Izin yang diperlukan
- Membuat integrasi nol-ETL
- Langkah selanjutnya

Prasyarat

Sebelum membuat integrasi Nol-ETL, Anda harus membuat cluster sumber dan gudang data Amazon Redshift target. Anda juga harus mengizinkan replikasi ke gudang data dengan menambahkan cluster DB sebagai sumber integrasi resmi.

Untuk petunjuk cara menyelesaikan setiap langkah ini, lihat the section called “Mulai menggunakan integrasi nol-ETL”.

Izin yang diperlukan

Izin IAM tertentu diperlukan untuk membuat integrasi nol-ETL. Setidaknya, Anda memerlukan izin untuk melakukan tindakan berikut:

-
- Lihat dan hapus semua integrasi nol-ETL.
- Buat integrasi masuk ke gudang data target. Anda tidak memerlukan izin ini jika akun yang sama memiliki gudang data Amazon Redshift dan akun ini merupakan prinsipal yang diotorisasi untuk gudang data tersebut. Untuk informasi tentang menambahkan prinsipal resmi, lihat Mengonfigurasi otorisasi untuk gudang data Amazon Redshift Anda.

Contoh kebijakan berikut menunjukkan izin hak akses paling rendah yang diperlukan untuk membuat dan mengelola integrasi. Anda mungkin tidak memerlukan izin persis ini jika pengguna atau peran Anda memiliki izin yang lebih luas, seperti kebijakan terkelola `AdministratorAccess`.

> **Note**
>
> Amazon Resource Name (ARN) Redshift memiliki format berikut ini. Perhatikan penggunaan garis miring ke depan (`/`), bukan titik dua (`:`), sebelum UUID ruang nama nirserver.
>
> - Klaster terprovisi – `arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid`
> - Nirserver – `arn:aws:redshift-serverless:{region}:{account-id}:namespace/namespace-uuid`

Contoh kebijakan

> **Important**
>
> Untuk pratinjau PostgreSQL Aurora, semua ARN dan tindakan dalam Lingkungan Pratinjau Basis Data Amazon RDS akan ditambahi `-preview` pada ruang nama layanannya. Misalnya, `rds-preview:CreateIntegration` dan `arn:aws:rds-preview:...`.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "rds:CreateIntegration"
        ],
        "Resource": [
            "arn:aws:rds:{region}:{account-id}:cluster:source-db",
            "arn:aws:rds:{region}:{account-id}:integration:*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "rds:DescribeIntegrations"
        ],
        "Resource": ["*"]
    },
    {
        "Effect": "Allow",
        "Action": [
            "rds:DeleteIntegration",
            "rds:ModifyIntegration"
        ],
        "Resource": [
            "arn:aws:rds:{region}:{account-id}:integration:*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "redshift:CreateInboundIntegration"
        ],
        "Resource": [
            "arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid"
        ]
    }]
}
```

Memilih gudang data target di akun yang berbeda

Jika Anda berencana untuk menentukan gudang data Amazon Redshift target yang ada di gudang lain Akun AWS, Anda harus membuat peran yang memungkinkan pengguna di akun saat ini mengakses sumber daya di akun target. Untuk informasi selengkapnya, lihat Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki.

Peran harus memiliki izin berikut, yang memungkinkan pengguna melihat klaster terprovisi Amazon Redshift dan ruang nama Redshift Nirserver di akun target.

Izin dan kebijakan kepercayaan yang diperlukan

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "redshift:DescribeClusters",
            "redshift-serverless:ListNamespaces"
         ],
         "Resource":[
            "*"
         ]
      }
   ]
}
```

Peran harus memiliki kebijakan kepercayaan berikut ini, yang menentukan ID akun target.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "AWS": "arn:aws:iam::{external-account-id}:root"
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

Untuk petunjuk cara membuat peran, lihat Membuat peran menggunakan kebijakan kepercayaan kustom.

Membuat integrasi nol-ETL

Anda dapat membuat integrasi nol-ETL Aurora MySQL menggunakan AWS Management Console, AWS CLI, atau API RDS. Untuk membuat integrasi Aurora PostgreSQL, Anda harus menggunakan AWS Management Console.

Konsol

Untuk membuat integrasi nol-ETL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

   Jika Anda menggunakan klaster DB Aurora PostgreSQL sebagai sumber integrasi, Anda harus masuk ke Lingkungan Pratinjau Basis Data Amazon RDS di https://us-east-2.console.aws.amazon.com/rds-preview/home?region=us-east-2#databases.
2. Di panel navigasi kiri, pilih Integrasi nol-ETL.
3. Pilih Buat integrasi nol-ETL.
4. Untuk Pengidentifikasi integrasi, masukkan nama untuk integrasi. Nama dapat memiliki maksimal 63 karakter alfanumerik dan dapat menyertakan tanda hubung.
5. Pilih Selanjutnya.
6. Untuk Sumber, pilih cluster DB tempat datanya berasal. Cluster harus menjalankan , atau Aurora PostgreSQL (kompatibel dengan PostgreSQL 15.4 dan Dukungan Zero-ETL).

> **Note**
>
> Untuk sumber MySQL, RDS memberi tahu Anda jika parameter klaster DB tidak dikonfigurasi dengan benar. Jika Anda menerima pesan ini, Anda dapat memilih Perbaiki untuk saya, atau mengonfigurasinya secara manual. Untuk petunjuk cara memperbaikinya secara manual, lihat the section called “Langkah 1: Buat grup parameter klaster DB kustom”.
> Memodifikasi parameter klaster DB membutuhkan boot ulang. Sebelum Anda dapat membuat integrasi, reboot harus lengkap dan nilai parameter baru harus berhasil diterapkan ke cluster .

7. Jika Anda memilih klaster sumber Aurora PostgreSQL, di bagian Basis data bernama, tentukan basis data bernama yang akan digunakan sebagai sumber integrasi Anda. Model sumber daya PostgreSQL memungkinkan pembuatan beberapa basis data dalam satu klaster DB, tetapi hanya satu yang dapat digunakan untuk setiap integrasi nol-ETL.

   Basis data bernama harus dibuat dari `template1`. Untuk informasi selengkapnya, lihat Template Databases dalam dokumentasi PostgreSQL.

8. (Opsional) Jika Anda memilih cluster DB sumber MySQL Aurora, pilih Sesuaikan opsi pemfilteran data dan tambahkan filter data ke integrasi Anda. Anda dapat menggunakan filter data untuk menentukan ruang lingkup replikasi ke gudang data target. Untuk informasi selengkapnya, lihat the section called “Pemfilteran data untuk integrasi nol-ETL”.

9. Setelah cluster DB sumber Anda berhasil dikonfigurasi, pilih Berikutnya.

10. Untuk Target, lakukan hal berikut:

    1. (Opsional) Untuk menggunakan target Amazon Redshift yang berbeda Akun AWS , pilih Tentukan akun yang berbeda. Kemudian, masukkan ARN peran IAM dengan izin untuk menampilkan gudang data Anda. Untuk petunjuk cara membuat peran IAM, lihat the section called “Memilih gudang data target di akun yang berbeda”.
    2. Untuk gudang data Amazon Redshift, pilih target untuk data yang direplikasi dari cluster DB sumber. Anda dapat memilih klaster Amazon Redshift terprovisi atau ruang nama Redshift Nirserver sebagai target.

> **Note**
>
> RDS akan memberi tahu Anda jika pengaturan kebijakan sumber daya atau kepekaan huruf besar/kecil untuk gudang data yang ditentukan tidak dikonfigurasi dengan benar. Jika Anda menerima pesan ini, Anda dapat memilih Perbaiki untuk saya, atau mengonfigurasinya secara manual. Untuk petunjuk cara memperbaikinya secara manual, lihat Mengaktifkan kepekaan huruf besar/kecil untuk gudang data Anda dan Mengonfigurasi otorisasi untuk gudang data Anda dalam Panduan Manajemen Amazon Redshift.
> Modifikasi kepekaan huruf besar/kecil untuk klaster Redshift terprovisi memerlukan boot ulang. Sebelum Anda dapat membuat integrasi, boot ulang harus selesai dan nilai parameter baru harus berhasil diterapkan ke klaster.

Jika sumber dan target yang Anda pilih berada di Akun AWS yang berbeda, maka Amazon RDS tidak dapat memperbaiki pengaturan ini untuk Anda. Anda harus menavigasi ke akun lain dan memperbaikinya secara manual di Amazon Redshift.

11. Setelah gudang data target Anda dikonfigurasi dengan benar, pilih Berikutnya.
12. (Opsional) Untuk Tag, tambahkan satu atau beberapa tag ke integrasi. Untuk informasi selengkapnya, lihat the section called “Memberi tag pada sumber daya RDS”.
13. Untuk Enkripsi, tentukan cara enkripsi integrasi Anda. Secara default, RDS mengenkripsi semua integrasi dengan file. Kunci milik AWS Untuk memilih kunci yang dikelola pelanggan, aktifkan Sesuaikan pengaturan enkripsi dan pilih kunci KMS yang akan digunakan untuk enkripsi. Untuk informasi selengkapnya, lihat the section called “Mengenkripsi sumber daya Amazon Aurora”.

    > **Note**
    >
    > Jika Anda menentukan kunci KMS kustom, kebijakan kunci harus mengizinkan tindakan `kms:CreateGrant` untuk prinsipal layanan Amazon Redshift (`redshift.amazonaws.com`). Untuk informasi selengkapnya, lihat Membuat kebijakan kunci di Panduan Developer AWS Key Management Service .

    Secara opsional, tambahkan konteks enkripsi. Untuk informasi lebih lanjut, lihat Konteks enkripsi di Panduan Developer AWS Key Management Service .
14. Pilih Selanjutnya.
15. Tinjau pengaturan integrasi Anda dan pilih Buat integrasi nol-ETL. Dibutuhkan sekitar 30 menit agar integrasi menjadi aktif.

    Jika pembuatan gagal, lihat the section called “Saya tidak dapat membuat integrasi nol-ETL” untuk langkah-langkah pemecahan masalah.

Integrasi memiliki status `Creating` ketika sedang dibuat dan gudang data Amazon Redshift target memiliki status `Modifying`. Selama waktu ini, Anda tidak dapat mengueri gudang data atau membuat perubahan konfigurasi apa pun di dalamnya.

Ketika integrasi berhasil dibuat, status integrasi dan gudang data Amazon Redshift target berubah menjadi `Active`.

AWS CLI

> **Note**
>
> Selama pratinjau integrasi nol-ETL Aurora PostgreSQL, Anda hanya dapat membuat integrasi melalui AWS Management Console. Anda tidak dapat menggunakan AWS CLI, Amazon RDS API, atau SDK mana pun.

Untuk membuat integrasi nol-ETL menggunakan AWS CLI, gunakan perintah create-integration dengan opsi berikut:

- `--integration-name` – Tentukan nama untuk integrasi.
- `--source-arn`— Tentukan ARN dari cluster Aurora Multi-AZ DB yang akan menjadi sumber integrasi.
- `--target-arn` – Tentukan ARN gudang data Amazon Redshift yang akan menjadi target integrasi.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-integration \
    --integration-name my-integration \
    --source-arn arn:aws:rds:{region}:{account-id}:my-cluster \
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

Untuk Windows:

```
aws rds create-integration ^
    --integration-name my-integration ^
    --source-arn arn:aws:rds:{region}:{account-id}:my-cluster ^
    --target-arn arn:aws:redshift:{region}:{account-id}:namespace:namespace-uuid
```

API RDS

> **Note**
>
> Selama pratinjau integrasi nol-ETL Aurora PostgreSQL, Anda hanya dapat membuat integrasi melalui AWS Management Console. Anda tidak dapat menggunakan AWS CLI, Amazon RDS API, atau SDK mana pun.

Untuk membuat integrasi nol-ETL menggunakan API Amazon RDS, gunakan operasi `CreateIntegration` dengan parameter berikut ini:

- `IntegrationName` – Tentukan nama untuk integrasi.
- `SourceArn`— Tentukan ARN dari cluster Aurora Multi-AZ DB yang akan menjadi sumber integrasi.
- `TargetArn` – Tentukan ARN gudang data Amazon Redshift yang akan menjadi target integrasi.

Langkah selanjutnya

Setelah berhasil membuat integrasi nol-ETL, Anda harus membuat basis data tujuan dalam klaster atau grup kerja Amazon Redshift target Anda. Kemudian, Anda dapat mulai menambahkan data ke sumber cluster Aurora DB dan menanyakannya di Amazon Redshift. Untuk petunjuknya, lihat Membuat basis data tujuan di Amazon Redshift.

Pemfilteran data untuk integrasi Zero-ETL dengan Amazon Redshift

Anda dapat menggunakan pemfilteran data untuk integrasi Zero-ETL untuk menentukan ruang lingkup replikasi dari sumber basis data Amazon Redshift target. Daripada mereplikasi semua data ke target, Anda dapat menentukan satu atau lebih filter yang secara selektif menyertakan atau mengecualikan tabel tertentu agar tidak direplikasi. Hanya penyaringan di tingkat database dan tabel yang tersedia untuk integrasi nol-ETL. Anda tidak dapat memfilter berdasarkan kolom atau baris.

Pemfilteran data dapat berguna ketika Anda ingin:

- Bergabunglah dengan tabel tertentu dari dua cluster sumber yang berbeda dan Anda tidak memerlukan data lengkap dari kedua cluster.

- Menghemat biaya dengan melakukan analitik hanya menggunakan subset tabel daripada seluruh armada database.
- Saring informasi sensitif—seperti nomor telepon, alamat, atau detail kartu kredit—dari tabel tertentu.

Anda dapat menambahkan filter data ke integrasi nol-ETL menggunakan, AWS Command Line Interface (AWS CLI) AWS Management Console, atau Amazon RDS API.

Jika integrasi memiliki cluster Amazon Redshift yang disediakan sebagai targetnya, cluster harus berada di patch 180 atau lebih tinggi.

> **Note**
>
> Saat ini, Anda dapat melakukan pemfilteran data hanya pada integrasi yang memiliki sumber Aurora MySQL. Rilis pratinjau integrasi Aurora PostgreSQL Zero-ETL dengan Amazon Redshift tidak mendukung pemfilteran data.

Topik

- Format filter data
- Filter logika
- Filter prioritas
- Contoh-contoh
- Menambahkan filter data ke integrasi
- Menghapus filter data dari integrasi

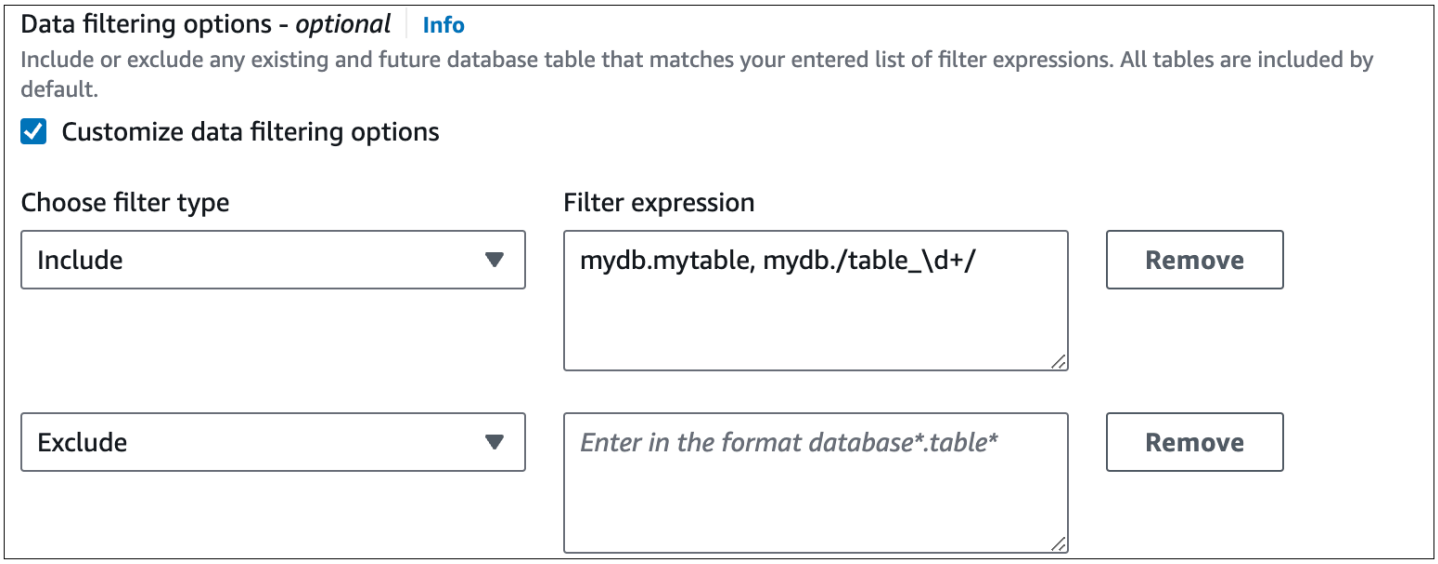
Format filter data

Anda dapat menentukan beberapa filter untuk satu integrasi. Setiap filter menyertakan atau mengecualikan tabel database yang ada dan yang akan datang yang cocok dengan salah satu pola dalam ekspresi filter. Integrasi Aurora Zero-ETL menggunakan sintaks filter Maxwell untuk pemfilteran data.

Setiap filter memiliki elemen-elemen berikut:

Elemen	Deskripsi
Jenis filter	Jenis `Include` filter mencakup semua tabel yang cocok dengan salah satu pola dalam ekspresi filter. Jenis `Exclude` filter mengecualikan semua tabel yang cocok dengan salah satu pola.
Ekspresi filter	Daftar pola yang dipisahkan koma. Ekspresi harus menggunakan sintaks filter Maxwell.
Pola	Pola filter dalam format*`database`*`*.`*`table`*`*`. Anda dapat menentukan database literal dan nama tabel (seperti`mydb.mytable` , atau menggunakan wildcard (*). Anda juga dapat menentukan ekspresi reguler dalam database dan nama tabel. Aurora mendukung pemfilteran hanya pada tingkat database dan tabel. Anda tidak dapat menyertakan filter tingkat kolom (`database.table.column`) atau daftar hitam (). `blacklist: bad_db.*` Integrasi tunggal dapat memiliki maksimum 99 pola total. Di konsol, Anda dapat berisi pola dalam satu ekspresi filter, atau menyebarkannya di antara beberapa ekspresi. Pola tunggal tidak dapat melebihi 256 karakter panjangnya.

Gambar berikut menunjukkan struktur filter data di konsol:



> **Important**
>
> Jangan sertakan informasi identitas pribadi, rahasia, atau sensitif dalam pola filter Anda.

Filter data di AWS CLI

Saat menggunakan AWS CLI untuk menambahkan filter data, sintaksnya sedikit berbeda dibandingkan dengan konsol. Setiap pola individu harus dikaitkan dengan jenis filternya sendiri (`Include`atau`Exclude`). Anda tidak dapat mengelompokkan beberapa pola dengan satu jenis filter.

Misalnya, di konsol Anda dapat mengelompokkan pola dipisahkan koma berikut dalam satu `Include` pernyataan:

```
mydb.mytable, mydb./table_\d+/
```

Namun, saat menggunakan AWS CLI, filter data yang sama harus dalam format berikut:

```
'include: mydb.mytable, include: mydb./table_\d+/'
```

Filter logika

Jika Anda tidak menentukan filter data apa pun dalam integrasi Anda, Aurora mengasumsikan filter `include:*.*` default dan mereplikasi semua tabel ke gudang data target. Namun, jika Anda menentukan setidaknya satu filter, logika dimulai dengan asumsi`exclude:*.*`, yang berarti bahwa

semua tabel secara otomatis dikecualikan dari replikasi. Ini memungkinkan Anda untuk secara langsung menentukan tabel dan database mana yang akan disertakan.

Misalnya, jika Anda menentukan filter berikut:

```
'include: table1, include: table2'
```

Aurora mengevaluasi filter sebagai berikut:

```
'exclude:*.*, include: table1, include: table2'
```

Oleh karena itu, hanya `table1` dan `table2` direplikasi ke gudang data target.

Filter prioritas

Aurora mengevaluasi filter data dalam urutan yang ditentukan. Dalam AWS Management Console, ini berarti bahwa Aurora mengevaluasi ekspresi filter dari kiri ke kanan dan dari atas ke bawah. Jika Anda menentukan pola tertentu untuk filter pertama, maka filter kedua atau bahkan pola individual yang ditentukan segera setelah itu dapat menimpanya.

Misalnya, filter pertama Anda mungkin `Includebooks.stephenking`, yang mencakup satu tabel bernama `stephenking` dari dalam `books` database. Namun, jika Anda menambahkan filter kedua `Excludebooks.*`, itu akan menggantikan `Include` filter yang ditentukan sebelumnya. Dengan demikian, tidak ada tabel dari `books` indeks yang direplikasi ke Amazon Redshift.

Jika Anda menentukan setidaknya satu filter, logika dimulai dengan asumsi`exclude:*.*`, yang berarti bahwa semua tabel secara otomatis dikecualikan dari replikasi. Oleh karena itu, sebagai praktik terbaik umum, tentukan filter Anda dari yang paling luas hingga yang paling tidak luas. Misalnya, gunakan satu atau beberapa `Include` pernyataan untuk menentukan semua data yang ingin Anda replikasi. Kemudian, mulailah menambahkan `Exclude` filter untuk secara selektif mengecualikan tabel tertentu agar tidak direplikasi.

Prinsip yang sama berlaku untuk filter yang Anda definisikan menggunakan AWS CLI. Aurora mengevaluasi pola filter ini dalam urutan yang ditentukan, jadi sebuah pola mungkin mengganti pola yang ditentukan sebelumnya.

Contoh-contoh

Contoh berikut menunjukkan cara kerja penyaringan tingkat tabel untuk integrasi nol-ETL:

- Sertakan semua database dan semua tabel:

```
'include: *.*'
```

- Sertakan semua tabel dalam `books` database:

```
'include: books.*'
```

- Sertakan dua tabel spesifik dalam `books` database:

```
'include: books.stephen_king, include: books.carolyn_keene'
```

- Sertakan semua tabel dalam `books` database, kecuali yang berisi kata-kata `horror` atau`mystery`:

```
'include: books.*, exclude: books./horror|mystery/'
```

- Sertakan semua database yang namanya berisi angka, kecuali yang berisi bilangan bulat antara 2 dan 35 (inklusif). Kecualikan semua tabel dalam database tersebut:

```
'include: /\d/.*, exclude: /[2-9]|[12]\d|3[0-5]/.*'
```

- Sertakan semua tabel dalam `books` database yang dimulai dengan`table_`, kecuali yang bernama`table_stephen_king`:

```
'include: books./^table_.*/, exclude: books.table_stephen_king'
```

Menambahkan filter data ke integrasi

Anda dapat mengonfigurasi pemfilteran data menggunakan AWS Management Console, API AWS CLI, atau Amazon RDS. Jika Anda menambahkan filter setelah membuat integrasi, Aurora mengevaluasi kembali filter seolah-olah selalu ada. Aurora menghapus data apa pun yang saat ini ada di gudang data Amazon Redshift target yang tidak sesuai dengan kriteria pemfilteran baru dari gudang data.

Saat ini, Anda hanya dapat melakukan pemfilteran data pada integrasi yang memiliki sumber Aurora MySQL. Rilis pratinjau integrasi Aurora PostgreSQL Zero-ETL dengan Amazon Redshift tidak mendukung pemfilteran data.

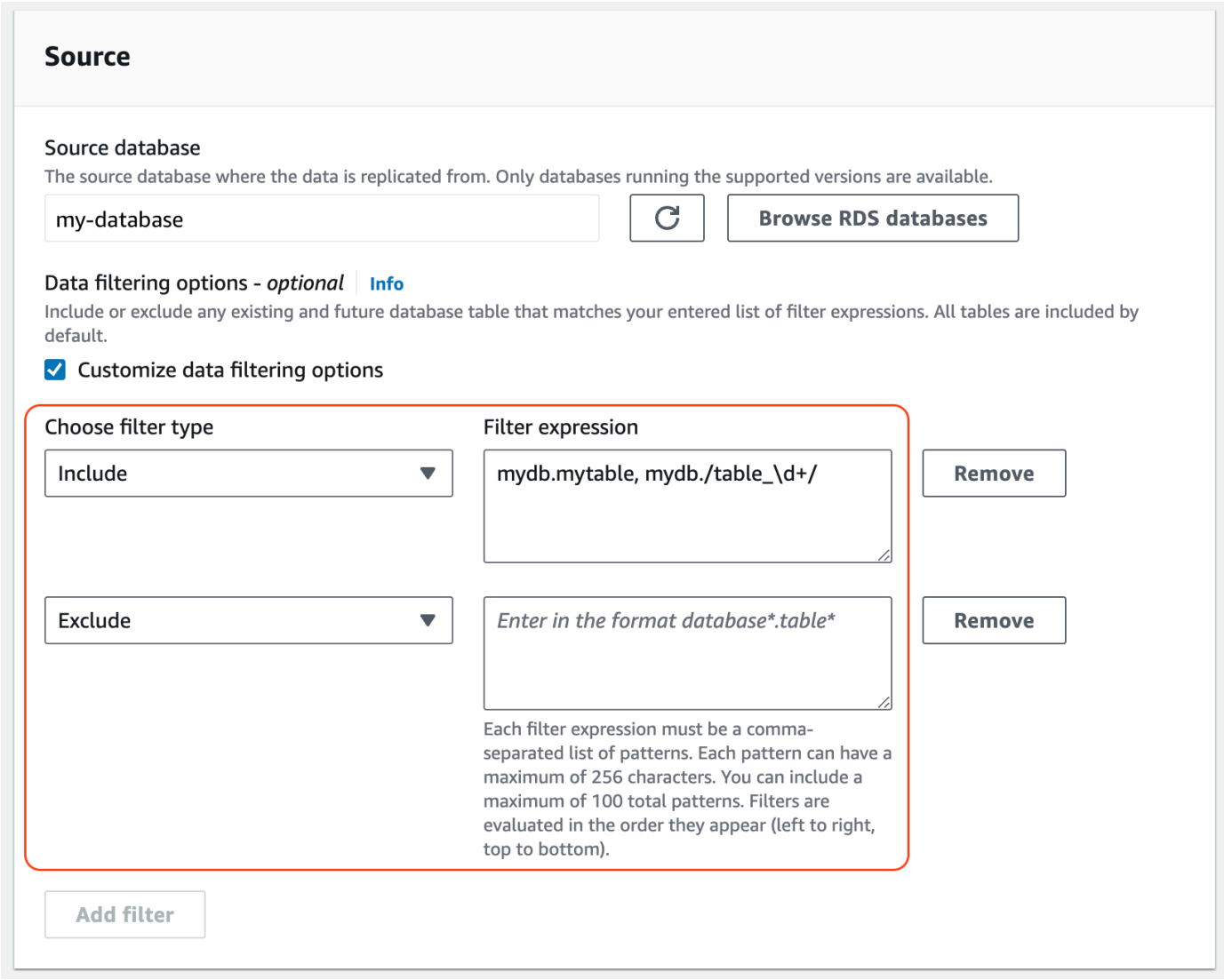
Jika integrasi memiliki cluster Amazon Redshift yang disediakan sebagai targetnya, cluster harus berada di patch 180 atau lebih tinggi.

Konsol

Untuk menambahkan filter data ke integrasi nol-ETL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih integrasi nol-ETL. Pilih integrasi yang ingin Anda tambahkan filter data, lalu pilih Ubah.
3. Di bawah Sumber, tambahkan satu atau lebih `Include` dan `Exclude` pernyataan.

   Gambar berikut menunjukkan contoh filter data untuk integrasi:



4. Ketika semua perubahan sesuai keinginan Anda, pilih Lanjutkan dan Simpan perubahan.

AWS CLI

Untuk menambahkan filter data ke integrasi nol-ETL menggunakan AWS CLI, panggil perintah modify-integration. Selain pengidentifikasi integrasi, tentukan `--data-filter` parameter dengan daftar filter Maxwell yang dipisahkan koma. `Include` `Exclude`

Example

Contoh berikut menambahkan pola filter kemy`-integration`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-integration \
    --integration-identifier my-integration \
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

Untuk Windows:

```
aws rds modify-integration ^
    --integration-identifier my-integration ^
    --data-filter 'include: foodb.*, exclude: foodb.tbl, exclude: foodb./table_\d+/'
```

API RDS

Untuk memodifikasi integrasi nol-ETL menggunakan RDS API, panggil operasi. ModifyIntegration Tentukan pengidentifikasi integrasi dan berikan daftar pola filter yang dipisahkan koma.

Menghapus filter data dari integrasi

Saat Anda menghapus filter data dari integrasi, Aurora mengevaluasi kembali filter yang tersisa seolah-olah filter yang dihapus tidak pernah ada. Aurora kemudian mereplikasi data apa pun yang sebelumnya tidak cocok dengan kriteria pemfilteran (tetapi sekarang) ke dalam gudang data Amazon Redshift target.

Menambahkan data ke sumber cluster Aurora DB dan menanyakannya di Amazon Redshift

Untuk menyelesaikan pembuatan integrasi nol-ETL yang mereplikasi data dari Amazon Aurora ke dalam Amazon Redshift, Anda harus membuat basis data tujuan di Amazon Redshift.

Pertama, hubungkan ke klaster atau grup kerja Amazon Redshift Anda dan buat basis data dengan referensi ke pengidentifikasi integrasi Anda. Kemudian, Anda dapat menambahkan data ke sumber Anda Aurora DB cluster dan melihatnya direplikasi di Amazon Redshift.

Topik

- Buat basis data tujuan di Amazon Redshift
- Tambahkan data ke sumber DB cluster
- Kueri data Anda di Amazon Redshift
- Perbedaan jenis data antara basis data Aurora dan Amazon Redshift

Buat basis data tujuan di Amazon Redshift

Sebelum Anda dapat mulai mereplikasi data ke Amazon Redshift, setelah Anda membuat integrasi, Anda harus membuat basis data tujuan di gudang data target Anda. Basis data tujuan ini harus menyertakan referensi ke pengidentifikasi integrasi. Anda dapat menggunakan konsol Amazon Redshift atau Editor kueri v2 untuk membuat basis data.

Untuk petunjuk cara membuat basis data tujuan, lihat Membuat basis data tujuan di Amazon Redshift.

Tambahkan data ke sumber DB cluster

Tambahkan beberapa data ke Kluster Aurora DB yang ingin Anda tiru ke gudang data Amazon Redshift Anda.

> **Note**
>
> Ada perbedaan antara jenis data di Amazon Aurora dan Amazon Redshift. Untuk tabel pemetaan jenis data, lihat the section called “Perbedaan jenis data”.

Pertama, sambungkan ke cluster DB sumber menggunakan klien MySQL atau PostgreSQL pilihan Anda. Untuk petunjuk, lihat the section called “Menghubungkan ke klaster DB”.

Kemudian, buat tabel dan masukkan urutan data sampel.

> **Important**
>
> Pastikan tabel memiliki kunci primer. Jika tidak, tabel tidak dapat direplikasi ke gudang data target.

Utilitas pg_dump dan pg_restore PostgreSQL awalnya membuat tabel tanpa kunci utama dan kemudian menambahkannya setelahnya. Jika Anda menggunakan salah satu utilitas ini, sebaiknya buat skema terlebih dahulu, lalu muat data dalam perintah terpisah.

MySQL

Contoh berikut menggunakan utilitas MySQL Workbench.

```
CREATE DATABASE my_db;

USE my_db;

CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL, Author
 VARCHAR(50) NOT NULL,
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

INSERT INTO books_table VALUES (1, 'The Shining', 'Stephen King', 1977, 'Supernatural
 fiction');
```

PostgreSQL

Contoh berikut menggunakan terminal interaktif PostgreSQL `psql`. Saat menghubungkan ke klaster, sertakan basis data bernama yang Anda tentukan saat membuat integrasi.

```
psql -h mycluster.cluster-123456789012.us-east-2.rds.amazonaws.com -p 5432 -U username
 -d named_db;

named_db=> CREATE TABLE books_table (ID int NOT NULL, Title VARCHAR(50) NOT NULL,
 Author VARCHAR(50) NOT NULL,
```

```
Copyright INT NOT NULL, Genre VARCHAR(50) NOT NULL, PRIMARY KEY (ID));

named_db=> INSERT INTO books_table VALUES (1, "The Shining", "Stephen King", 1977,
 "Supernatural fiction");
```

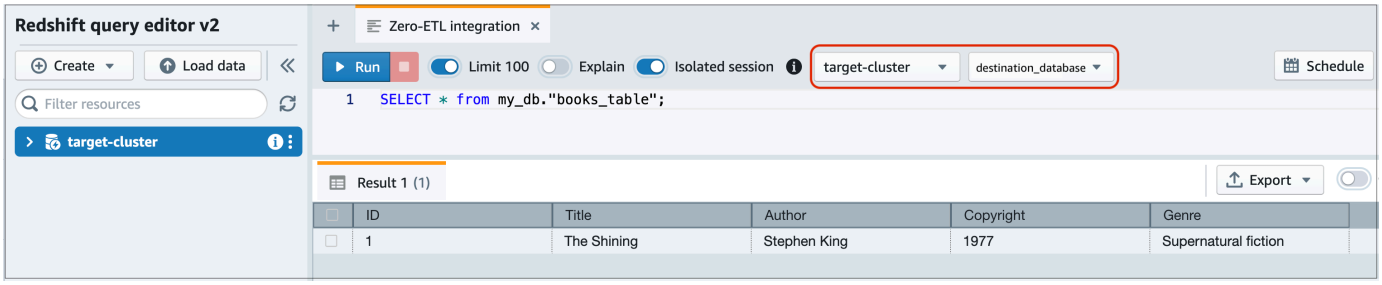
Kueri data Anda di Amazon Redshift

Setelah Anda menambahkan data ke cluster Aurora DB, itu direplikasi ke Amazon Redshift dan siap untuk ditanyakan.

Untuk mengueri data yang direplikasi

1. Buka konsol Amazon Redshift dan pilih Editor kueri v2 dari panel navigasi kiri.
2. Hubungkan ke klaster atau grup kerja Anda dan pilih basis data tujuan Anda (yang Anda buat dari integrasi) dari menu dropdown (destination_database dalam contoh ini). Untuk petunjuk cara membuat basis data tujuan, lihat Membuat basis data tujuan di Amazon Redshift.
3. Jalankan perintah berikut untuk memilih semua data dari tabel yang Anda buat di sumber Aurora DB cluster:

   ```
   SELECT * from my_db."books_table";
   ```

   - *my_db* adalah nama skema basis data Aurora. Opsi ini hanya diperlukan untuk basis data MySQL.
   - *books_table* adalah nama tabel Aurora.

Anda juga dapat mengueri data menggunakan klien baris perintah:

```
destination_database=# select * from my_db."books_table";

 ID |       Title |        Author |   Copyright |                 Genre |  txn_seq |
 txn_id
```

```
----+-------------+---------------+-------------+-----------------------+----------
+--------+
  1 | The Shining |  Stephen King |        1977 |   Supernatural fiction |        2 |
 12192
```

> **Note**
>
> Untuk kepekaan huruf besar/kecil, gunakan tanda kutip ganda (" ") untuk nama skema, tabel, dan kolom. Untuk informasi selengkapnya, lihat enable_case_sensitive_identifier.

Perbedaan jenis data antara basis data Aurora dan Amazon Redshift

Tabel berikut menunjukkan pemetaan jenis data Aurora MySQL atau Aurora PostgreSQL ke jenis data Amazon Redshift yang sesuai. Amazon Aurora saat ini hanya mendukung jenis data ini untuk integrasi nol-ETL.

Jika tabel di cluster DB sumber Anda menyertakan tipe data yang tidak didukung, tabel akan tidak sinkron dan tidak dapat dikonsumsi oleh target Amazon Redshift. Streaming dari sumber ke target berlanjut, tetapi tabel dengan jenis data yang tidak didukung tidak tersedia. Untuk memperbaiki tabel dan membuatnya tersedia di Amazon Redshift, Anda harus mengembalikan perubahan yang melanggar secara manual, lalu menyegarkan integrasi dengan menjalankan `ALTER DATABASE...INTEGRATION REFRESH`.

Topik

- Aurora MySQL
- Aurora PostgreSQL

Aurora MySQL

Jenis data Aurora MySQL	Jenis data Amazon Redshift	Deskripsi	Batasan
INT	INTEGER	Bilangan bulat empat byte bertanda	

Jenis data Aurora MySQL	Jenis data Amazon Redshift	Deskripsi	Batasan
SMALLINT	SMALLINT	Bilangan bulat dua byte bertanda	
TINYINT	SMALLINT	Bilangan bulat dua byte bertanda	
MEDIUMINT	INTEGER	Bilangan bulat empat byte bertanda	
BIGINT	BIGINT	Bilangan bulat delapan byte bertanda	
INT UNSIGNED	BIGINT	Bilangan bulat delapan byte bertanda	
TINYINT UNSIGNED	SMALLINT	Bilangan bulat dua byte bertanda	
MEDIUMINT UNSIGNED	INTEGER	Bilangan bulat empat byte bertanda	
BIGINT UNSIGNED	DECIMAL(20,0)	Numerik persis dari presisi yang dapat dipilih	

Jenis data Aurora MySQL	Jenis data Amazon Redshift	Deskripsi	Batasan
DECIMAL(p,s) = NUMERIC(p,s)	DECIMAL(p,s)	Numerik persis dari presisi yang dapat dipilih	Presisi lebih besar dari 38 dan penskalaan lebih besar dari 37 tidak didukung
DECIMAL(p,s) UNSIGNED = NUMERIC(p,s) UNSIGNED	DECIMAL(p,s)	Numerik persis dari presisi yang dapat dipilih	Presisi lebih besar dari 38 dan penskalaan lebih besar dari 37 tidak didukung
FLOAT4/REAL	REAL	Angka floating-point presisi tunggal	
FLOAT4/REAL UNSIGNED	REAL	Angka floating-point presisi tunggal	
DOUBLE/REAL/FLOAT8	DOUBLE PRECISION	Angka floating-point presisi ganda	
DOUBLE/REAL/FLOAT8 UNSIGNED	DOUBLE PRECISION	Angka floating-point presisi ganda	
BIT(n)	VARBYTE(8)	Nilai biner dengan panjang variabel	

Jenis data Aurora MySQL	Jenis data Amazon Redshift	Deskripsi	Batasan
BINER (n)	VARBYTE(n)	Nilai biner dengan panjang variabel	
VARBINER (n)	VARBYTE(n)	Nilai biner dengan panjang variabel	
CHAR(n)	VARCHAR(n)	Nilai string dengan panjang variabel	
VARCHAR(n)	VARCHAR(n)	Nilai string dengan panjang variabel	
TEXT	VARCHAR(65535)	Nilai string dengan panjang variabel hingga 65535 byte	
TINYTEXT	VARCHAR(255)	Nilai string dengan panjang variabel hingga 255 byte	
MEDIUMTEXT	VARCHAR(65535)	Nilai string dengan panjang variabel hingga 65535 byte	
LONGTEXT	VARCHAR(65535)	Nilai string dengan panjang variabel hingga 65535 byte	

Jenis data Aurora MySQL	Jenis data Amazon Redshift	Deskripsi	Batasan
ENUM	VARCHAR(1020)	Nilai string dengan panjang variabel hingga 1020 byte	
SET	VARCHAR(1020)	Nilai string dengan panjang variabel hingga 1020 byte	
DATE	DATE	Tanggal kalender (tahun, bulan, hari)	
DATETIME	TIMESTAMP	Tanggal dan waktu (tanpa zona waktu)	
TIMESTAMP(p)	TIMESTAMP	Tanggal dan waktu (tanpa zona waktu)	
TIME	VARCHAR(18)	Nilai string dengan panjang variabel hingga 18 byte	
YEAR	VARCHAR(4)	Nilai string dengan panjang variabel hingga 4 byte	

Jenis data Aurora MySQL	Jenis data Amazon Redshift	Deskripsi	Batasan
JSON	SUPER	Data atau dokumen semi-terstruktur sebagai nilai	

Aurora PostgreSQL

Integrasi nol-ETL untuk Aurora PostgreSQL tidak mendukung jenis data kustom atau jenis data yang dibuat oleh ekstensi.

> **Important**
>
> Integrasi nol-ETL dengan fitur Amazon Redshift untuk Aurora PostgreSQL sedang dalam rilis pratinjau. Dokumentasi dan fitur dapat berubah. Anda dapat menggunakan fitur ini hanya dalam lingkungan pengujian, bukan dalam lingkungan produksi. Untuk syarat dan ketentuan pratinjau, lihat Beta dan Pratinjau dalam Persyaratan Layanan AWS.

Jenis data Aurora PostgreSQL	Jenis data Amazon Redshift	Deskripsi	Batasan
bigint	BIGINT	Bilangan bulat delapan byte bertanda	
bigserial	BIGINT	Bilangan bulat delapan byte bertanda	
bit(n)	VARBYTE(n)	Nilai biner dengan panjang variabel	

Jenis data Aurora PostgreSQL	Jenis data Amazon Redshift	Deskripsi	Batasan
bit varying(n)	VARBYTE(n)	Nilai biner dengan panjang variabel	
bit	VARBYTE(1024000)	Nilai string dengan panjang variabel hingga 1.024.000 byte	
boolean	BOOLEAN	Boolean logis (true/false)	
bytea	VARBYTE(1024000)	Nilai string dengan panjang variabel hingga 1.024.000 byte	
character(n)	CHAR(n)	String karakter dengan panjang tetap	
character varying(n)	VARCHAR(65535)	Nilai string dengan panjang variabel	
date	DATE	Tanggal kalender (tahun, bulan, hari)	• Nilai yang lebih besar dari `9999-12-31` tidak didukung • Nilai B.C. tidak didukung

Jenis data Aurora PostgreSQL	Jenis data Amazon Redshift	Deskripsi	Batasan
double precision	DOUBLE PRECISION	Angka floating-point presisi ganda	Nilai subnormal tidak didukung
integer	INTEGER	Bilangan bulat empat byte bertanda	
money	DECIMAL(20,3)	Jumlah mata uang	
numeric(p,s)	DECIMAL(p,s)	Nilai string dengan panjang variabel	• Nilai `NaN` tidak didukung • Presisi lebih besar dari 38 dan penskalaan lebih besar dari 37 tidak didukung • Skala negatif tidak didukung
real	REAL	Angka floating-point presisi tunggal	
smallint	SMALLINT	Bilangan bulat dua byte bertanda	
smallserial	SMALLINT	Bilangan bulat dua byte bertanda	

Jenis data Aurora PostgreSQL	Jenis data Amazon Redshift	Deskripsi	Batasan
serial	INTEGER	Bilangan bulat empat byte bertanda	
text	VARCHAR(65535)	Nilai string dengan panjang variabel hingga 65.535 byte	
time [(p)] [without time zone]	VARCHAR(19)	Nilai string dengan panjang variabel hingga 19 byte	Nilai `Infinity` dan `-Infinity` tidak didukung
time [(p)] with time zone	VARCHAR(22)	Nilai string dengan panjang variabel hingga 22 byte	• Nilai `Infinity` dan `-Infinity` tidak didukung
timestamp [(p)] [without timezone]	TIMESTAMP	Tanggal dan waktu (tanpa zona waktu)	• Nilai `Infinity` dan `-Infinity` tidak didukung • Nilai yang lebih besar dari `9999-12-31` tidak didukung • Nilai B.C. tidak didukung

Jenis data Aurora PostgreSQL	Jenis data Amazon Redshift	Deskripsi	Batasan
timestamp [(p)] with time zone	TIMESTAMPTZ	Tanggal dan waktu (dengan zona waktu)	• Nilai `Infinity` dan `-Infinity` tidak didukung • Nilai yang lebih besar dari `9999-12-31` tidak didukung • Nilai B.C. tidak didukung

Melihat dan memantau integrasi nol-ETL Aurora dengan Amazon Redshift

Anda dapat melihat detail integrasi nol-ETL Amazon Aurora untuk melihat informasi konfigurasi dan statusnya saat ini. Anda juga dapat memantau status integrasi dengan mengueri tampilan sistem tertentu di Amazon Redshift. Selain itu, Amazon Redshift menerbitkan metrik terkait integrasi tertentu ke Amazon, yang dapat Anda lihat dalam konsol CloudWatch Amazon Redshift.

Topik

- Melihat integrasi
- Memantau integrasi menggunakan tabel sistem
- Integrasi pemantauan dengan Amazon EventBridge

Melihat integrasi

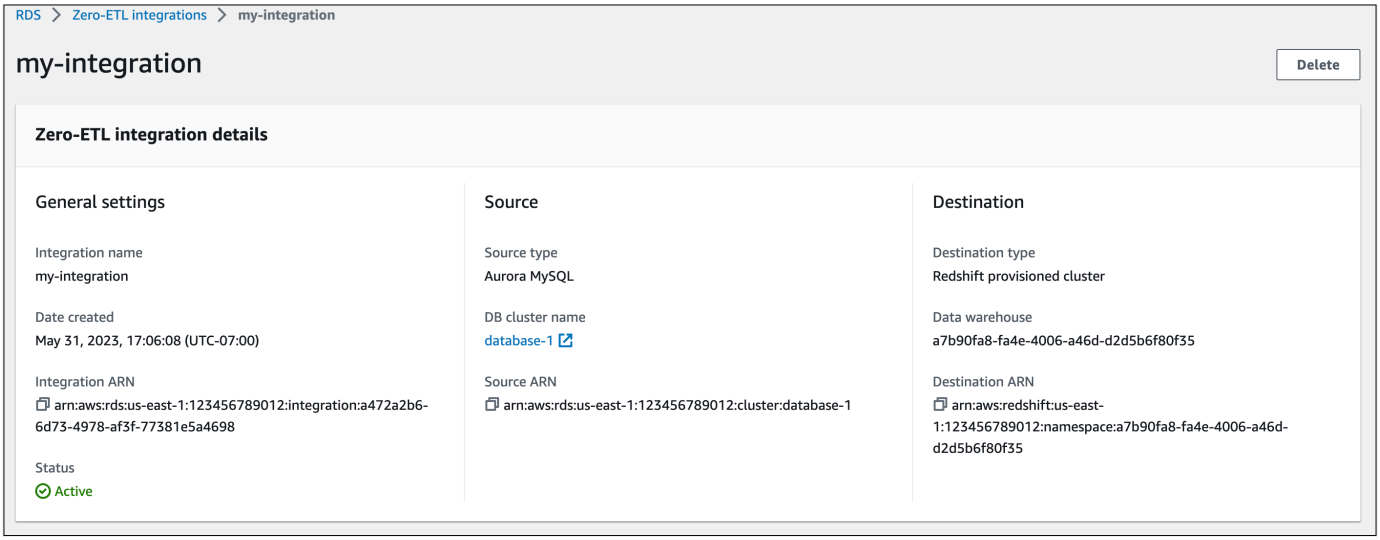
Anda dapat melihat integrasi Aurora Zero-ETL dengan Amazon Redshift menggunakan,, atau RDS AWS Management Console API. AWS CLI

Konsol

Untuk melihat detail integrasi nol-ETL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

   Jika integrasi memiliki klaster DB sumber PostgreSQL Aurora, Anda harus masuk ke Lingkungan Pratinjau Basis Data Amazon RDS di https://us-east-2.console.aws.amazon.com/rds-preview/home?region=us-east-2#databases.
2. Di panel navigasi sebelah kiri, pilih Integrasi nol-ETL.
3. Pilih integrasi untuk melihat detail lebih lanjut tentangnya, seperti cluster DB data sumber dan gudang data target.



Integrasi dapat memiliki status berikut:

- `Creating` – Integrasi sedang dibuat.
- `Active` – Integrasi sedang mengirimkan data transaksional ke gudang data target.
- `Syncing` – Integrasi telah mengalami kesalahan yang dapat dipulihkan dan sedang melakukan reseeding data. Tabel yang terpengaruh tidak tersedia untuk kueri di Amazon Redshift hingga selesai disinkronkan ulang.
- `Needs attention` – Integrasi mengalami peristiwa atau kesalahan yang memerlukan intervensi manual untuk menyelesaikannya. Untuk memperbaiki masalah, ikuti petunjuk dalam pesan kesalahan di halaman detail integrasi.

- `Failed` – Integrasi mengalami peristiwa atau kesalahan yang tidak dapat dipulihkan dan tidak dapat diperbaiki. Anda harus menghapus dan membuat ulang integrasi.
- `Deleting` – Integrasi sedang dihapus.

AWS CLI

Untuk melihat semua integrasi nol-ETL di akun saat ini menggunakan AWS CLI, gunakan perintah deskripsi-integrasi dan tentukan opsi. `--integration-identifier`

Example

Untuk Linux, macOS, atau Unix:

```
aws rds describe-integrations \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

Untuk Windows:

```
aws rds describe-integrations ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

API RDS

Untuk melihat integrasi nol-ETL menggunakan API Amazon RDS, gunakan operasi `DescribeIntegrations` dengan parameter `IntegrationIdentifier`.

Memantau integrasi menggunakan tabel sistem

Amazon Redshift memiliki tabel dan tampilan sistem yang berisi informasi tentang bagaimana sistem berfungsi. Anda dapat mengueri tabel dan tampilan sistem ini dengan cara yang sama seperti Anda akan mengueri tabel basis data lainnya. Untuk informasi selengkapnya tentang tabel dan tampilan sistem di Amazon Redshift, lihat Referensi tabel sistem dalam Panduan Developer Basis Data Amazon Redshift.

Anda dapat mengueri tampilan dan tabel sistem berikut untuk mendapatkan informasi tentang integrasi nol-ETL Aurora Anda dengan Amazon Redshift:

- SVV_INTEGRATION – Menyediakan detail konfigurasi untuk integrasi Anda.
- SVV_INTEGRATION_TABLE_STATE – Menjelaskan status setiap tabel dalam integrasi.

- SYS_INTEGRATION_TABLE_STATE_CHANGE – Menampilkan log perubahan status tabel untuk integrasi.
- SYS_INTEGRATION_ACTIVITY – Menyediakan informasi tentang proses integrasi yang selesai.

Semua CloudWatch metrik Amazon terkait integrasi berasal dari Amazon Redshift. Untuk informasi selengkapnya, lihat Memantau integrasi nol-ETL dalam Panduan Manajemen Amazon Redshift. Saat ini, Amazon Aurora tidak mempublikasikan metrik integrasi apa pun. CloudWatch

Integrasi pemantauan dengan Amazon EventBridge

Amazon Redshift mengirimkan peristiwa terkait integrasi ke Amazon. EventBridge Untuk daftar peristiwa dan ID peristiwa terkait, lihat notifikasi peristiwa integrasi nol-ETL dengan Amazon di Panduan Manajemen Pergeseran Merah EventBridge Amazon.

Memodifikasi integrasi Zero-ETL dengan Amazon Redshift

Anda hanya dapat memodifikasi nama, deskripsi, dan opsi pemfilteran data untuk integrasi nol-ETL dengan Amazon Redshift. Anda tidak dapat memodifikasi AWS KMS kunci yang digunakan untuk mengenkripsi integrasi, atau basis data sumber atau target.

Anda dapat memodifikasi integrasi nol-ETL menggunakan AWS Management Console, API, AWS CLI atau Amazon RDS.

> **Note**
>
> Saat ini, Anda hanya dapat memodifikasi integrasi yang memiliki cluster DB sumber Aurora MySQL. Memodifikasi integrasi tidak didukung untuk rilis pratinjau integrasi Aurora PostgreSQL Zero-ETL dengan Amazon Redshift.

Konsol

Untuk memodifikasi integrasi nol-ETL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih integrasi nol-ETL, lalu pilih integrasi yang ingin Anda ubah.

3. Pilih Ubah dan buat modifikasi pada pengaturan apa pun yang tersedia.
4. Ketika semua perubahan seperti yang Anda inginkan, pilih Ubah.

AWS CLI

Untuk memodifikasi integrasi nol-ETL menggunakan AWS CLI, panggil perintah modify-integration. Seiring dengan`--integration-identifier`, tentukan salah satu opsi berikut:

- `--integration-name`— Tentukan nama baru untuk integrasi.
- `--description`— Tentukan deskripsi baru untuk integrasi.
- `--data-filter`— Tentukan opsi pemfilteran data untuk integrasi. Untuk informasi selengkapnya, lihat the section called “Pemfilteran data untuk integrasi nol-ETL”.

Example

Permintaan berikut memodifikasi integrasi yang ada.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 \
    --integration-name my-renamed-integration
```

Untuk Windows:

```
aws rds modify-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374 ^
    --integration-name my-renamed-integration
```

API RDS

Untuk memodifikasi integrasi nol-ETL menggunakan RDS API, panggil operasi. ModifyIntegration Tentukan pengidentifikasi integrasi, dan parameter yang ingin Anda modifikasi.

Menghapus integrasi nol-ETL Aurora dengan Amazon Redshift

Data transaksional Anda tidak dihapus dari Amazon Aurora atau Amazon Redshift, tetapi Aurora tidak mengirim data baru ke Amazon Redshift.

Anda hanya dapat menghapus integrasi ketika integrasi ini memiliki status `Active`, `Failed`, `Syncing`, atau `Needs attention`.

Anda dapat menghapus integrasi nol-ETL menggunakan AWS Management Console,, atau RDS API AWS CLI.

Konsol

Untuk menghapus integrasi nol-ETL

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

   Jika integrasi memiliki klaster DB sumber PostgreSQL Aurora, Anda harus masuk ke Lingkungan Pratinjau Basis Data Amazon RDS di https://us-east-2.console.aws.amazon.com/rds-preview/home?region=us-east-2#databases.
2. Di panel navigasi sebelah kiri, pilih Integrasi nol-ETL.
3. Pilih integrasi nol-ETL yang ingin Anda hapus.
4. Pilih Tindakan, Hapus, dan konfirmasi penghapusan.

AWS CLI

> **Note**
>
> Selama pratinjau integrasi nol-ETL Aurora PostgreSQL, Anda hanya dapat menghapus integrasi melalui AWS Management Console. Anda tidak dapat menggunakan AWS CLI, Amazon RDS API, atau SDK mana pun.

Untuk menghapus integrasi nol-ETL, gunakan perintah delete-integration dan tentukan opsi `--integration-identifier`.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds delete-integration \
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

Untuk Windows:

```
aws rds delete-integration ^
    --integration-identifier ee605691-6c47-48e8-8622-83f99b1af374
```

API RDS

> **Note**
>
> Selama pratinjau integrasi nol-ETL Aurora PostgreSQL, Anda hanya dapat menghapus integrasi melalui AWS Management Console. Anda tidak dapat menggunakan AWS CLI, Amazon RDS API, atau SDK mana pun.

Untuk menghapus integrasi nol-ETL menggunakan API Amazon RDS, gunakan operasi `DeleteIntegration` dengan parameter `IntegrationIdentifier`.

Memecahkan masalah integrasi nol-ETL Aurora dengan Amazon Redshift

Anda dapat memeriksa status integrasi nol-ETL dengan mengueri tabel sistem SVV_INTEGRATION di Amazon Redshift. Jika kolom `state` memiliki nilai `ErrorState`, artinya ada sesuatu yang salah. Untuk informasi selengkapnya, lihat the section called “Pemantauan menggunakan tabel sistem”.

Gunakan informasi berikut untuk memecahkan masalah umum integrasi nol-ETL Aurora dengan Amazon Redshift.

Topik

- Saya tidak dapat membuat integrasi nol-ETL
- Integrasi saya dalam status Syncing permanen
- Satu atau beberapa tabel Amazon Redshift saya memerlukan sinkronisasi ulang

Saya tidak dapat membuat integrasi nol-ETL

Jika Anda tidak dapat membuat integrasi nol-ETL, pastikan hal berikut sudah benar untuk klaster DB sumber Anda:

- Cluster DB sumber Anda menjalankan , atau Aurora PostgreSQL (kompatibel dengan PostgreSQL 15.4 dan Dukungan Zero-ETL). Untuk memvalidasi versi mesin, pilih tab Konfigurasi untuk klaster DB dan periksa Versi mesin.
- Anda mengonfigurasi parameter klaster DB dengan benar. Jika parameter yang diperlukan tidak diatur dengan benar atau tidak terkait dengan klaster, pembuatan akan gagal. Lihat the section called “Langkah 1: Buat grup parameter klaster DB kustom”.

Selain itu, pastikan hal-hal berikut ini sudah benar untuk gudang data target Anda:

- Kepekaan huruf besar/kecil diaktifkan. Lihat Mengaktifkan kepekaan huruf besar/kecil untuk gudang data Anda.
- Anda menambahkan pengguna utama resmi dan sumber integrasi yang benar. Untuk petunjuknya, lihat Mengonfigurasi otorisasi untuk gudang data Amazon Redshift Anda.

Integrasi saya dalam status **`Syncing`** permanen

Integrasi Anda mungkin secara konsisten menunjukkan status `Syncing` jika Anda mengubah nilai salah satu parameter klaster DB yang diperlukan.

Untuk memperbaiki masalah ini, periksa nilai parameter dalam grup parameter yang terkait dengan klaster DB sumber, dan pastikan nilai tersebut cocok dengan nilai yang diperlukan. Untuk informasi selengkapnya, lihat the section called “Langkah 1: Buat grup parameter klaster DB kustom”.

Jika Anda memodifikasi parameter, pastikan untuk melakukan boot ulang klaster DB untuk menerapkan perubahan.

Satu atau beberapa tabel Amazon Redshift saya memerlukan sinkronisasi ulang

Untuk menjalankan perintah tertentu di klaster DB sumber, Anda mungkin perlu menyinkronkan ulang tabel Anda. Dalam kasus ini, tampilan sistem SVV_INTEGRATION_TABLE_STATE menunjukkan `table_state` dari `ResyncRequired`. Artinya, integrasi harus memuat ulang data sepenuhnya untuk tabel tersebut dari MySQL ke Amazon Redshift.

Tabel akan memasuki status `Syncing` saat mulai disinkronkan ulang. Anda tidak perlu mengambil tindakan manual apa pun untuk menyinkronkan ulang tabel. Saat data tabel disinkronkan ulang, Anda mungkin tidak dapat mengaksesnya di Amazon Redshift.

Berikut ini adalah beberapa contoh operasi yang dapat menempatkan tabel ke dalam status `ResyncRequired`, dan beberapa kemungkinan alternatif untuk dipertimbangkan.

Operasi	Contoh	Alternatif
Menambahkan kolom ke posisi tertentu	`ALTER TABLE table_name ADD COLUMN column_name INTEGER NOT NULL first;`	Amazon Redshift tidak mendukung penambahan kolom ke posisi tertentu menggunakan kata kunci `first` atau `after`. Jika urutan kolom dalam tabel target tidak penting, tambahkan kolom ke akhir tabel menggunakan perintah yang lebih sederhana: `ALTER TABLE table_name ADD COLUMN column_name column_type ;`
Menambahkan kolom stempel	`ALTER TABLE table_name ADD COLUMN column_name TIMESTAMP`	Nilai CURRENT_TIMESTAMP

Operasi	Contoh	Alternatif
waktu dengan `CURRENT_TIMESTAMP` default	`NOT NULL DEFAULT CURRENT_TIMESTAMP;`	untuk baris tabel yang ada dihitung oleh Aurora MySQL dan tidak dapat disimulasikan di Amazon Redshift tanpa sinkronisasi ulang data tabel penuh. Jika memungkinkan, alihkan nilai default ke konstanta literal seperti `2023-01-01 00:00:15` untuk menghindari latensi dalam ketersediaan tabel.

Operasi	Contoh	Alternatif
Melakukan beberapa operasi kolom dalam satu perintah	`ALTER TABLE table_name ADD COLUMN column_1, RENAME COLUMN column_2 TO column_3;`	Pertimbangkan untuk membagi perintah menjadi dua operasi terpisah, `ADD` dan `RENAME`, yang tidak memerlukan sinkronisasi ulang.

Menggunakan Aurora Serverless v2

Aurora Serverless v2 adalah konfigurasi penskalaan otomatis sesuai permintaan untuk Amazon Aurora. Aurora Serverless v2 membantu mengotomatiskan proses pemantauan beban kerja dan menyesuaikan kapasitas basis data Anda. Kapasitas disesuaikan secara otomatis berdasarkan permintaan aplikasi. Anda hanya dikenai biaya untuk sumber daya yang menggunakan klaster DB Anda. Dengan demikian, Aurora Serverless v2 dapat membantu Anda untuk tetap menepati anggaran dan menghindari biaya dari sumber daya komputasi yang tidak Anda gunakan.

Jenis otomatisasi ini sangat berharga untuk basis data multipenghuni, basis data terdistribusi, sistem pengembangan dan pengujian, serta lingkungan lain dengan beban kerja yang sangat bervariasi dan tidak dapat diprediksi.

Topik

- Kasus penggunaan Aurora Serverless v2
- Keuntungan Aurora Serverless v2
- Cara kerja Aurora Serverless v2
- Persyaratan dan batasan untuk Aurora Serverless v2
- Membuat cluster DB yang menggunakan Aurora Serverless v2
- Mengelola cluster Aurora Serverless v2 DB
- Performa dan penskalaan untuk Aurora Serverless v2
- Migrasi ke Aurora Serverless v2

Kasus penggunaan Aurora Serverless v2

Aurora Serverless v2 mendukung banyak jenis beban kerja basis data. Hal ini berkisar dari lingkungan pengembangan dan pengujian, hingga situs web dan aplikasi yang memiliki beban kerja yang tidak terduga, hingga aplikasi bisnis yang paling menuntut dan kritis yang membutuhkan skala dan ketersediaan tinggi.

Aurora Serverless v2 sangat berguna untuk kasus penggunaan berikut:

- Beban kerja variabel – Anda menjalankan beban kerja yang mengalami peningkatan aktivitas secara tiba-tiba dan tidak dapat diprediksi. Contohnya adalah situs info lalu lintas yang mengalami lonjakan aktivitas saat hujan mulai turun. Contoh yang lain adalah situs e-commerce yang

mengalami peningkatan lalu lintas ketika Anda menawarkan obral atau promosi khusus. Dengan Aurora Serverless v2, kapasitas basis data Anda diskalakan secara otomatis untuk memenuhi kebutuhan beban puncak aplikasi dan menurunkan skala kembali saat lonjakan aktivitas berakhir. Dengan Aurora Serverless v2, Anda tidak perlu lagi menyediakan kapasitas puncak atau rata-rata. Anda dapat menentukan batas kapasitas atas untuk menangani situasi terburuk, dan kapasitas tersebut tidak akan digunakan kecuali jika diperlukan.

Granularitas penskalaan di Aurora Serverless v2 membantu Anda mencocokkan kapasitas dengan kebutuhan basis data Anda. Untuk klaster terprovisi, penskalaan memerlukan penambahan instans DB yang sama sekali baru. Untuk klaster Aurora Serverless v1, kenaikan skala membutuhkan penggandaan jumlah unit kapasitas Aurora (ACU) untuk klaster, seperti dari 16 menjadi 32 atau 32 menjadi 64. Sebaliknya, Aurora Serverless v2 dapat menambahkan separuh ACU ketika hanya sedikit peningkatan kapasitas yang dibutuhkan. Layanan ini dapat menambahkan 0,5, 1, 1,5, 2, atau separuh ACU tambahan berdasarkan kapasitas tambahan yang diperlukan untuk menangani peningkatan beban kerja. Layanan ini juga dapat menghapus 0,5, 1, 1,5, 2, atau separuh ACU tambahan ketika beban kerja menurun dan kapasitas tersebut tidak lagi diperlukan.

- Aplikasi multi-penghuni – Dengan Aurora Serverless v2, Anda tidak perlu mengelola kapasitas basis data satu per satu untuk setiap aplikasi dalam armada Anda. Aurora Serverless v2 akan mengelola kapasitas basis data individu untuk Anda.

  Anda dapat membuat klaster untuk setiap penghuni. Dengan demikian, Anda dapat menggunakan fitur seperti kloning, pemulihan snapshot, dan basis data global Aurora untuk meningkatkan ketersediaan tinggi dan pemulihan bencana yang sesuai untuk setiap penghuni.

  Setiap penghuni mungkin memiliki periode sibuk dan idle tertentu tergantung waktu dalam sehari, periode dalam setahun, acara promosi, dan sebagainya. Setiap klaster dapat memiliki rentang kapasitas yang luas. Dengan demikian, klaster dengan aktivitas rendah dikenai biaya instans DB minimal. Setiap klaster dapat dengan cepat menaikkan skala untuk menangani periode aktivitas tinggi.
- Aplikasi baru – Anda men-deploy aplikasi baru dan Anda tidak yakin dengan ukuran instans DB yang Anda butuhkan. Dengan Aurora Serverless v2, Anda dapat menyiapkan klaster dengan satu atau banyak instans DB dan mengatur agar basis data melakukan penskalaan otomatis berdasarkan persyaratan kapasitas aplikasi Anda.
- Aplikasi penggunaan campuran – Misalkan Anda memiliki aplikasi pemrosesan transaksi online (OLTP), tetapi Anda secara berkala mengalami lonjakan lalu lintas kueri. Dengan menentukan tingkat promosi untuk instans DB Aurora Serverless v2 dalam klaster, Anda dapat mengonfigurasi klaster Anda sehingga instans DB pembaca dapat diskalakan secara independen dari instans

DB penulis untuk menangani beban tambahan. Saat lonjakan penggunaan mereda, instans DB pembaca menurunkan skala kembali agar sesuai dengan kapasitas instans DB penulis.

- Perencanaan kapasitas – Misalkan Anda biasanya menyesuaikan kapasitas basis data Anda, atau memverifikasi kapasitas basis data optimal untuk beban kerja Anda, dengan memodifikasi kelas instans DB untuk semua instans DB dalam sebuah klaster. Dengan Aurora Serverless v2, Anda dapat menghindari overhead administratif ini. Anda dapat menentukan kapasitas minimum dan maksimum yang sesuai dengan menjalankan beban kerja dan memeriksa seberapa besar skala instans DB sebenarnya.

  Anda dapat memodifikasi instans DB yang ada dari terprovisi menjadi Aurora Serverless v2 atau dari Aurora Serverless v2 menjadi terprovisi. Anda tidak perlu membuat klaster baru atau instans DB baru dalam kasus seperti itu.

  Dengan basis data global Aurora, Anda mungkin tidak memerlukan kapasitas untuk klaster sekunder sebanyak untuk klaster primer. Anda dapat menggunakan instans DB Aurora Serverless v2 di klaster sekunder. Dengan demikian, kapasitas klaster dapat dinaikkan skalanya jika wilayah sekunder dipromosikan dan mengambil alih beban kerja aplikasi Anda.
- Pengembangan dan pengujian – Selain menjalankan aplikasi yang paling menuntut, Anda juga dapat menggunakan Aurora Serverless v2 untuk lingkungan pengembangan dan pengujian. Dengan Aurora Serverless v2, Anda dapat membuat instans DB dengan kapasitas minimum yang rendah alih-alih menggunakan kelas instans DB db.t* yang dapat melonjak. Anda dapat mengatur kapasitas maksimum cukup tinggi sehingga instans DB tersebut masih dapat menjalankan beban kerja yang substansial tanpa kehabisan memori. Ketika basis data tidak digunakan, semua instans DB diturunkan skalanya untuk menghindari biaya yang tidak perlu.

> **Tip**
>
> Untuk membuatnya nyaman digunakan Aurora Serverless v2 dalam lingkungan pengembangan dan pengujian, AWS Management Console menyediakan pintasan Easy create saat Anda membuat cluster baru. Jika Anda memilih opsi Dev/Tes, Aurora membuat klaster dengan instans DB Aurora Serverless v2 dan rentang kapasitas yang biasa untuk sistem pengembangan dan pengujian.

Menggunakan Aurora Serverless v2 untuk beban kerja terprovisi yang ada

Misalkan Anda sudah memiliki aplikasi Aurora yang berjalan pada klaster terprovisi. Anda dapat memeriksa bagaimana aplikasi akan beroperasi bersama Aurora Serverless v2 dengan menambahkan satu atau beberapa instans DB Aurora Serverless v2 ke klaster yang ada sebagai instans DB pembaca. Anda dapat memeriksa seberapa sering instans DB pembaca dinaikkan dan diturunkan skalanya. Anda dapat menggunakan mekanisme failover Aurora untuk mempromosikan instans DB Aurora Serverless v2 menjadi penulis dan memeriksa fungsinya dalam menangani beban kerja baca/tulis. Dengan demikian, Anda dapat beralih dengan waktu henti minimal dan tanpa mengubah titik akhir yang digunakan aplikasi klien Anda. Untuk detail tentang prosedur untuk mengonversi klaster yang ada menjadi Aurora Serverless v2, lihat Migrasi ke Aurora Serverless v2.

Keuntungan Aurora Serverless v2

Aurora Serverless v2 ditujukan untuk beban kerja variabel atau berfluktuasi. Dengan beban kerja yang tidak dapat diprediksi seperti itu, Anda mungkin mengalami kesulitan dalam merencanakan kapan harus mengubah kapasitas basis data Anda. Anda mungkin juga mengalami kesulitan dalam membuat perubahan kapasitas dengan cukup cepat menggunakan mekanisme yang sudah dikenal seperti menambahkan instans DB atau mengubah kelas instans DB. Aurora Serverless v2 memberikan keuntungan berikut untuk membantu kasus penggunaan seperti:

- Manajemen kapasitas yang lebih sederhana daripada terprovisi – Aurora Serverless v2 mengurangi upaya untuk merencanakan ukuran instans DB dan mengubah ukuran instans DB saat beban kerja berubah. Layanan ini juga mengurangi upaya untuk mempertahankan kapasitas yang konsisten untuk semua instans DB dalam klaster.
- Penskalaan yang lebih cepat dan lebih mudah selama periode aktivitas tinggi – Aurora Serverless v2 menskalakan komputasi dan kapasitas memori sesuai kebutuhan, tanpa gangguan pada transaksi klien atau beban kerja Anda secara keseluruhan. Kemampuan untuk menggunakan instans DB pembaca di Aurora Serverless v2 akan membantu Anda memanfaatkan penskalaan horizontal selain penskalaan vertikal. Kemampuan untuk menggunakan basis data global Aurora berarti Anda dapat menyebarkan beban kerja baca Aurora Serverless v2 Anda ke beberapa Wilayah AWS. Kemampuan ini lebih praktis daripada mekanisme penskalaan untuk klaster terprovisi. Ini juga lebih cepat dan lebih terperinci daripada kemampuan penskalaan di Aurora Serverless v1.
- Hemat biaya selama periode aktivitas rendah – Aurora Serverless v2 membantu Anda menghindari penyediaan instans DB yang berlebihan. Aurora Serverless v2 menambahkan sumber daya secara

bertahap saat instans DB dinaikkan skalanya. Anda hanya membayar untuk sumber daya basis data yang Anda konsumsi. Penggunaan sumber daya Aurora Serverless v2 diukur per detik. Dengan demikian, ketika instans DB menurunkan skala, penggunaan sumber daya yang berkurang segera didaftarkan.

- Kesetaraan fitur yang lebih besar dengan terprovisi – Anda dapat menggunakan banyak fitur Aurora dengan Aurora Serverless v2 yang tidak tersedia untuk Aurora Serverless v1. Misalnya, Aurora Serverless v2 Anda dapat menggunakan instans DB pembaca, database global, otentikasi database AWS Identity and Access Management (IAM), dan Performance Insights. Anda juga dapat menggunakan lebih banyak parameter konfigurasi daripada dengan Aurora Serverless v1.

  Secara khusus, dengan Aurora Serverless v2, Anda dapat memanfaatkan fitur berikut dari klaster terprovisi:

  - Instans DB pembaca – Aurora Serverless v2 dapat memanfaatkan instans DB pembaca untuk menskalakan secara horizontal. Ketika klaster berisi satu atau beberapa instans DB pembaca, klaster dapat segera melakukan failover jika terjadi masalah dengan instans DB penulis. Ini adalah kemampuan yang tidak tersedia di Aurora Serverless v1.
  - Klaster Multi-AZ – Anda dapat mendistribusikan instans DB Aurora Serverless v2 untuk sebuah klaster ke beberapa Zona Ketersediaan (AZ). Menyiapkan klaster Multi-AZ akan membantu memastikan kelangsungan bisnis bahkan dalam kasus masalah yang jarang terjadi yang memengaruhi seluruh AZ. Ini adalah kemampuan yang tidak tersedia di Aurora Serverless v1.
  - Database global — Anda dapat menggunakan Aurora Serverless v2 dalam kombinasi dengan database global Aurora untuk membuat salinan read-only tambahan dari cluster Anda di tempat Wilayah AWS lain untuk tujuan pemulihan bencana.
  - Proksi RDS – Anda dapat menggunakan Proksi Amazon RDS untuk memungkinkan aplikasi Anda mengumpulkan dan berbagi koneksi basis data untuk meningkatkan kemampuan penskalaan.
- Penskalaan yang lebih cepat, lebih granular, kurang mengganggu daripada Aurora Serverless v1 – Aurora Serverless v2 dapat meningkatkan dan menurunkan skala secara lebih cepat. Penskalaan dapat mengubah kapasitas sedikitnya 0,5 ACU, alih-alih menggandakan atau mengurangi separuh jumlah ACU. Penskalaan biasanya terjadi tanpa jeda dalam pemrosesan sama sekali. Penskalaan tidak memerlukan peristiwa yang harus Anda pantau, seperti halnya di Aurora Serverless v1. Penskalaan dapat terjadi saat pernyataan SQL berjalan dan transaksi terbuka, tanpa perlu menunggu titik diam.

Cara kerja Aurora Serverless v2

Gambaran umum berikut menjelaskan cara kerja Aurora Serverless v2.

Topik

- Gambaran umum Aurora Serverless v2
- Konfigurasi untuk cluster Aurora DB
- Kapasitas Aurora Serverless v2
- Penskalaan Aurora Serverless v2
- Aurora Serverless v2 dan ketersediaan tinggi
- Aurora Serverless v2 dan penyimpanan
- Parameter konfigurasi untuk klaster Aurora

Gambaran umum Aurora Serverless v2

Amazon Aurora Serverless v2 cocok untuk beban kerja yang paling menuntut dan sangat bervariasi. Misalnya, penggunaan basis data Anda mungkin berat untuk waktu yang singkat, diikuti dengan aktivitas ringan dalam waktu lama atau tidak ada aktivitas sama sekali. Beberapa contohnya adalah situs web retail, game, atau olahraga dengan peristiwa promosi berkala, dan basis data yang menghasilkan laporan jika diperlukan. Contoh lainnya adalah lingkungan pengembangan dan pengujian, serta aplikasi baru yang penggunaannya dapat meningkat dengan cepat. Untuk kasus seperti ini dan banyak kasus lainnya, mengonfigurasi kapasitas dengan benar sebelumnya tidak selalu dapat dilakukan dengan model terprovisi. Hal ini juga dapat mengakibatkan biaya yang lebih tinggi jika Anda menetapkan penyediaan yang berlebih dan memiliki kapasitas yang tidak Anda gunakan.

Sebaliknya, klaster terprovisi Aurora cocok untuk beban kerja yang stabil. Dengan klaster terprovisi, Anda memilih kelas instans DB yang memiliki jumlah standar memori, daya CPU, bandwidth I/O, dan sebagainya. Jika beban kerja Anda berubah, Anda secara manual memodifikasi kelas instans penulis dan pembaca Anda. Model terprovisi berfungsi dengan baik saat Anda dapat menyesuaikan kapasitas sebelum pola konsumsi yang diharapkan dan Anda sanggup mengalami pemadaman singkat saat mengubah kelas instans penulis dan pembaca di klaster Anda.

Aurora Serverless v2 dirancang secara menyeluruh untuk mendukung klaster DB nirserver yang dapat diskalakan secara instan. Aurora Serverless v2 direkayasa untuk memberikan tingkat

keamanan dan isolasi yang sama seperti penulis dan pembaca terprovisi. Aspek-aspek ini sangat penting dalam lingkungan cloud nirserver multipenghuni. Mekanisme penskalaan dinamis memiliki overhead yang sangat sedikit sehingga dapat merespons dengan cepat perubahan beban kerja basis data. Mekanisme ini juga cukup kuat untuk memenuhi permintaan memproses permintaan yang meningkat secara drastis.

Dengan menggunakan Aurora Serverless v2, Anda dapat membuat klaster DB Aurora tanpa dibatasi kapasitas basis data tertentu untuk setiap penulis dan pembaca. Anda menentukan rentang kapasitas minimum dan maksimum. Aurora menskalakan setiap penulis atau pembaca Aurora Serverless v2 di klaster dalam rentang kapasitas tersebut. Dengan menggunakan klaster Multi-AZ yang memungkinkan setiap penulis atau pembaca diskalakan secara dinamis, Anda dapat memanfaatkan penskalaan dinamis dan ketersediaan tinggi.

Aurora Serverless v2 menskalakan sumber daya basis data secara otomatis berdasarkan spesifikasi kapasitas minimum dan maksimum Anda. Penskalaan dilakukan dengan cepat karena sebagian besar operasi peristiwa penskalaan mempertahankan penulis atau pembaca di host yang sama. Dalam kasus yang jarang terjadi saat penulis atau pembaca Aurora Serverless v2 dipindahkan dari satu host ke host lainnya, Aurora Serverless v2 akan mengelola koneksinya secara otomatis. Anda tidak perlu mengubah kode aplikasi klien basis data Anda atau string koneksi basis data Anda.

Dengan Aurora Serverless v2, seperti halnya klaster terprovisi, kapasitas penyimpanan dan kapasitas komputasinya terpisah. Ketika kita mengacu pada kapasitas dan penskalaan Aurora Serverless v2, selalu kapasitas komputasinya yang meningkat atau menurun. Dengan demikian, klaster Anda dapat berisi banyak terabyte data bahkan ketika kapasitas CPU dan memori diturunkan skalanya ke tingkat rendah.

Alih-alih menyediakan dan mengelola server basis data, Anda menentukan kapasitas basis data. Untuk detail tentang kapasitas Aurora Serverless v2, lihat Kapasitas Aurora Serverless v2. Kapasitas sebenarnya setiap penulis atau pembaca Aurora Serverless v2 bervariasi dari waktu ke waktu, tergantung pada beban kerja Anda. Untuk detail tentang mekanisme tersebut, lihat Penskalaan Aurora Serverless v2.

> **Important**
>
> Dengan Aurora Serverless v1, klaster Anda memiliki satu ukuran kapasitas komputasi yang dapat diskalakan antara nilai kapasitas minimum dan maksimum. Dengan Aurora Serverless v2, klaster Anda dapat berisi pembaca selain penulis. Setiap penulis dan pembaca Aurora Serverless v2 dapat diskalakan antara nilai kapasitas minimum dan maksimum. Dengan demikian, total kapasitas klaster Aurora Serverless v2 Anda tergantung pada rentang

kapasitas yang Anda tentukan untuk klaster DB Anda serta jumlah penulis dan pembaca di klaster ini. Pada waktu tertentu, Anda hanya dikenai biaya untuk kapasitas Aurora Serverless v2 yang secara aktif digunakan dalam klaster DB Aurora Anda.

Konfigurasi untuk cluster Aurora DB

Untuk setiap klaster DB Aurora Anda, Anda dapat memilih kombinasi kapasitas Aurora Serverless v2, kapasitas terprovisi, atau keduanya.

Anda dapat mengatur klaster yang berisi kapasitas Aurora Serverless v2 dan kapasitas terprovisi, yang disebut klaster konfigurasi campuran. Misalnya, anggaplah Anda membutuhkan lebih banyak kapasitas baca/tulis daripada yang tersedia untuk sebuah penulis Aurora Serverless v2. Dalam hal ini, Anda dapat mengatur klaster dengan penulis terprovisi yang sangat besar. Dalam hal ini, Anda masih dapat menggunakan Aurora Serverless v2 untuk pembaca. Atau anggaplah beban kerja tulis untuk klaster Anda bervariasi, tetapi beban kerja baca stabil. Dalam hal ini, Anda dapat mengatur klaster Anda dengan sebuah penulis Aurora Serverless v2 dan satu atau beberapa pembaca terprovisi.

Anda juga dapat mengatur klaster DB yang semua kapasitasnya dikelola oleh Aurora Serverless v2. Untuk melakukannya, Anda dapat membuat klaster baru dan menggunakan Aurora Serverless v2 dari awal. Atau Anda dapat mengganti semua kapasitas terprovisi di klaster yang ada dengan Aurora Serverless v2. Misalnya, beberapa jalur upgrade dari versi mesin yang lebih lama harus dimulai dengan penulis terprovisi lalu diganti dengan penulis Aurora Serverless v2. Untuk prosedur dalam membuat klaster DB baru dengan Aurora Serverless v2 atau untuk mengalihkan klaster DB yang ada ke Aurora Serverless v2, lihat Membuat klaster DB Aurora Serverless v2 dan Beralih dari klaster terprovisi ke Aurora Serverless v2.

Jika Anda tidak menggunakan Aurora Serverless v2 sama sekali dalam klaster DB, semua penulis dan pembaca di klaster DB akan memiliki jenis terprovisi. Ini adalah jenis klaster DB terlama dan paling umum yang sudah dikenal oleh sebagian besar pengguna. Bahkan, sebelum Aurora Serverless, tidak ada nama khusus untuk klaster DB Aurora semacam ini. Kapasitas terprovisi bersifat konstan. Biayanya relatif mudah untuk diperkirakan. Namun, Anda harus memprediksi sebelumnya berapa banyak kapasitas yang Anda butuhkan. Dalam beberapa kasus, prediksi Anda mungkin tidak akurat atau kebutuhan kapasitas Anda mungkin berubah. Dalam kasus ini, klaster DB Anda dapat mengalami kekurangan penyediaan (lebih lambat dari yang Anda inginkan) atau kelebihan penyediaan (lebih mahal dari yang Anda inginkan).

Kapasitas Aurora Serverless v2

Satuan ukur untuk Aurora Serverless v2 adalah unit kapasitas Aurora (ACU). Kapasitas Aurora Serverless v2 tidak terkait dengan kelas instans DB yang Anda gunakan untuk klaster terprovisi.

Setiap ACU adalah kombinasi dari sekitar 2 gibibyte (GiB) memori, CPU yang sesuai, dan jaringan. Anda menentukan rentang kapasitas basis data menggunakan satuan ukur ini. Metrik `ACUUtilization` dan `ServerlessDatabaseCapacity` membantu Anda menentukan berapa banyak kapasitas yang sebenarnya digunakan basis data Anda dan di mana kapasitas tersebut berada dalam rentang yang ditentukan.

Setiap penulis atau pembaca DB Aurora Serverless v2 memiliki kapasitas setiap saat. Kapasitas direpresentasikan sebagai angka floating–point yang merepresentasikan ACU. Kapasitas ini meningkat atau menurun setiap kali penulis atau pembaca diskalakan. Nilai ini diukur setiap detik. Untuk setiap klaster DB tempat Anda ingin menggunakan Aurora Serverless v2, Anda menentukan rentang kapasitas: nilai kapasitas minimum dan maksimum untuk menskalakan setiap penulis atau pembaca Aurora Serverless v2. Rentang kapasitasnya sama untuk setiap penulis atau pembaca Aurora Serverless v2 dalam klaster DB. Setiap penulis atau pembaca Aurora Serverless v2 memiliki kapasitasnya sendiri, yang berada pada titik tertentu dalam rentang tersebut.

Kapasitas Aurora Serverless v2 terbesar yang dapat Anda tentukan adalah 128 ACU. Untuk semua pertimbangan saat memilih nilai kapasitas maksimum, lihat Memilih pengaturan kapasitas Aurora Serverless v2 maksimum untuk klaster.

Kapasitas Aurora Serverless v2 terkecil yang dapat Anda tentukan adalah 0,5 ACU. Anda dapat menentukan angka yang lebih tinggi jika kapasitasnya kurang dari atau sama dengan nilai kapasitas maksimum. Mengatur kapasitas minimum ke angka kecil akan memungkinkan klaster DB yang berbeban ringan mengonsumsi sumber daya komputasi minimal. Pada saat yang sama, klaster tersebut tetap siap untuk menerima koneksi dengan segera dan menaikkan skalanya ketika menjadi sibuk.

Kami merekomendasikan pengaturan kapasitas minimum ke sebuah nilai yang memungkinkan setiap penulis atau pembaca DB mempertahankan rangkaian aplikasi yang beroperasi di pool buffer. Dengan begitu, konten pool buffer tidak dibuang selama periode idle. Untuk semua pertimbangan saat memilih nilai kapasitas minimum, lihat Memilih pengaturan kapasitas Aurora Serverless v2 minimum untuk klaster.

Bergantung pada cara Anda mengonfigurasi pembaca dalam klaster DB Multi-AZ, kapasitas klaster ini dapat dikaitkan dengan kapasitas penulis atau terpisah. Untuk detail tentang cara melakukannya, lihat Penskalaan Aurora Serverless v2.

Pemantauan Aurora Serverless v2 berkaitan dengan pengukuran nilai kapasitas untuk penulis dan pembaca di klaster DB Anda dari waktu ke waktu. Jika basis data Anda tidak menurunkan skalanya ke kapasitas minimum, Anda dapat mengambil tindakan seperti menyesuaikan kapasitas minimum dan mengoptimalkan aplikasi basis data Anda. Jika basis data Anda secara konsisten mencapai kapasitas maksimumnya, Anda dapat mengambil tindakan seperti meningkatkan kapasitas maksimum. Anda juga dapat mengoptimalkan aplikasi basis data Anda dan menyebarkan beban kueri ke lebih banyak pembaca.

Biaya untuk kapasitas Aurora Serverless v2 diukur dalam hal ACU-jam. Untuk informasi tentang cara penghitungan biaya Aurora Serverless v2, lihat halaman Harga Aurora.

Misalkan jumlah total penulis dan pembaca di klaster Anda adalah N. Dalam hal ini, klaster mengonsumsi kira-kira *n* x *minimum ACUs* ketika Anda tidak menjalankan operasi basis data apa pun. Aurora sendiri mungkin menjalankan operasi pemantauan atau pemeliharaan yang menyebabkan sejumlah kecil beban. Klaster tersebut mengonsumsi tidak lebih dari *n* x *maximum ACUs* ketika basis data berjalan pada kapasitas penuh.

Untuk detail selengkapnya tentang memilih nilai ACU minimum dan maksimum yang sesuai, lihat Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora. Nilai ACU minimum dan maksimum yang Anda tentukan juga memengaruhi cara kerja beberapa parameter konfigurasi Aurora untuk Aurora Serverless v2. Untuk detail tentang interaksi antara rentang kapasitas dan parameter konfigurasi, lihat Menggunakan grup parameter untuk Aurora Serverless v2.

Penskalaan Aurora Serverless v2

Untuk setiap penulis atau pembaca Aurora Serverless v2, Aurora terus melacak pemanfaatan sumber daya seperti CPU, memori, dan jaringan. Pengukuran ini secara kolektif disebut beban. Beban mencakup operasi basis data yang dilakukan oleh aplikasi Anda. Hal ini juga mencakup pemrosesan latar belakang untuk server basis data dan tugas administratif Aurora. Ketika kapasitas dibatasi oleh salah satu dari hal ini, Aurora Serverless v2 akan menaikkan skalanya. Aurora Serverless v2 juga menaikkan skalanya ketika mendeteksi masalah performa yang dapat diselesaikan dengan penaikan skala. Anda dapat memantau pemanfaatan sumber daya dan pengaruhnya terhadap penskalaan Aurora Serverless v2 dengan menggunakan prosedur dalam CloudWatch Metrik Amazon penting untuk Aurora Serverless v2 dan Memantau performa Aurora Serverless v2 dengan Wawasan Performa.

Beban dapat bervariasi di seluruh penulis dan pembaca di klaster DB Anda. Penulis menangani semua laporan bahasa definisi data (DDL), seperti `CREATE TABLE`, `ALTER TABLE`, dan `DROP TABLE`. Penulis juga menangani semua laporan bahasa manipulasi data (DML), seperti `INSERT` dan `UPDATE`. Pembaca dapat memproses pernyataan hanya baca, seperti kueri `SELECT`.

Penskalaan adalah operasi yang meningkatkan atau menurunkan kapasitas Aurora Serverless v2 untuk basis data Anda. Dengan Aurora Serverless v2, setiap penulis dan pembaca memiliki nilai kapasitas terkininya sendiri, yang diukur dalam ACU. Aurora Serverless v2 menaikkan skala penulis atau pembaca hingga kapasitas yang lebih tinggi ketika kapasitas saat ini terlalu rendah untuk menangani beban. Operasi ini menurunkan skala penulis atau pembaca ke kapasitas yang lebih rendah ketika kapasitas saat ini lebih tinggi dari yang dibutuhkan.

Tidak seperti Aurora Serverless v1, yang diskalakan dengan menggandakan kapasitas setiap kali klaster DB mencapai ambang batas, Aurora Serverless v2 dapat meningkatkan kapasitas secara bertahap. Ketika permintaan beban kerja Anda mulai mencapai kapasitas basis data saat ini dari penulis atau pembaca, Aurora Serverless v2 akan meningkatkan jumlah ACU untuk penulis atau pembaca tersebut. Aurora Serverless v2 menskalakan kapasitas dalam inkremen yang diperlukan untuk memberikan performa terbaik untuk sumber daya yang dikonsumsi. Penskalaan terjadi dalam inkremen, paling kecil 0,5 ACU. Semakin besar kapasitas saat ini, semakin besar inkremen penskalaan dan akibatnya semakin cepat penskalaan dapat terjadi.

Karena Aurora Serverless v2 penskalaan sangat sering, granular, dan tidak mengganggu, itu tidak menyebabkan peristiwa diskrit seperti yang AWS Management Console terjadi. Aurora Serverless v1 Sebagai gantinya, Anda dapat mengukur CloudWatch metrik Amazon seperti `ServerlessDatabaseCapacity` dan `ACUUtilization` dan melacak nilai minimum, maksimum, dan rata-ratanya dari waktu ke waktu. Untuk mempelajari selengkapnya tentang metrik Aurora, lihat Memantau metrik di klaster Amazon Aurora. Untuk tips tentang pemantauan Aurora Serverless v2, lihat CloudWatch Metrik Amazon penting untuk Aurora Serverless v2.

Anda dapat memilih untuk membuat pembaca diskalakan pada saat yang sama dengan penulis terkait, atau secara independen dari penulis. Anda dapat melakukannya dengan menentukan tingkat promosi untuk pembaca tersebut.

- Pembaca di tingkat promosi 0 dan 1 akan diskalakan pada saat yang sama dengan penulis. Perilaku penskalaan tersebut membuat pembaca di tingkat prioritas 0 dan 1 cocok untuk mendukung ketersediaan. Hal ini karena pembaca tersebut selalu diatur ukurannya sesuai dengan kapasitas yang tepat untuk mengambil alih beban kerja dari penulis jika terjadi failover.

- Pembaca di tingkat promosi 2–15 akan diskalakan secara independen dari penulis. Setiap pembaca tetap berada dalam nilai ACU minimum dan maksimum yang Anda tentukan untuk klaster Anda. Ketika pembaca diskalakan secara independen dari DB penulis terkait, pembaca tersebut bisa menjadi idle dan menurunkan skalanya sementara penulis terus memproses volume transaksi yang tinggi. Pembaca tersebut masih tersedia sebagai target failover jika tidak ada pembaca lain yang tersedia di tingkat promosi yang lebih rendah. Namun, jika dipromosikan menjadi penulis, pembaca tersebut mungkin perlu dinaikkan skalanya untuk menangani beban kerja penuh dari penulis.

Untuk detail tentang tingkat promosi, lihat Memilih tingkat promosi untuk pembaca Aurora Serverless v2.

Gagasan tentang titik penskalaan dan periode batas waktu terkait dari Aurora Serverless v1 tidak berlaku di Aurora Serverless v2. Penskalaan Aurora Serverless v2 dapat terjadi saat koneksi basis data terbuka, saat transaksi SQL sedang dalam proses, saat tabel dikunci, dan saat tabel sementara sedang digunakan. Aurora Serverless v2 tidak menunggu titik tenang untuk memulai penskalaan. Penskalaan tidak mengganggu operasi basis data apa pun yang sedang berlangsung.

Jika beban kerja Anda membutuhkan lebih banyak kapasitas baca daripada yang tersedia dengan satu penulis dan satu pembaca, Anda dapat menambahkan beberapa pembaca Aurora Serverless v2 ke klaster. Setiap pembaca Aurora Serverless v2 dapat diskalakan dalam rentang nilai kapasitas minimum dan maksimum yang Anda tentukan untuk klaster DB Anda. Anda dapat menggunakan titik akhir pembaca klaster untuk mengarahkan sesi hanya baca ke pembaca dan mengurangi beban pada penulis.

Apakah Aurora Serverless v2 melakukan penskalaan atau tidak, dan seberapa cepat penskalaan terjadi begitu dimulai, juga akan tergantung pada pengaturan ACU minimum dan maksimum untuk klaster. Selain itu, hal tersebut bergantung pada apakah pembaca dikonfigurasi untuk diskalakan bersama dengan penulis atau secara independen. Untuk detail tentang faktor-faktor yang mempengaruhi penskalaan Aurora Serverless v2, lihat Performa dan penskalaan untuk Aurora Serverless v2.

> **Note**
>
> Saat ini, penulis dan pembaca Aurora Serverless v2 tidak menurunkan skalanya hingga nol ACU. Penulis dan pembaca Aurora Serverless v2 yang idle dapat menurunkan skalanya ke nilai ACU minimum yang Anda tentukan untuk klaster.

Perilaku tersebut berbeda dari Aurora Serverless v1, yang dapat dijeda setelah periode idle, tetapi kemudian membutuhkan waktu untuk dilanjutkan ketika Anda membuka koneksi baru. Ketika klaster DB Anda dengan kapasitas Aurora Serverless v2 tidak diperlukan untuk beberapa waktu, Anda dapat menghentikan dan memulai klaster seperti dengan klaster DB terprovisi. Untuk detail tentang menghentikan dan memulai klaster, lihat Menghentikan dan memulai klaster DB Amazon Aurora.

Aurora Serverless v2 dan ketersediaan tinggi

Cara untuk menetapkan ketersediaan tinggi untuk klaster DB Aurora adalah dengan menjadikannya sebagai klaster DB Multi-AZ. Klaster DB Aurora Multi-AZ memiliki kapasitas komputasi yang tersedia setiap saat di lebih dari satu Zona Ketersediaan (AZ). Konfigurasi tersebut membuat basis data Anda tetap aktif dan berjalan bahkan jika terjadi pemadaman yang signifikan. Aurora melakukan failover otomatis jika terjadi masalah yang memengaruhi penulis atau bahkan seluruh AZ. Dengan Aurora Serverless v2, Anda dapat memilih kapasitas komputasi siaga yang akan dinaikkan dan diturunkan skalanya bersama dengan kapasitas penulis. Dengan begitu, kapasitas komputasi di AZ kedua siap mengambil alih beban kerja saat ini kapan saja. Pada saat yang sama, kapasitas komputasi di semua AZ dapat diturunkan skalanya saat basis data idle. Untuk detail tentang cara kerja Aurora Wilayah AWS dan Availability Zones, lihat. Ketersediaan yang tinggi untuk instans DB Aurora

Kemampuan Multi-AZ Aurora Serverless v2 menggunakan pembaca selain penulis. Dukungan untuk pembaca baru tersedia di Aurora Serverless v2 dibandingkan dengan Aurora Serverless v1. Anda dapat menambahkan hingga 15 pembaca Aurora Serverless v2 yang tersebar di 3 AZ ke klaster DB Aurora.

Untuk aplikasi bisnis penting yang harus tetap tersedia bahkan jika terjadi masalah yang memengaruhi seluruh klaster Anda atau seluruh AWS Wilayah, Anda dapat menyiapkan basis data global Aurora. Anda dapat menggunakan kapasitas Aurora Serverless v2 di klaster sekunder agar siap mengambil alih selama pemulihan bencana. Klaster tersebut juga dapat menurunkan skalanya ketika basis data tidak sibuk. Untuk detail tentang basis data global Aurora, lihat Menggunakan basis data global Amazon Aurora.

Aurora Serverless v2 berfungsi seperti klaster terprovisi untuk failover dan fitur ketersediaan tinggi lainnya. Untuk informasi selengkapnya, lihat Ketersediaan yang tinggi untuk Amazon Aurora.

Misalkan Anda ingin memastikan ketersediaan maksimum untuk klaster Aurora Serverless v2 Anda. Anda dapat membuat pembaca selain penulis. Jika Anda menetapkan pembaca ke tingkat promosi

0 atau 1, penskalaan apa pun yang terjadi pada penulis juga terjadi pada pembaca. Dengan begitu, pembaca dengan kapasitas yang sama selalu siap mengambil alih untuk penulis jika terjadi failover.

Misalkan Anda ingin menjalankan laporan triwulanan untuk bisnis Anda seiring klaster Anda terus memproses transaksi. Jika Anda menambahkan pembaca Aurora Serverless v2 ke klaster dan menetapkannya ke tingkat promosi dari 2 hingga 15, Anda dapat terhubung langsung ke pembaca tersebut untuk menjalankan laporan. Bergantung pada berapa banyak memori dan CPU yang diperlukan kueri pelaporan, pembaca dapat menaikkan skala untuk mengakomodasi beban kerja. Pembaca ini kemudian dapat kembali diturunkan skalanya ketika laporannya sudah selesai.

Aurora Serverless v2 dan penyimpanan

Penyimpanan untuk setiap klaster DB Aurora terdiri dari enam salinan dari semua data Anda, yang tersebar di tiga AZ. Replikasi data bawaan ini berlaku terlepas dari apakah klaster DB Anda mencakup pembaca selain penulis. Dengan begitu, data Anda aman, bahkan dari masalah yang memengaruhi kapasitas komputasi klaster.

Penyimpanan Aurora Serverless v2 memiliki karakteristik keandalan dan durabilitas yang sama seperti yang dijelaskan dalam Penyimpanan dan keandalan Amazon Aurora. Hal ini karena penyimpanan untuk klaster DB Aurora beroperasi dengan cara yang sama, baik kapasitas komputasinya menggunakan Aurora Serverless v2 maupun terprovisi.

Parameter konfigurasi untuk klaster Aurora

Anda dapat menyesuaikan semua parameter konfigurasi klaster dan basis data yang sama untuk klaster dengan kapasitas Aurora Serverless v2 seperti untuk klaster DB terprovisi. Namun, beberapa parameter terkait kapasitas ditangani secara berbeda untuk Aurora Serverless v2. Dalam klaster konfigurasi campuran, nilai parameter yang Anda tentukan untuk parameter terkait kapasitas tersebut masih berlaku untuk penulis dan pembaca terprovisi.

Hampir semua parameter beroperasi dengan cara yang sama untuk penulis Aurora Serverless v2 dan pembaca seperti untuk klaster terprovisi. Hal ini tidak termasuk beberapa parameter yang disesuaikan oleh Aurora secara otomatis selama penskalaan, dan beberapa parameter yang disimpan oleh Aurora pada nilai tetap yang bergantung pada pengaturan kapasitas maksimum.

Misalnya, jumlah memori yang dicadangkan untuk cache buffer meningkat saat penulis atau pembaca dinaikkan skalanya, dan berkurang saat penulis atau pembaca diturunkan skalanya. Dengan begitu, memori dapat dilepaskan ketika basis data Anda tidak sibuk. Sebaliknya, Aurora

secara otomatis menetapkan jumlah maksimum koneksi ke nilai yang sesuai berdasarkan pengaturan kapasitas maksimum. Dengan begitu, koneksi aktif tidak terputus jika beban berkurang dan Aurora Serverless v2 menurunkan skalanya. Untuk informasi tentang cara Aurora Serverless v2 menangani parameter tertentu, lihat Menggunakan grup parameter untuk Aurora Serverless v2.

Persyaratan dan batasan untuk Aurora Serverless v2

Saat Anda membuat cluster tempat Anda berniat menggunakan instans Aurora Serverless v2 DB, perhatikan persyaratan dan batasan berikut.

Topik

- Ketersediaan Wilayah dan versi
- Klaster yang menggunakan Aurora Serverless v2 harus memiliki rentang kapasitas yang ditentukan
- Beberapa fitur terprovisi tidak didukung di Aurora Serverless v2
- Beberapa aspek Aurora Serverless v2 berbeda dari Aurora Serverless v1

Ketersediaan Wilayah dan versi

Ketersediaan fitur dan dukungan bervariasi di seluruh versi spesifik dari setiap mesin basis data Aurora, dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang versi dan ketersediaan Wilayah dengan Aurora dan Aurora Serverless v2, lihat Aurora Serverless v2.

Contoh berikut menunjukkan AWS CLI perintah untuk mengonfirmasi nilai mesin DB yang tepat yang dapat Anda gunakan Aurora Serverless v2 untuk spesifik Wilayah AWS. Parameter `--db-instance-class` untuk Aurora Serverless v2 selalu berupa `db.serverless`. Parameter `--engine` dapat berupa `aurora-mysql` atau `aurora-postgresql`. Gantikan nilai `--region` dan `--engine` yang sesuai untuk mengonfirmasi nilai `--engine-version` yang dapat Anda gunakan. Jika perintah tidak menghasilkan output apa pun, Aurora Serverless v2 tidak tersedia untuk kombinasi mesin Wilayah AWS dan DB itu.

```
aws rds describe-orderable-db-instance-options --engine aurora-mysql --db-instance-
class db.serverless \
  --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output
 text

aws rds describe-orderable-db-instance-options --engine aurora-postgresql --db-
instance-class db.serverless \
```

```
 --region my_region --query 'OrderableDBInstanceOptions[].[EngineVersion]' --output
text
```

Klaster yang menggunakan Aurora Serverless v2 harus memiliki rentang kapasitas yang ditentukan

Klaster Aurora harus memiliki atribut `ServerlessV2ScalingConfiguration` sebelum Anda dapat menambahkan instans DB apa pun yang menggunakan kelas instans DB `db.serverless`. Atribut ini menentukan rentang kapasitas. Kapasitas Aurora Serverless v2 berkisar dari minimum 0,5 unit kapasitas Aurora (ACU) hingga 128 ACU, dengan inkremen 0,5 ACU. Setiap ACU menyediakan kapasitas yang setara dengan sekitar 2 gibibyte (GiB) RAM serta CPU dan jaringan terkait. Untuk detail tentang cara Aurora Serverless v2 menggunakan pengaturan rentang kapasitas, lihat Cara kerja Aurora Serverless v2.

Anda dapat menentukan nilai ACU minimum dan maksimum AWS Management Console saat Anda membuat cluster dan instans Aurora Serverless v2 DB terkait. Anda juga dapat menentukan opsi `--serverless-v2-scaling-configuration` di AWS CLI. Atau, Anda dapat menentukan parameter `ServerlessV2ScalingConfiguration` dengan API Amazon RDS. Anda dapat menentukan atribut ini saat Anda membuat klaster atau mengubah klaster yang sudah ada. Untuk prosedur dalam mengatur rentang kapasitas, lihat Mengatur rentang kapasitas Aurora Serverless v2 untuk sebuah klaster. Untuk pembahasan mendetail tentang cara memilih nilai kapasitas minimum dan maksimum dan bagaimana pengaturan tersebut memengaruhi beberapa parameter basis data, lihat Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora.

Beberapa fitur terprovisi tidak didukung di Aurora Serverless v2

Fitur berikut dari instans DB terprovisi Aurora saat ini tidak tersedia untuk Amazon Aurora Serverless v2:

- Stream aktivitas basis data (DAS).
- Manajemen cache klaster untuk Aurora PostgreSQL. Parameter konfigurasi `apg_ccm_enabled` tidak berlaku untuk instans DB Aurora Serverless v2.

Beberapa fitur Aurora berfungsi dengan Aurora Serverless v2, tetapi dapat menyebabkan masalah jika rentang kapasitas Anda lebih rendah dari yang diperlukan untuk persyaratan memori fitur-fitur tersebut dengan beban kerja spesifik Anda. Dalam hal ini, database Anda mungkin tidak berfungsi sebaik biasanya, atau mungkin mengalami out-of-memory kesalahan. Untuk rekomendasi tentang

pengaturan rentang kapasitas yang sesuai, lihat Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora. Untuk informasi pemecahan masalah jika database Anda menemukan out-of-memory kesalahan karena rentang kapasitas yang salah dikonfigurasi, lihat. Menghindari out-of-memory kesalahan

Aurora Auto Scaling tidak didukung. Jenis penskalaan ini menambahkan pembaca baru untuk menangani beban kerja intensif baca tambahan, berdasarkan penggunaan CPU. Namun, penskalaan berdasarkan penggunaan CPU tidak berarti untukAurora Serverless v2. Sebagai alternatif, Anda dapat membuat instans DB pembaca Aurora Serverless v2 terlebih dahulu dan membiarkannya agar mengalami penurunan skala ke kapasitas rendah. Hal tersebut adalah cara yang lebih cepat dan tidak terlalu mengganggu untuk menskalakan kapasitas baca klaster daripada menambahkan instans DB baru secara dinamis.

Beberapa aspek Aurora Serverless v2 berbeda dari Aurora Serverless v1

Jika Anda adalah Aurora Serverless v1 pengguna dan ini adalah pertama kalinya Anda menggunakanAurora Serverless v2, konsultasikan perbedaan antara Aurora Serverless v2 dan Aurora Serverless v1 persyaratan untuk memahami bagaimana persyaratan berbeda antara Aurora Serverless v1 danAurora Serverless v2.

Membuat cluster DB yang menggunakan Aurora Serverless v2

Untuk membuat klaster Aurora tempat Anda dapat menambahkan instans DB Aurora Serverless v2, Anda perlu mengikuti prosedur yang sama seperti dalam Membuat klaster DB Amazon Aurora. Dengan Aurora Serverless v2, klaster Anda dapat dipertukarkan dengan klaster terprovisi. Anda dapat memiliki klaster yang sebagian instans DB-nya menggunakan Aurora Serverless v2 dan sebagian instans DB lainnya terprovisi.

Topik

- Pengaturan untuk cluster Aurora Serverless v2 DB
- Membuat klaster DB Aurora Serverless v2
- Membuat instance DB Aurora Serverless v2 penulis

Pengaturan untuk cluster Aurora Serverless v2 DB

Pastikan bahwa pengaturan awal cluster memenuhi persyaratan yang tercantum dalamPersyaratan dan batasan untuk Aurora Serverless v2. Tentukan pengaturan berikut untuk memastikan bahwa Anda dapat menambahkan instans DB Aurora Serverless v2 ke klaster:

Wilayah AWS

Buat cluster di Wilayah AWS tempat instance Aurora Serverless v2 DB tersedia. Untuk detail tentang Wilayah yang tersedia, lihat Aurora Serverless v2.

Versi mesin DB

Pilih versi mesin yang kompatibel dengan Aurora Serverless v2. Untuk informasi tentang persyaratan versi Aurora Serverless v2, lihat Persyaratan dan batasan untuk Aurora Serverless v2.

Kelas instans DB

Jika Anda membuat cluster menggunakan AWS Management Console, Anda memilih kelas instans DB untuk instance DB penulis pada saat yang sama. Pilih kelas instans DB Nirserver. Ketika Anda memilih kelas instans DB tersebut, Anda juga menentukan rentang kapasitas untuk instans DB penulis. Rentang kapasitas yang sama berlaku untuk semua instans DB Aurora Serverless v2 lain yang Anda tambahkan ke klaster tersebut.

Jika Anda tidak melihat pilihan Tanpa Server untuk kelas instans DB, pastikan Anda memilih versi mesin DB yang didukung. Aurora Serverless v2

Bila Anda menggunakan AWS CLI atau Amazon RDS API, parameter yang Anda tentukan untuk kelas instans DB adalah`db.serverless`.

Rentang kapasitas

Isi nilai unit kapasitas Aurora (ACU) minimum dan maksimum yang berlaku untuk semua instans DB di klaster tersebut. Opsi ini tersedia di halaman konsol Buat klaster dan Tambahkan pembaca saat Anda memilih Nirserver untuk kelas instans DB.

Jika Anda tidak melihat bidang ACU minimum dan maksimum, pastikan Anda memilih kelas instans DB Tanpa Server untuk instance DB penulis.

Jika Anda awalnya membuat klaster dengan instans DB terprovisi, Anda tidak perlu menentukan ACU minimum dan maksimum. Dalam hal ini Anda dapat memodifikasi klaster sesudahnya untuk

menambahkan pengaturan tersebut. Anda juga dapat menambahkan instans DB pembaca Aurora Serverless v2 ke klaster. Anda menentukan rentang kapasitas sebagai bagian dari proses tersebut.

Sampai Anda menentukan rentang kapasitas untuk klaster Anda, Anda tidak dapat menambahkan instans Aurora Serverless v2 DB apa pun ke cluster menggunakan AWS CLI atau RDS API. Jika Anda mencoba menambahkan instans DB Aurora Serverless v2, Anda mendapatkan pesan kesalahan. Dalam AWS CLI atau prosedur RDS API, rentang kapasitas diwakili oleh `ServerlessV2ScalingConfiguration` atribut.

Untuk klaster yang berisi lebih dari satu instans DB pembaca, prioritas failover dari setiap instans DB pembaca Aurora Serverless v2 memainkan peran penting dalam cara instans DB tersebut menaikkan dan menurunkan skalanya. Anda tidak dapat menentukan prioritas saat membuat klaster pada awalnya. Ingat properti ini saat Anda menambahkan instans DB pembaca kedua atau berikutnya ke klaster Anda. Untuk informasi selengkapnya, lihat Memilih tingkat promosi untuk pembaca Aurora Serverless v2.

Membuat klaster DB Aurora Serverless v2

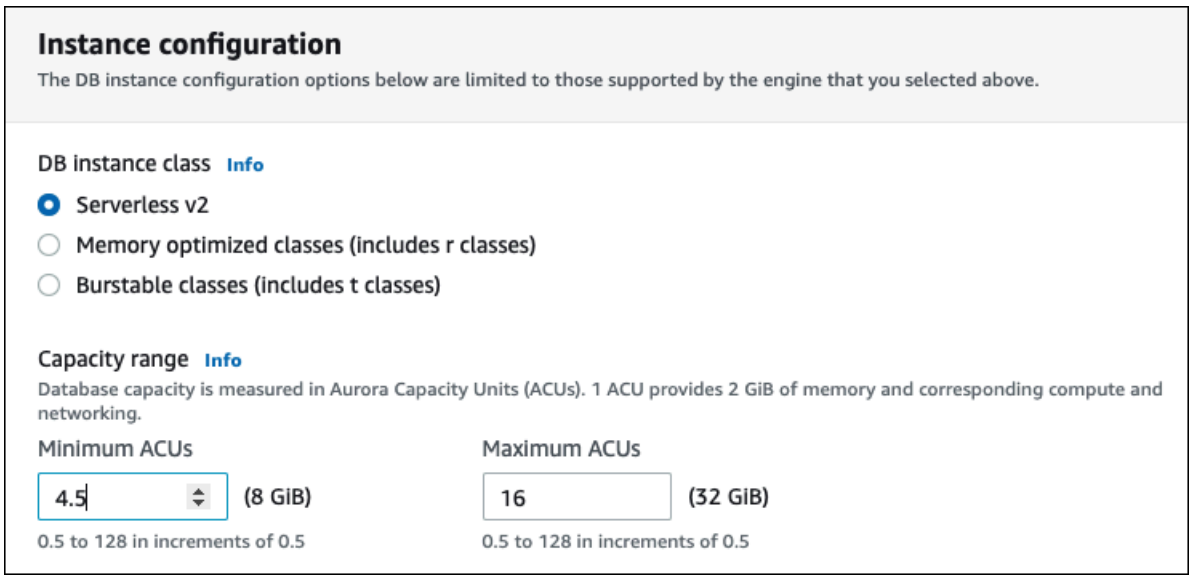
Anda dapat menggunakan AWS Management Console, AWS CLI, atau RDS API untuk membuat cluster Aurora Serverless v2 DB.

Konsol

Untuk membuat klaster dengan penulis Aurora Serverless v2

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih Buat basis data. Di halaman yang muncul, pilih opsi berikut:
   - Untuk Jenis mesin, pilih Aurora (Kompatibel MySQL) atau Aurora (Kompatibel PostgreSQL).
   - Untuk Versi, pilih salah satu versi yang didukung untukAurora Serverless v2.
4. Untuk kelas instans DB, pilih Serverless v2.
5. Untuk rentang Kapasitas, Anda dapat menerima rentang default. Atau, Anda dapat memilih nilai lain untuk unit kapasitas minimum dan maksimum. Anda dapat memilih dari 0,5 ACU minimum hingga 128 ACU maksimum, dengan inkremen 0,5 ACU.

   Untuk informasi selengkapnya tentang unit kapasitas Aurora Serverless v2, lihat Kapasitas Aurora Serverless v2 dan Performa dan penskalaan untuk Aurora Serverless v2.



6. Pilih pengaturan cluster DB lainnya, seperti yang dijelaskan dalamPengaturan untuk klaster Aurora DB.
7. Pilih Buat database untuk membuat cluster Aurora DB Anda dengan instance Aurora Serverless v2 DB sebagai instance penulis, juga dikenal sebagai instans DB utama.

CLI

Untuk membuat cluster DB yang kompatibel dengan instans Aurora Serverless v2 DB menggunakan AWS CLI, Anda mengikuti prosedur CLI di. Membuat klaster DB Amazon Aurora Sertakan parameter berikut dalam perintah `create-db-cluster` Anda:

- --region *AWS_Region_where_Aurora Serverless v2_instances_are_available*
- --engine-version *serverless_v2_compatible_engine_version*
- MinCapacity*--serverless-v2-scaling-configuration = minimum_capacity, = maksimum_kapasitas MaxCapacity*

Contoh berikut menunjukkan pembuatan klaster DB Aurora Serverless v2.

```
aws rds create-db-cluster \
    --db-cluster-identifier my-serverless-v2-cluster \
    --region eu-central-1 \
    --engine aurora-mysql \
    --engine-version 8.0.mysql_aurora.3.04.1 \
    --serverless-v2-scaling-configuration MinCapacity=1,MaxCapacity=4 \
```

```
    --master-username myuser \
    --manage-master-user-password
```

> **Note**
>
> Saat Anda membuat cluster Aurora Serverless v2 DB menggunakan AWS CLI, mode mesin muncul di output sebagai `provisioned` bukan`serverless`. Mode mesin `serverless` mengacu pada Aurora Serverless v1.

Contoh ini menentukan opsi `--manage-master-user-password` untuk menghasilkan kata sandi pengguna master dan mengelolanya di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager. Alternatifnya, Anda dapat menggunakan opsi `--master-password` untuk menentukan dan mengelola kata sandi sendiri.

Untuk informasi tentang persyaratan versi Aurora Serverless v2, lihat Persyaratan dan batasan untuk Aurora Serverless v2. Untuk informasi tentang angka yang diizinkan untuk rentang kapasitas dan apa yang direpresentasikan oleh angka-angka tersebut, lihat Kapasitas Aurora Serverless v2 dan Performa dan penskalaan untuk Aurora Serverless v2.

Untuk memverifikasi apakah klaster yang ada memiliki pengaturan kapasitas yang ditentukan, periksa output dari perintah `describe-db-clusters` untuk atribut `ServerlessV2ScalingConfiguration`. Thatattribute terlihat mirip dengan berikut ini.

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

> **Tip**
>
> Jika Anda tidak menentukan ACU minimum dan maksimum saat membuat klaster, Anda dapat menggunakan perintah `modify-db-cluster` sesudahnya untuk menambahkan pengaturan tersebut. Sebelum Anda melakukannya, Anda tidak dapat menambahkan instans DB Aurora Serverless v2 ke klaster. Jika Anda mencoba menambahkan instans DB `db.serverless`, Anda mendapatkan pesan kesalahan.

API

Untuk membuat klaster DB yang kompatibel dengan instans DB Aurora Serverless v2 menggunakan API RDS, Anda perlu mengikuti prosedur API di Membuat klaster DB Amazon Aurora. Pilih pengaturan berikut: Pastikan bahwa operasi `CreateDBCluster` Anda mencakup parameter berikut:

```
EngineVersion serverless_v2_compatible_engine_version
ServerlessV2ScalingConfiguration with MinCapacity=minimum_capacity and
 MaxCapacity=maximum_capacity
```

Untuk informasi tentang persyaratan versi Aurora Serverless v2, lihat Persyaratan dan batasan untuk Aurora Serverless v2. Untuk informasi tentang angka yang diizinkan untuk rentang kapasitas dan apa yang direpresentasikan oleh angka-angka tersebut, lihat Kapasitas Aurora Serverless v2 dan Performa dan penskalaan untuk Aurora Serverless v2.

Untuk memeriksa apakah klaster yang ada memiliki pengaturan kapasitas yang ditentukan, periksa output dari operasi `DescribeDBClusters` untuk atribut `ServerlessV2ScalingConfiguration`. Atribut tersebut terlihat seperti yang berikut ini.

```
"ServerlessV2ScalingConfiguration": {
    "MinCapacity": 1.5,
    "MaxCapacity": 24.0
}
```

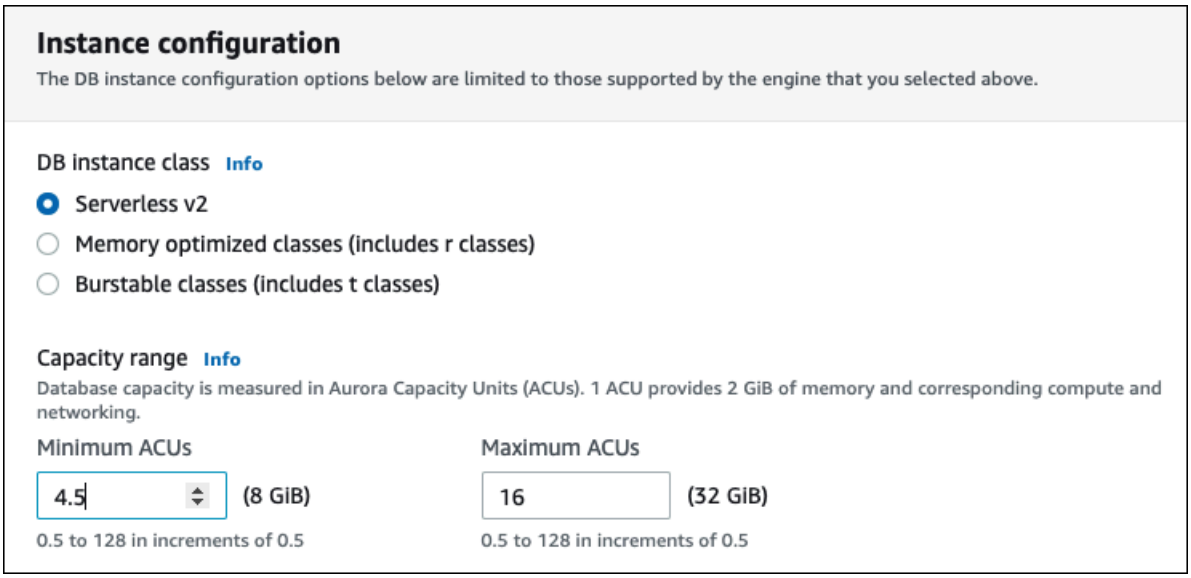
> **Tip**
>
> Jika Anda tidak menentukan ACU minimum dan maksimum saat membuat klaster, Anda dapat menggunakan operasi `ModifyDBCluster` sesudahnya untuk menambahkan pengaturan tersebut. Sebelum Anda melakukannya, Anda tidak dapat menambahkan instans DB Aurora Serverless v2 ke klaster. Jika Anda mencoba menambahkan instans DB `db.serverless`, Anda mendapatkan pesan kesalahan.

Membuat instance DB Aurora Serverless v2 penulis

Konsol

Saat Anda membuat cluster DB menggunakan AWS Management Console, Anda menentukan properti instance DB penulis pada saat yang bersamaan. Untuk membuat instans DB penulis menggunakan Aurora Serverless v2, pilih kelas instans DB Nirserver.

Kemudian Anda perlu mengatur rentang kapasitas untuk klaster dengan menentukan nilai unit kapasitas Aurora (ACU) minimum dan maksimum. Nilai minimum dan maksimum ini berlaku untuk setiap instans DB Aurora Serverless v2 di klaster.



Jika Anda tidak membuat instans DB Aurora Serverless v2 saat pertama kali membuat klaster, Anda dapat menambahkan satu atau beberapa instans DB Aurora Serverless v2 nanti. Untuk melakukannya, ikuti prosedur dalam Menambahkan pembaca Aurora Serverless v2 dan Mengonversi penulis atau pembaca terprovisi menjadi Aurora Serverless v2. Anda menentukan rentang kapasitas saat Anda menambahkan instans DB Aurora Serverless v2 pertama ke klaster. Anda dapat mengubah rentang kapasitas nanti dengan mengikuti prosedur dalam Mengatur rentang kapasitas Aurora Serverless v2 untuk sebuah klaster.

CLI

Saat Anda membuat cluster Aurora Serverless v2 DB menggunakan AWS CLI, Anda secara eksplisit menambahkan instance DB penulis menggunakan perintah. create-db-instance Sertakan parameter berikut:

- `--db-instance-class db.serverless`

Contoh berikut menunjukkan pembuatan instans DB penulis Aurora Serverless v2.

```
aws rds create-db-instance \
    --db-cluster-identifier my-serverless-v2-cluster \
    --db-instance-identifier my-serverless-v2-instance \
    --db-instance-class db.serverless \
    --engine aurora-mysql
```

Mengelola cluster Aurora Serverless v2 DB

Dengan Aurora Serverless v2, klaster Anda dapat dipertukarkan dengan klaster terprovisi. Properti Aurora Serverless v2 berlaku untuk satu atau beberapa instans DB dalam sebuah klaster. Dengan demikian, prosedur untuk membuat klaster, memodifikasi klaster, membuat dan memulihkan snapshot, dan sebagainya, pada dasarnya sama dengan jenis klaster Aurora lainnya. Untuk prosedur umum untuk mengelola klaster Aurora dan instans DB, lihat Mengelola klaster DB Amazon Aurora.

Dalam topik berikut, Anda dapat mempelajari tentang pertimbangan manajemen untuk cluster yang berisi instans Aurora Serverless v2 DB.

Topik

- Mengatur rentang kapasitas Aurora Serverless v2 untuk sebuah klaster
- Memeriksa rentang kapasitas untuk Aurora Serverless v2
- Menambahkan pembaca Aurora Serverless v2
- Mengonversi penulis atau pembaca terprovisi menjadi Aurora Serverless v2
- Mengonversi penulis atau pembaca Aurora Serverless v2 menjadi terprovisi
- Memilih tingkat promosi untuk pembaca Aurora Serverless v2
- Menggunakan TLS/SSL dengan Aurora Serverless v2
- Melihat penulis dan pembaca Aurora Serverless v2
- Logging untuk Aurora Serverless v2

Mengatur rentang kapasitas Aurora Serverless v2 untuk sebuah klaster

Untuk memodifikasi parameter konfigurasi atau pengaturan lain untuk klaster yang berisi instans DB Aurora Serverless v2, atau instans DB itu sendiri, ikuti prosedur umum yang sama seperti untuk klaster terprovisi. Untuk detailnya, lihat Memodifikasi klaster DB Amazon Aurora.

Pengaturan terpenting yang unik untuk Aurora Serverless v2 adalah rentang kapasitas. Setelah Anda menetapkan nilai unit kapasitas Aurora (ACU) minimum dan maksimum untuk klaster Aurora, Anda tidak perlu secara aktif menyesuaikan kapasitas instans DB Aurora Serverless v2 di klaster. Aurora akan melakukannya untuk Anda. Pengaturan ini dikelola pada tingkat klaster. Nilai ACU minimum dan maksimum yang sama berlaku untuk setiap instans DB Aurora Serverless v2 dalam klaster.

Anda dapat mengatur nilai spesifik berikut:

- ACU Minimum – Instans DB Aurora Serverless v2 dapat mengurangi kapasitas hingga mencapai jumlah ACU ini.
- ACU Maksimum – instans DB Aurora Serverless v2 dapat menambah kapasitas hingga mencapai jumlah ACU ini.

> **Note**
>
> Ketika Anda memodifikasi rentang kapasitas untuk klaster DB Aurora Serverless v2, perubahannya akan terjadi dengan segera, terlepas dari apakah Anda memilih untuk menerapkannya langsung atau selama periode pemeliharaan terjadwal berikutnya.

Untuk mengetahui detail tentang efek rentang kapasitas dan cara memantau dan menyesuaikannya, lihat CloudWatch Metrik Amazon penting untuk Aurora Serverless v2 dan Performa dan penskalaan untuk Aurora Serverless v2. Tujuan Anda adalah memastikan bahwa kapasitas maksimum klaster cukup tinggi untuk menangani lonjakan beban kerja, dan kapasitas minimumnya cukup rendah untuk meminimalkan biaya saat klaster tidak sibuk.

Misalnya, berdasarkan pemantauan, Anda menentukan bahwa rentang ACU untuk klaster harus lebih tinggi, lebih rendah, lebih luas, atau lebih sempit. Anda dapat mengatur kapasitas cluster Aurora ke rentang ACU tertentu dengan, API AWS Management Console, atau Amazon RDS. AWS CLI Rentang kapasitas ini berlaku untuk setiap instans DB Aurora Serverless v2 di klaster.

Misalnya, anggaplah klaster Anda memiliki rentang kapasitas 1–16 ACU dan berisi dua instans DB Aurora Serverless v2. Kemudian, klaster ini secara keseluruhan menggunakan antara 2 ACU (saat idle) dan 32 ACU (saat dimanfaatkan sepenuhnya). Jika Anda mengubah rentang kapasitas dari 8 menjadi 20,5 ACU, sekarang klaster ini menggunakan 16 ACU saat idle, dan hingga 41 ACU saat dimanfaatkan sepenuhnya.

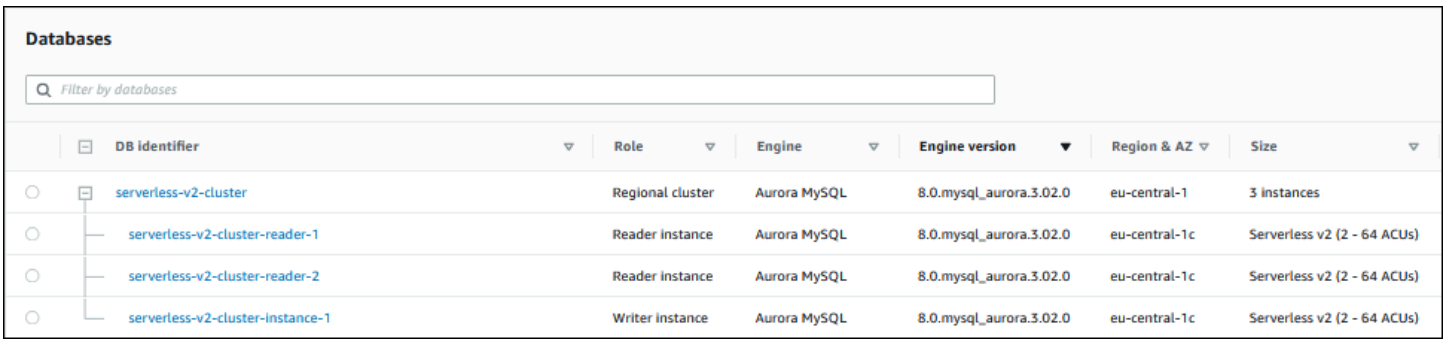
Aurora secara otomatis menetapkan parameter tertentu untuk instans DB Aurora Serverless v2 ke nilai yang didasarkan pada nilai ACU maksimum dalam rentang kapasitas. Untuk mengetahui daftar parameter tersebut, lihat Koneksi maksimum untuk Aurora Serverless v2. Untuk parameter statis yang bergantung pada jenis perhitungan ini, nilainya dievaluasi lagi saat Anda mem-boot ulang instans DB. Dengan demikian, Anda dapat memperbarui nilai parameter tersebut dengan mem-boot ulang instans DB setelah mengubah rentang kapasitas. Untuk memeriksa apakah Anda perlu mem-boot ulang instans DB Anda untuk menerapkan perubahan parameter tersebut, lihat atribut `ParameterApplyStatus` instans DB. Nilai `pending-reboot` menunjukkan bahwa boot ulang akan menerapkan perubahan pada beberapa nilai parameter.

Konsol

Anda dapat mengatur rentang kapasitas klaster yang berisi instans DB Aurora Serverless v2 dengan AWS Management Console.

Saat Anda menggunakan konsol, Anda mengatur rentang kapasitas untuk klaster saat Anda menambahkan instans DB Aurora Serverless v2 pertama ke klaster tersebut. Anda dapat melakukannya saat Anda memilih kelas instans DB Serverless v2 untuk instans DB penulis saat Anda membuat klaster. Anda dapat melakukannya saat Anda memilih kelas instans DB Nirserver saat Anda menambahkan instans DB pembaca Aurora Serverless v2 ke klaster. Anda juga dapat melakukannya saat mengonversi instans DB yang sudah ada di klaster ke kelas instans DB Nirserver. Untuk versi lengkap prosedur tersebut, lihat Membuat instance DB Aurora Serverless v2 penulis, Menambahkan pembaca Aurora Serverless v2, dan Mengonversi penulis atau pembaca terprovisi menjadi Aurora Serverless v2.

Rentang kapasitas apa pun yang Anda tetapkan di tingkat klaster berlaku untuk semua instans DB Aurora Serverless v2 di klaster Anda. Gambar berikut menunjukkan klaster dengan beberapa instans DB pembaca Aurora Serverless v2. Masing-masing memiliki rentang kapasitas yang identik, yaitu 2–64 ACU.

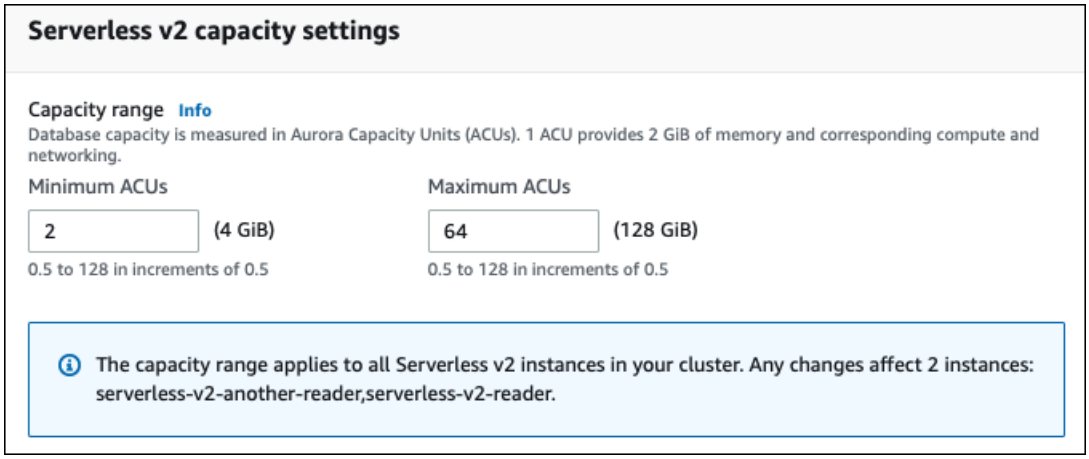


Untuk memodifikasi rentang kapasitas klaster Aurora Serverless v2

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster yang berisi instans DB Aurora Serverless v2 Anda dari daftar. Klaster ini harus sudah berisi setidaknya satu instans DB Aurora Serverless v2. Jika tidak, Aurora tidak menampilkan bagian Rentang kapasitas.
4. Untuk Tindakan, pilih Modifikasi.
5. Di bagian Rentang kapasitas, pilih hal berikut:

   a. Masukkan nilai untuk ACU Minimum. Konsol menampilkan rentang nilai yang diizinkan. Anda dapat memilih dari 0,5 ACU minimum hingga 128 ACU maksimum, dengan inkremen 0,5 ACU.

   b. Masukkan nilai untuk ACU Maksimum. Nilai ini harus lebih besar atau sama dengan ACU Minimum. Konsol menampilkan rentang nilai yang diizinkan. Gambar berikut menunjukkan pilihan tersebut.



6. Pilih Lanjutkan. Halaman Ringkasan modifikasi akan muncul.
7. Pilih Terapkan segera.

   Perubahan kapasitas terjadi segera, terlepas dari apakah Anda memilih untuk menerapkannya segera atau selama periode pemeliharaan terjadwal berikutnya.
8. Pilih Ubah klaster untuk menerima ringkasan modifikasi tersebut. Anda juga dapat memilih Kembali untuk memodifikasi perubahan atau Batalkan untuk menghapus perubahan Anda.

AWS CLI

Untuk mengatur kapasitas cluster tempat Anda ingin menggunakan instance Aurora Serverless v2 DB menggunakan AWS CLI, jalankan modify-db-cluster AWS CLI perintah. Tentukan opsi `--serverless-v2-scaling-configuration`. Klaster ini mungkin sudah berisi satu atau beberapa instans DB Aurora Serverless v2, atau Anda dapat menambahkan instans DB nanti. Nilai valid untuk bidang `MinCapacity` dan `MaxCapacity` mencakup hal berikut:

- `0.5`, `1`, `1.5`, `2`, dan sebagainya, dalam inkremen 0,5, hingga maksimum 128.

Dalam contoh ini, Anda mengatur rentang ACU dari klaster DB Aurora bernama `sample-cluster` hingga minimum `48.5` dan maksimum 64.

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--serverless-v2-scaling-configuration MinCapacity=48.5,MaxCapacity=64
```

Perubahan kapasitas terjadi segera, terlepas dari apakah Anda memilih untuk menerapkannya segera atau selama periode pemeliharaan terjadwal berikutnya.

Setelah melakukannya, Anda dapat menambahkan instans DB Aurora Serverless v2 ke klaster dan setiap instans DB baru dapat diskalakan antara 48,5 dan 64 ACU. Rentang kapasitas baru juga berlaku untuk semua instans DB Aurora Serverless v2 yang sudah ada di klaster. Instans DB menaikkan dan menurunkan skala jika perlu agar berada dalam rentang kapasitas baru.

Untuk contoh tambahan tentang pengaturan rentang kapasitas menggunakan CLI, lihat Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora.

Untuk memodifikasi konfigurasi penskalaan cluster Aurora Serverless DB menggunakan AWS CLI, jalankan modify-db-cluster AWS CLI perintah. Tentukan opsi `--serverless-v2-scaling-configuration` untuk mengonfigurasi kapasitas minimum dan kapasitas maksimum. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: `0.5`, `1`, `1.5`, `2`, dan sebagainya, dalam inkremen sebesar 0,5 ACU hingga maksimum `128`.
- Aurora PostgreSQL: `0.5`, `1`, `1.5`, `2`, dan sebagainya, dalam inkremen sebesar 0,5 ACU hingga maksimum `128`.

Dalam contoh berikut, Anda memodifikasi konfigurasi penskalaan instans DB Aurora Serverless v2 bernama `sample-instance` yang merupakan bagian dari klaster bernama `sample-cluster`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

Untuk Windows:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster ^
--serverless-v2-scaling-configuration MinCapacity=8,MaxCapacity=64
```

API RDS

Anda dapat mengatur kapasitas instans DB Aurora dengan operasi API ModifyDBCluster. Tentukan parameter `ServerlessV2ScalingConfiguration`. Nilai valid untuk bidang `MinCapacity` dan `MaxCapacity` mencakup hal berikut:

- `0.5`, `1`, `1.5`, `2`, dan sebagainya, dalam inkremen 0,5, hingga maksimum 128.

Anda dapat mengubah konfigurasi penskalaan klaster yang berisi instans DB Aurora Serverless v2 dengan operasi API ModifyDBCluster. Tentukan parameter `ServerlessV2ScalingConfiguration` untuk mengonfigurasi kapasitas minimum dan kapasitas maksimum. Nilai kapasitas yang valid mencakup hal berikut:

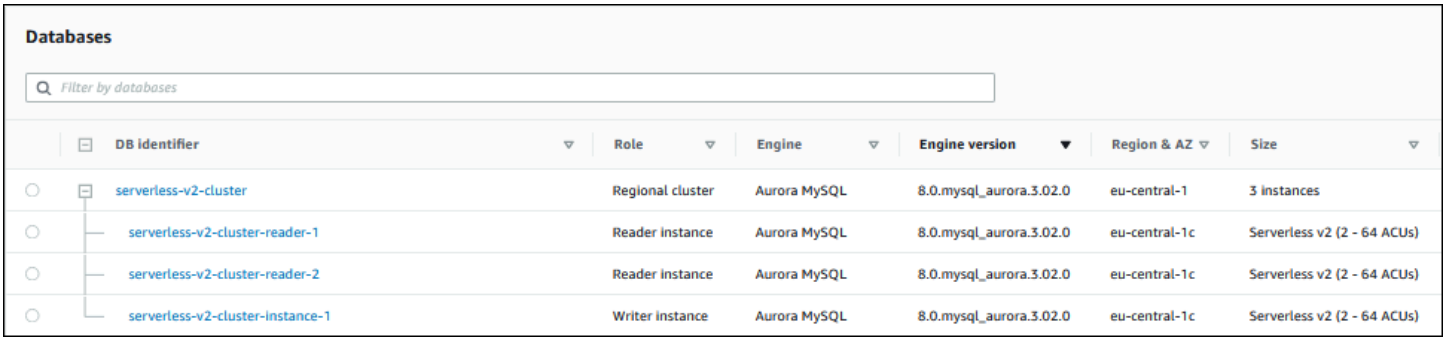
- Aurora MySQL: `0.5`, `1`, `1.5`, `2`, dan sebagainya, dalam inkremen sebesar 0,5 ACU hingga maksimum `128`.
- Aurora PostgreSQL: `0.5`, `1`, `1.5`, `2`, dan sebagainya, dalam inkremen sebesar 0,5 ACU hingga maksimum `128`.

Perubahan kapasitas terjadi segera, terlepas dari apakah Anda memilih untuk menerapkannya segera atau selama periode pemeliharaan terjadwal berikutnya.

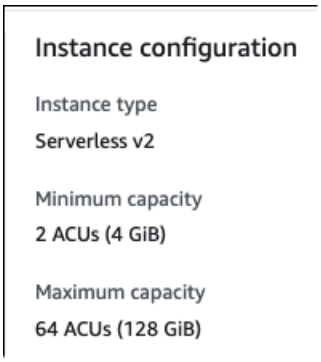
Memeriksa rentang kapasitas untuk Aurora Serverless v2

Prosedur untuk memeriksa rentang kapasitas untuk klaster Aurora Serverless v2 Anda mengharuskan Anda menetapkan rentang kapasitas terlebih dahulu. Jika Anda belum melakukannya, ikuti prosedur dalam Mengatur rentang kapasitas Aurora Serverless v2 untuk sebuah klaster.

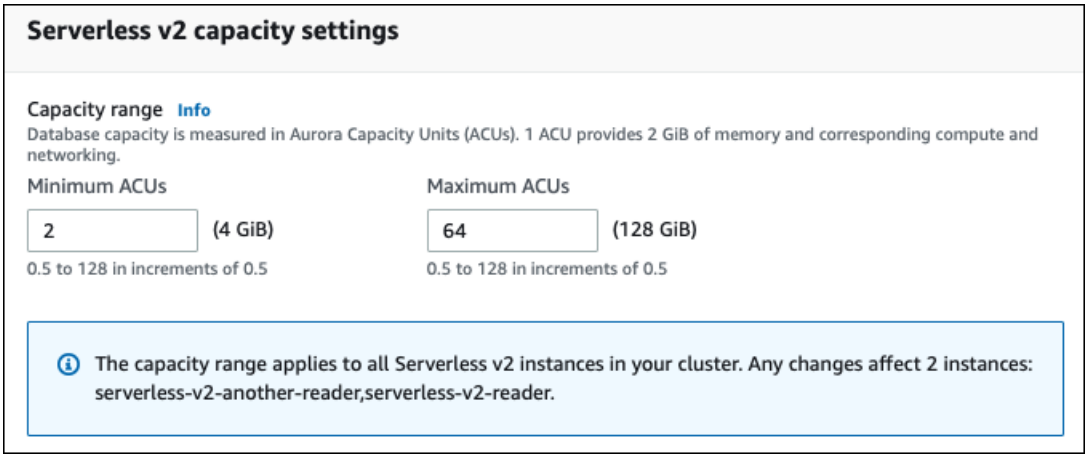
Rentang kapasitas apa pun yang Anda tetapkan di tingkat klaster berlaku untuk semua instans DB Aurora Serverless v2 di klaster Anda. Gambar berikut menunjukkan klaster dengan beberapa instans DB Aurora Serverless v2. Masing-masing memiliki rentang kapasitas yang identik.



Anda juga dapat melihat halaman detail untuk instans DB Aurora Serverless v2 apa pun di klaster. Rentang kapasitas instans DB muncul di tab Konfigurasi.



Anda juga dapat melihat rentang kapasitas saat ini untuk klaster pada halaman Ubah untuk klaster. Gambar berikut menunjukkan caranya. Pada tahap ini, Anda dapat mengubah rentang kapasitas. Untuk semua cara yang dapat Anda gunakan untuk mengatur atau mengubah rentang kapasitas, lihat Mengatur rentang kapasitas Aurora Serverless v2 untuk sebuah klaster.



Memeriksa rentang kapasitas saat ini untuk klaster Aurora

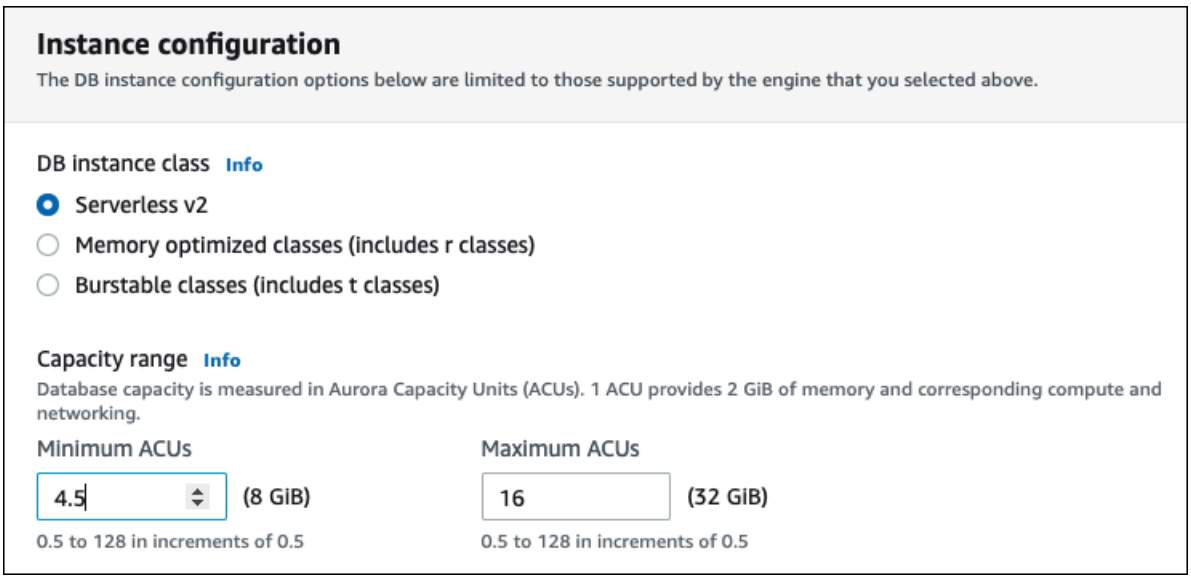
Anda dapat memeriksa rentang kapasitas yang dikonfigurasi untuk instans DB Aurora Serverless v2 di klaster dengan memeriksa atribut `ServerlessV2ScalingConfiguration` untuk klaster. Contoh AWS CLI berikut menunjukkan klaster dengan kapasitas minimum 0,5 unit kapasitas Aurora (ACU) dan kapasitas maksimum 16 ACU.

```
$ aws rds describe-db-clusters --db-cluster-identifier serverless-v2-64-acu-cluster \
  --query 'DBClusters[*].[ServerlessV2ScalingConfiguration]'
[
    [
        {
            "MinCapacity": 0.5,
            "MaxCapacity": 16.0
        }
    ]
]
```

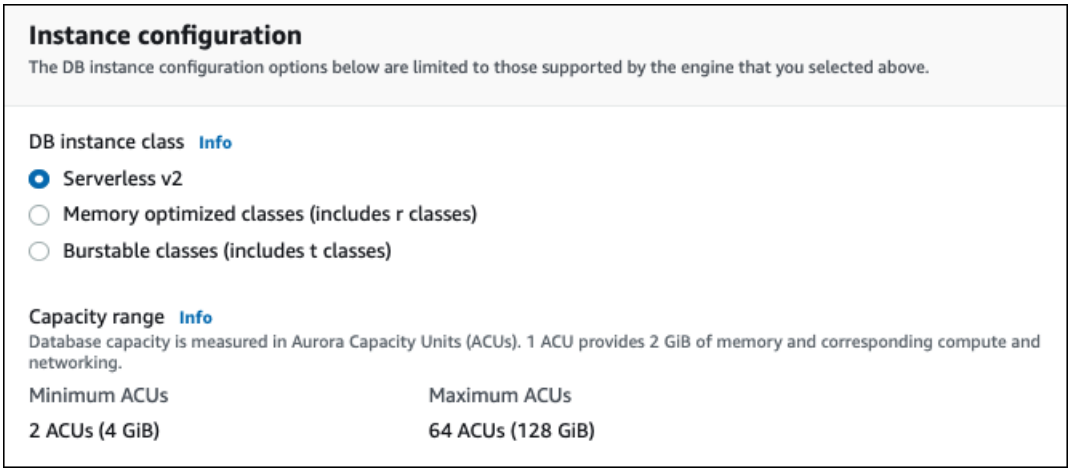
Menambahkan pembaca Aurora Serverless v2

Untuk menambahkan instans DB pembaca Aurora Serverless v2 ke klaster Anda, Anda mengikuti prosedur umum yang sama seperti dalam Menambahkan Replika Aurora ke klaster DB. Pilih kelas instans Serverless v2 untuk instans DB baru.

Jika instans DB pembaca adalah instans DB Aurora Serverless v2 pertama di klaster, Anda juga perlu memilih rentang kapasitas. Gambar berikut menunjukkan kontrol yang Anda gunakan untuk menentukan unit kapasitas Aurora (ACU) minimum dan maksimum. Pengaturan ini berlaku untuk instans DB pembaca ini dan instans DB Aurora Serverless v2 lainnya yang ditambahkan ke klaster. Setiap instans DB Aurora Serverless v2 dapat diskalakan antara nilai ACU minimum dan maksimum.



Jika Anda sudah menambahkan instans DB Aurora Serverless v2 apa pun ke klaster, rentang kapasitas saat ini akan ditampilkan saat menambahkan instans DB pembaca Aurora Serverless v2 lain. Gambar berikut menunjukkan kontrol hanya baca tersebut.



Jika Anda ingin mengubah rentang kapasitas klaster, ikuti prosedur dalam Mengatur rentang kapasitas Aurora Serverless v2 untuk sebuah klaster.

Untuk klaster yang berisi lebih dari satu instans DB pembaca, prioritas failover dari setiap instans DB pembaca Aurora Serverless v2 memainkan peran penting dalam cara instans DB tersebut menaikkan dan menurunkan skalanya. Anda tidak dapat menentukan prioritas saat membuat klaster pada awalnya. Ingat properti ini saat Anda menambahkan pembaca kedua atau instans DB yang lebih baru ke klaster Anda. Untuk informasi selengkapnya, lihat Memilih tingkat promosi untuk pembaca Aurora Serverless v2.

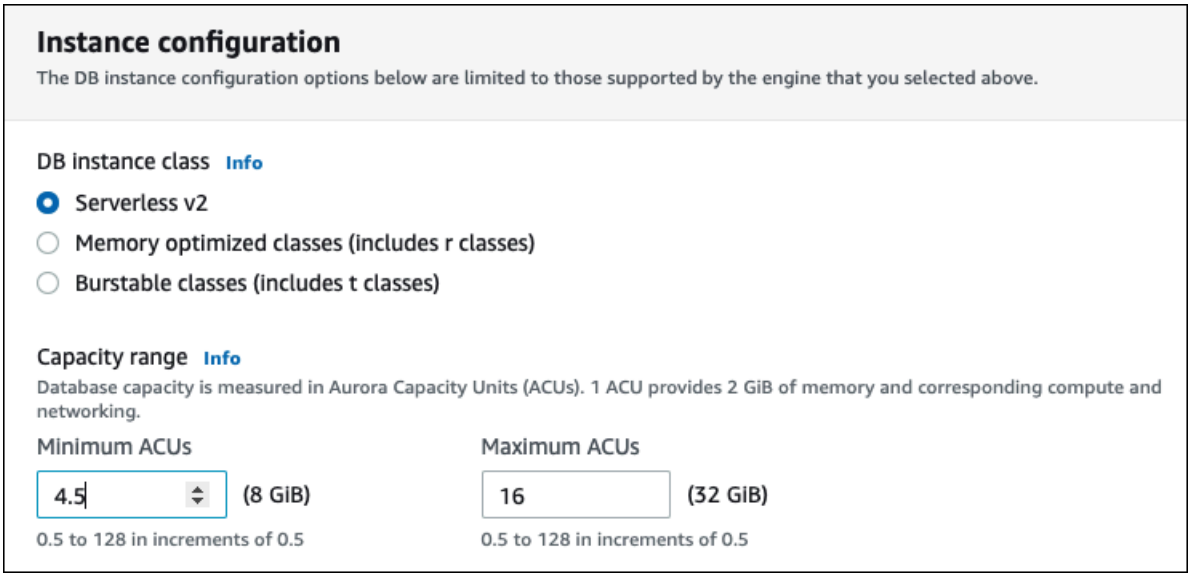
Untuk cara lain agar Anda dapat melihat rentang kapasitas saat ini untuk sebuah klaster, lihat Memeriksa rentang kapasitas untuk Aurora Serverless v2.

Mengonversi penulis atau pembaca terprovisi menjadi Aurora Serverless v2

Anda dapat mengonversi instans DB terprovisi untuk menggunakan Aurora Serverless v2. Untuk melakukannya, ikuti prosedur dalam Memodifikasi instans DB dalam klaster DB. Klaster harus memenuhi persyaratan dalam Persyaratan dan batasan untuk Aurora Serverless v2. Misalnya, instans DB Aurora Serverless v2 mengharuskan klaster menjalankan versi mesin minimum tertentu.

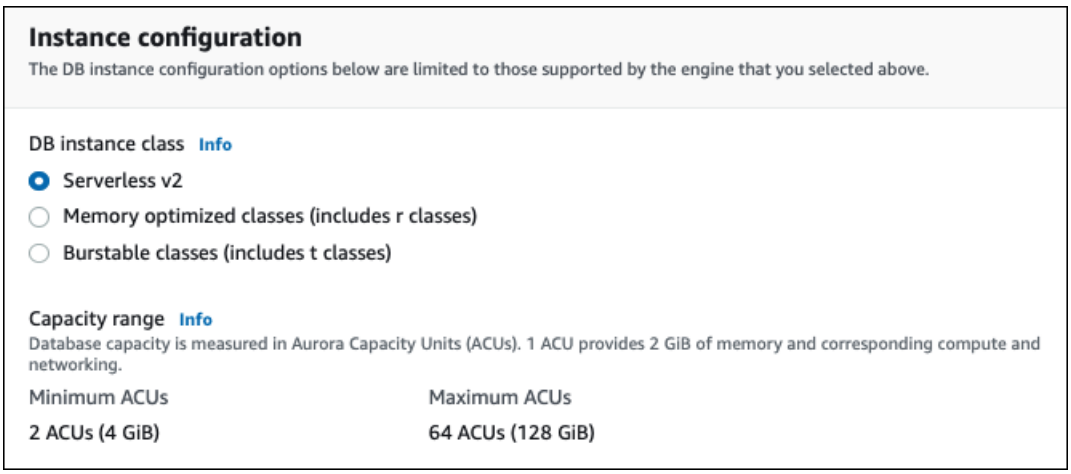
Misalkan Anda mengonversi klaster terprovisi yang sedang berjalan untuk memanfaatkan Aurora Serverless v2. Dalam hal ini, Anda dapat meminimalkan waktu henti dengan mengonversi instans DB menjadi Aurora Serverless v2 sebagai langkah pertama dalam proses switchover. Untuk prosedur lengkapnya, lihat Beralih dari klaster terprovisi ke Aurora Serverless v2.

Jika instans DB yang Anda konversi adalah instans DB Aurora Serverless v2 pertama di klaster, Anda perlu memilih rentang kapasitas untuk klaster sebagai bagian dari operasi Ubah. Rentang kapasitas ini berlaku untuk setiap instans DB Aurora Serverless v2 yang ditambahkan ke klaster. Gambar berikut menunjukkan halaman tempat Anda menentukan unit kapasitas Aurora (ACU) minimum dan maksimum.



Untuk detail tentang pentingnya rentang kapasitas, lihat Kapasitas Aurora Serverless v2.

Jika klaster sudah berisi satu atau beberapa instans DB Aurora Serverless v2, Anda akan melihat rentang kapasitas yang ada selama operasi Ubah. Gambar berikut menunjukkan contoh panel informasi tersebut.



Jika Anda ingin mengubah rentang kapasitas untuk klaster setelah Anda menambahkan instans DB Aurora Serverless v2 lainnya, ikuti prosedurnya dalam Mengatur rentang kapasitas Aurora Serverless v2 untuk sebuah klaster.

Mengonversi penulis atau pembaca Aurora Serverless v2 menjadi terprovisi

Anda dapat mengonversi instans DB Aurora Serverless v2 menjadi instans DB terprovisi. Untuk melakukannya, ikuti prosedur dalam Memodifikasi instans DB dalam klaster DB. Pilih kelas instans DB selain Nirserver.

Anda dapat mengonversi instans DB Aurora Serverless v2 menjadi terprovisi jika memerlukan kapasitas yang lebih besar daripada yang tersedia dengan unit kapasitas Aurora (ACU) maksimum dari instans DB Aurora Serverless v2. Misalnya, kelas instans DB db.r5 dan db.r6g terbesar memiliki kapasitas memori yang lebih besar daripada yang dapat dicapai oleh instans DB Aurora Serverless v2 yang diskalakan.

> **Tip**
>
> Beberapa kelas instans DB yang lebih lama seperti db.r3 dan db.t2 tidak tersedia untuk versi Aurora yang Anda gunakan dengan Aurora Serverless v2. Untuk melihat kelas instans DB mana yang dapat Anda gunakan saat mengonversi instans DB Aurora Serverless v2 menjadi instans terprovisi, lihat Mesin DB yang didukung untuk kelas instans DB.

Jika Anda mengonversi instans DB penulis untuk klaster Anda dari Aurora Serverless v2 menjadi terprovisi, Anda dapat mengikuti prosedur dalam Beralih dari klaster terprovisi ke Aurora Serverless v2, tetapi secara terbalik. Alihkan salah satu instans DB pembaca di klaster dari Aurora Serverless v2

menjadi terprovisi. Kemudian lakukan failover untuk membuat instans DB terprovisi tersebut menjadi penulis.

Rentang kapasitas apa pun yang sebelumnya Anda tentukan untuk klaster akan tetap ada, bahkan jika semua instans DB Aurora Serverless v2 dihapus dari klaster. Jika Anda ingin mengubah rentang kapasitas, Anda dapat memodifikasi klaster, seperti yang dijelaskan dalam Mengatur rentang kapasitas Aurora Serverless v2 untuk sebuah klaster.

Memilih tingkat promosi untuk pembaca Aurora Serverless v2

Untuk klaster yang berisi beberapa instans DB Aurora Serverless v2 atau gabungan instans terprovisi dan instans DB Aurora Serverless v2, perhatikan pengaturan tingkat promosi untuk setiap instans DB Aurora Serverless v2. Pengaturan ini mengontrol lebih banyak perilaku untuk instans DB Aurora Serverless v2 dibandingkan dengan instans DB terprovisi.

Dalam AWS Management Console, Anda menentukan pengaturan ini menggunakan pilihan prioritas Failover di bawah Konfigurasi tambahan untuk Buat database, Modify instance, dan Add reader pages. Anda melihat properti ini untuk instans DB yang ada di kolom Tingkat prioritas opsional pada halaman Basis data. Anda juga dapat melihat properti ini di halaman detail untuk klaster DB atau instans DB.

Untuk instans DB terprovisi, pilihan tingkat 0–15 hanya menentukan urutan yang digunakan Aurora dalam memilih instans DB pembaca mana yang akan dipromosikan menjadi penulis selama operasi failover. Untuk instans DB pembaca Aurora Serverless v2, nomor tingkat juga menentukan apakah instans DB menaikkan skalanya agar sesuai dengan kapasitas instans DB penulis atau diskalakan secara independen berdasarkan beban kerjanya sendiri. Instans DB pembaca Aurora Serverless v2 di tingkat 0 atau 1 dipertahankan pada kapasitas minimum setidaknya setinggi instans DB penulis. Dengan begitu, instans DB pembaca tersebut akan siap mengambil alih dari instans DB penulis jika terjadi failover. Jika instans DB penulis adalah instans DB terprovisi, Aurora memperkirakan kapasitas Aurora Serverless v2 yang setara. Aurora menggunakan perkiraan tersebut sebagai kapasitas minimum untuk instans DB pembaca Aurora Serverless v2.

Instans DB pembaca Aurora Serverless v2 di tingkat 2–15 tidak memiliki batasan yang sama pada kapasitas minimumnya. Saat idle, instans DB ini dapat diturunkan skalanya ke nilai unit kapasitas Aurora (ACU) minimum yang ditentukan dalam rentang kapasitas klaster.

AWS CLI Contoh Linux berikut menunjukkan cara memeriksa tingkatan promosi semua instans DB di cluster Anda. Bidang akhir menyertakan nilai `True` untuk instans DB penulis dan `False` untuk semua instans DB pembaca.

```
$ aws rds describe-db-clusters --db-cluster-identifier promotion-tier-demo \
  --query 'DBClusters[*].DBClusterMembers[*].
[PromotionTier,DBInstanceIdentifier,IsClusterWriter]' \
  --output text

1   instance-192  True
1   tier-01-4840  False
10  tier-10-7425  False
15  tier-15-6694  False
```

AWS CLI Contoh Linux berikut menunjukkan cara mengubah tingkat promosi instans DB tertentu di cluster Anda. Perintah ini pertama-tama memodifikasi instans DB dengan tingkat promosi baru. Kemudian perintah ini akan menunggu instans DB tersedia lagi dan mengonfirmasi tingkat promosi baru untuk instans DB.

```
$ aws rds modify-db-instance --db-instance-identifier instance-192 --promotion-tier 0
$ aws rds wait db-instance-available --db-instance-identifier instance-192
$ aws rds describe-db-instances --db-instance-identifier instance-192 \
  --query '*[].[PromotionTier]' --output text
0
```

Untuk panduan selengkapnya tentang menentukan tingkat promosi untuk kasus penggunaan yang berbeda-beda, lihat Penskalaan Aurora Serverless v2.

Menggunakan TLS/SSL dengan Aurora Serverless v2

Aurora Serverless v2 dapat menggunakan protokol Keamanan Lapisan Pengangkutan/Lapisan Soket Aman (TLS/SSL) untuk mengenkripsi komunikasi antara klien dan instans DB Aurora Serverless v2 Anda. Layanan ini mendukung protokol TLS/SSL versi 1.0, 1.1, dan 1.2. Untuk informasi umum tentang penggunaan TLS/SSL dengan Aurora, lihat Menggunakan TLS dengan klaster DB Aurora MySQL.

Untuk mempelajari selengkapnya tentang menghubungkan ke basis data Aurora MySQL dengan klien MySQL, lihat Menghubungkan ke instans DB yang menjalankan mesin basis data MySQL.

Aurora Serverless v2 mendukung semua mode TLS/SSL yang tersedia untuk klien MySQL (`mysql`) dan klien PostgreSQL (`psql`), termasuk yang tercantum dalam tabel berikut.

Deskripsi mode TLS/SSL	mysql	psql
Hubungkan tanpa menggunakan TLS/SSL.	DISABLED	disable
Mencoba koneksi menggunakan TLS/SSL terlebih dahulu, tetapi kembali ke non-SSL jika diperlukan.	PREFERRED	prefer (default)
Memberlakukan penggunaan TLS/SSL.	REQUIRED	require
Memberlakukan TLS/SSL dan memverifikasi otorisasi sertifikat (CA).	VERIFY_CA	verify-ca
Memberlakukan TLS/SSL, memverifikasi CA, dan memverifikasi nama host CA.	VERIFY_IDENTITY	verify-full

Aurora Serverless v2 menggunakan sertifikat wildcard. Jika Anda menentukan opsi "memverifikasi CA" atau "memverifikasi CA dan nama host CA" saat menggunakan TLS/SSL, pertama-tama unduh trust store Amazon root CA 1 dari Amazon Trust Services. Setelah melakukannya, Anda dapat mengidentifikasi file berformat PEM ini dalam perintah klien Anda. Untuk melakukannya menggunakan PostgreSQL, lakukan hal berikut.

Untuk Linux, macOS, atau Unix:

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

Untuk mempelajari selengkapnya tentang mengoperasikan basis data Aurora PostgreSQL menggunakan klien Postgres, lihat Menghubungkan ke instans DB yang menjalankan mesin basis data PostgreSQL.

Untuk informasi selengkapnya tentang menghubungkan ke klaster DB Aurora secara umum, lihat Menghubungkan ke klaster DB Amazon Aurora.

Cipher suite yang didukung untuk koneksi ke klaster DB Aurora Serverless v2

Dengan menggunakan cipher suite yang dapat dikonfigurasi, Anda dapat memiliki kontrol lebih besar atas keamanan koneksi basis data Anda. Anda dapat menentukan daftar cipher suite yang ingin Anda izinkan untuk mengamankan koneksi TLS/SSL klien ke basis data Anda. Dengan cipher suite yang dapat dikonfigurasi, Anda dapat mengontrol enkripsi koneksi yang diterima server basis data Anda. Tindakan ini mencegah penggunaan cipher yang tidak aman atau yang sudah ditiadakan.

Klaster DB Aurora Serverless v2 yang didasarkan pada Aurora MySQL mendukung cipher suite yang sama dengan klaster DB terprovisi Aurora MySQL. Untuk informasi tentang cipher suite ini, lihat Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora MySQL.

Klaster DB Aurora Serverless v2 yang didasarkan pada Aurora PostgreSQL mendukung cipher suite yang sama dengan klaster DB terprovisi Aurora PostgreSQL. Untuk informasi tentang cipher suite ini, lihat Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora PostgreSQL.

Melihat penulis dan pembaca Aurora Serverless v2

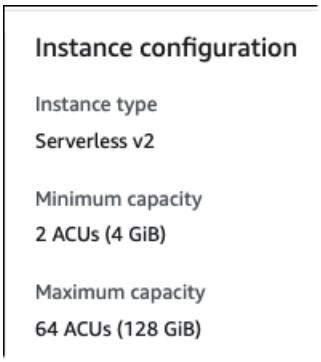
Anda dapat melihat detail instans DB Aurora Serverless v2 dengan cara yang sama seperti yang Anda lakukan untuk instans DB terprovisi. Untuk melakukannya, ikuti prosedur umum dari Melihat klaster DB Amazon Aurora. Sebuah klaster mungkin berisi seluruh instans DB Aurora Serverless v2, seluruh instans DB terprovisi, atau beberapa dari masing-masing.

Setelah membuat satu atau beberapa instans DB Aurora Serverless v2, Anda dapat melihat instans DB mana yang memiliki jenis Nirserver dan mana yang memiliki jenis Instans. Anda juga dapat melihat unit kapasitas Aurora (ACU) minimum dan maksimum yang dapat digunakan instans DB Aurora Serverless v2. Setiap ACU merupakan kombinasi dari kapasitas pemrosesan (CPU) dan memori (RAM). Rentang kapasitas ini berlaku untuk setiap instans DB Aurora Serverless v2 di klaster. Untuk prosedur untuk memeriksa rentang kapasitas klaster atau instans DB Aurora Serverless v2 apa pun di klaster, lihat Memeriksa rentang kapasitas untuk Aurora Serverless v2.

Dalam AWS Management Console, instance Aurora Serverless v2 DB ditandai di bawah kolom Ukuran di halaman Database. Instans DB terprovisi menunjukkan nama kelas instans DB seperti r6g.xlarge. Instans DB Aurora Serverless menunjukkan Nirserver untuk kelas instans DB, bersama dengan kapasitas minimum dan maksimum instans DB. Misalnya, Anda mungkin melihat Serverless v2 (4–64 ACU) atau Serverless v2 (1–40 ACU).

Anda dapat menemukan informasi yang sama pada tab Konfigurasi untuk setiap instans DB Aurora Serverless v2 di konsol. Misalnya, Anda mungkin melihat bagian Tipe instans seperti berikut ini. Di

sini, nilai Tipe instans adalah Serverless v2, nilai Kapasitas minimum adalah 2 ACU (4 GiB), dan nilai Kapasitas maksimum adalah 64 ACU (128 GiB).



Anda dapat memantau kapasitas setiap instans DB Aurora Serverless v2 dari waktu ke waktu. Dengan begitu, Anda dapat memeriksa ACU minimum, maksimum, dan rata-rata yang dikonsumsi oleh setiap instans DB. Anda juga dapat memeriksa seberapa dekat instans DB dengan kapasitas minimum atau maksimumnya. Untuk melihat detail tersebut di AWS Management Console, periksa grafik CloudWatch metrik Amazon pada tab Monitoring untuk instans DB. Untuk informasi tentang metrik yang perlu dilihat dan cara menafsirkannya, lihat CloudWatch Metrik Amazon penting untuk Aurora Serverless v2.

Logging untuk Aurora Serverless v2

Untuk mengaktifkan logging basis data, Anda menentukan log yang akan diaktifkan menggunakan parameter konfigurasi di grup parameter kustom Anda.

Untuk Aurora MySQL, Anda dapat mengaktifkan log berikut.

Aurora MySQL	Deskripsi
`general_log`	Membuat log umum. Atur ke 1 untuk mengaktif kan. Default-nya adalah nonaktif (0).
`log_queries_not_using_indexes`	Mencatat setiap kueri ke log kueri lambat yang tidak menggunakan indeks. Default-nya adalah nonaktif (0). Atur ke 1 untuk mengaktifkan log ini.
`long_query_time`	Mencegah kueri yang berjalan cepat agar tidak dicatat dalam log kueri lambat. Dapat diatur ke

Aurora MySQL	Deskripsi
	angka float antara 0 dan 31536000. Default-nya adalah 0 (tidak aktif).
`server_audit_events`	Daftar peristiwa untuk dicatat dalam log. Nilai yang didukung adalah `CONNECT`, `QUERY`, `QUERY_DCL` , `QUERY_DDL` , `QUERY_DML` , dan `TABLE`.
`server_audit_logging`	Atur ke 1 untuk mengaktifkan logging audit server. Jika Anda mengaktifkan ini, Anda dapat menentukan peristiwa audit yang akan dikirim CloudWatch dengan mencantumkannya di `server_audit_events` parameter.
`slow_query_log`	Membuat log kueri lambat. Atur ke 1 untuk mengaktifkan log kueri lambat. Default-nya adalah nonaktif (0).

Untuk informasi selengkapnya, lihat Menggunakan Audit Lanjutan dengan klaster DB Amazon Aurora MySQL.

Untuk Aurora PostgreSQL, Anda dapat mengaktifkan log berikut pada instans DB Aurora Serverless v2 Anda.

Aurora PostgreSQL	Deskripsi
`log_connections`	Mencatat log setiap koneksi yang berhasil.
`log_disconnections`	Mencatat log terkait akhir sebuah sesi, termasuk durasinya.
`log_lock_waits`	Default-nya adalah 0 (nonaktif). Atur ke 1 untuk mencatat log peristiwa tunggu kunci.

Aurora PostgreSQL	Deskripsi
`log_min_duration_statement`	Durasi minimum (dalam milidetik) untuk menjalankan pernyataan sebelum dicatat lognya.
`log_min_messages`	Mengatur tingkat pesan yang dicatat lognya. Nilai yang didukung adalah `debug5`, `debug4`, `debug3`, `debug2`, `debug1`, `info`, `notice`, `warning`, `error`, `log`, `fatal`, `panic`. Untuk mencatat data performa ke log `postgres`, tetapkan nilainya ke `debug1`.
`log_temp_files`	Mencatat log penggunaan file sementara yang melampaui kilobyte (kB) yang ditentukan.
`log_statement`	Mengontrol pernyataan SQL tertentu yang dicatat lognya. Nilai yang didukung adalah `none`, `ddl`, `mod`, dan `all`. Default-nya adalah `none`.

Topik

- Logging dengan Amazon CloudWatch
- Melihat Aurora Serverless v2 log di Amazon CloudWatch
- Kapasitas pemantauan dengan Amazon CloudWatch

Logging dengan Amazon CloudWatch

Setelah Anda menggunakan prosedur Logging untuk Aurora Serverless v2 untuk memilih log database mana yang akan diaktifkan, Anda dapat memilih log mana yang akan diunggah (“publikasikan”) ke Amazon CloudWatch.

Anda dapat menggunakan Amazon CloudWatch untuk menganalisis data log, membuat alarm, dan melihat metrik. Secara default, log kesalahan untuk Aurora Serverless v2 diaktifkan dan secara otomatis diunggah ke CloudWatch. Anda juga dapat mengunggah log lain dari instans Aurora Serverless v2 DB ke CloudWatch.

Kemudian Anda memilih log mana yang akan diunggah CloudWatch, dengan menggunakan pengaturan ekspor Log di AWS Management Console atau `--enable-cloudwatch-logs-exports` opsi di AWS CLI.

Anda dapat memilih Aurora Serverless v2 log mana yang akan diunggah CloudWatch. Untuk informasi selengkapnya, lihat Menggunakan Audit Lanjutan dengan klaster DB Amazon Aurora MySQL.

Sama seperti jenis apa pun dari klaster DB Aurora, Anda tidak dapat memodifikasi grup parameter klaster DB default. Sebagai gantinya, buat grup parameter klaster DB Anda sendiri berdasarkan parameter default untuk klaster DB dan jenis mesin Anda. Kami sarankan Anda membuat grup parameter klaster DB kustom Anda sebelum membuat klaster DB Aurora Serverless v2 Anda, agar tersedia untuk dipilih saat membuat basis data pada konsol.

> **Note**
>
> Untuk Aurora Serverless v2, Anda dapat membuat klaster DB dan grup parameter DB. Ini kontras dengan Aurora Serverless v1, yang hanya memungkinkan Anda membuat grup parameter klaster DB.

Melihat Aurora Serverless v2 log di Amazon CloudWatch

Setelah Anda menggunakan prosedur dalam Logging dengan Amazon CloudWatch untuk memilih log basis data mana yang akan diaktifkan, Anda dapat melihat konten log.

Untuk informasi lebih lanjut tentang penggunaan CloudWatch dengan log Aurora MySQL dan Aurora PostgreSQL, lihat dan. Memantau peristiwa log di Amazon CloudWatch Menerbitkan log Aurora PostgreSQL ke Amazon Logs CloudWatch

Untuk melihat log untuk klaster DB Aurora Serverless v2 Anda

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Pilih Anda Wilayah AWS.
3. Pilih Grup log.
4. Pilih log klaster DB Aurora Serverless v2 Anda dari daftar. Pola penamaan log adalah sebagai berikut.

   ```
   /aws/rds/cluster/cluster-name/log_type
   ```

> **Note**
>
> Untuk klaster DB Aurora Serverless v2 yang kompatibel dengan Aurora MySQL, log kesalahan menyertakan peristiwa penskalaan pool buffer meskipun tidak ada kesalahan.

Kapasitas pemantauan dengan Amazon CloudWatch

DenganAurora Serverless v2, Anda dapat menggunakannya CloudWatch untuk memantau kapasitas dan pemanfaatan semua instans Aurora Serverless v2 DB di cluster Anda. Anda dapat melihat metrik tingkat instans untuk memeriksa kapasitas setiap instans DB Aurora Serverless v2 saat menaikkan dan menurunkan skalanya. Anda juga dapat membandingkan metrik terkait kapasitas dengan metrik lain untuk melihat bagaimana perubahan beban kerja memengaruhi konsumsi sumber daya. Misalnya, Anda dapat membandingkan `ServerlessDatabaseCapacity` dengan `DatabaseUsedMemory`, `DatabaseConnections`, dan `DMLThroughput` untuk menilai bagaimana klaster DB Anda merespons selama operasi. Untuk detail tentang metrik terkait kapasitas yang berlaku untuk Aurora Serverless v2, lihat CloudWatch Metrik Amazon penting untuk Aurora Serverless v2.

Untuk memantau kapasitas klaster DB Aurora Serverless v2 Anda

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Pilih Metrik. Semua metrik yang tersedia muncul sebagai kartu di konsol, yang dikelompokkan berdasarkan nama layanan.
3. Pilih RDS.
4. (Opsional) Gunakan kotak Pencarian untuk menemukan metrik yang sangat penting untuk Aurora Serverless v2: `ServerlessDatabaseCapacity`, `ACUUtilization`, `CPUUtilization`, dan `FreeableMemory`.

Kami menyarankan Anda menyiapkan CloudWatch dasbor untuk memantau kapasitas cluster Aurora Serverless v2 DB Anda menggunakan metrik terkait kapasitas. Untuk mempelajari caranya, lihat Membangun dasbor dengan CloudWatch.

Untuk mempelajari lebih lanjut tentang menggunakan Amazon CloudWatch dengan Amazon Aurora, lihat. Menerbitkan log Amazon Aurora MySQL ke Amazon Logs CloudWatch

Performa dan penskalaan untuk Aurora Serverless v2

Prosedur dan contoh berikut menunjukkan bagaimana Anda dapat mengatur rentang kapasitas untuk klaster Aurora Serverless v2 dan instans DB terkait. Anda juga dapat menggunakan prosedur berikut untuk memantau seberapa sibuk instans DB Anda. Kemudian, Anda dapat menggunakan temuan Anda untuk menentukan apakah Anda perlu menambah atau mengurangi rentang kapasitas.

Sebelum Anda menggunakan prosedur ini, pastikan Anda memahami cara kerja penskalaan Aurora Serverless v2. Mekanisme penskalaannya berbeda dari Aurora Serverless v1. Untuk detailnya, lihat Penskalaan Aurora Serverless v2.

Daftar Isi

- Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora
  - Memilih pengaturan kapasitas Aurora Serverless v2 minimum untuk klaster
  - Memilih pengaturan kapasitas Aurora Serverless v2 maksimum untuk klaster
  - Contoh: Ubah rentang kapasitas Aurora Serverless v2 klaster untuk Aurora MySQL
  - Contoh: Ubah rentang kapasitas Aurora Serverless v2 untuk klaster Aurora PostgreSQL
- Menggunakan grup parameter untuk Aurora Serverless v2
  - Nilai parameter default
  - Koneksi maksimum untuk Aurora Serverless v2
  - Parameter yang disesuaikan Aurora saat Aurora Serverless v2 menaikkan dan menurunkan skalanya
  - Parameter yang dihitung Aurora berdasarkan kapasitas maksimum Aurora Serverless v2
- Menghindari out-of-memory kesalahan
- CloudWatch Metrik Amazon penting untuk Aurora Serverless v2
  - Bagaimana Aurora Serverless v2 metrik berlaku untuk tagihan Anda AWS
  - Contoh CloudWatch perintah untuk Aurora Serverless v2 metrik
- Memantau performa Aurora Serverless v2 dengan Wawasan Performa
- Memecahkan masalah kapasitas Aurora Serverless v2

Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora

Dengan instans DB Aurora Serverless v2, Anda mengatur rentang kapasitas yang berlaku untuk semua instans DB di klaster DB Anda pada saat yang sama saat Anda menambahkan instans DB

Aurora Serverless v2 pertama ke klaster DB. Untuk prosedur dalam melakukannya, lihat Mengatur rentang kapasitas Aurora Serverless v2 untuk sebuah klaster.

Anda juga dapat mengubah rentang kapasitas untuk klaster yang ada. Bagian berikut membahas secara lebih mendetail tentang cara memilih nilai minimum dan maksimum yang sesuai dan apa yang terjadi ketika Anda membuat perubahan pada rentang kapasitas. Misalnya, mengubah rentang kapasitas dapat memodifikasi nilai default dari beberapa parameter konfigurasi. Untuk menerapkan semua perubahan parameter, setiap instans DB Aurora Serverless v2 mungkin harus di-boot ulang.

Topik

- Memilih pengaturan kapasitas Aurora Serverless v2 minimum untuk klaster
- Memilih pengaturan kapasitas Aurora Serverless v2 maksimum untuk klaster
- Contoh: Ubah rentang kapasitas Aurora Serverless v2 klaster untuk Aurora MySQL
- Contoh: Ubah rentang kapasitas Aurora Serverless v2 untuk klaster Aurora PostgreSQL

Memilih pengaturan kapasitas Aurora Serverless v2 minimum untuk klaster

Pengguna cenderung selalu memilih 0,5 untuk pengaturan kapasitas Aurora Serverless v2 minimum. Nilai tersebut memungkinkan instans DB menurunkan skala semaksimal mungkin saat instans DB ini sepenuhnya idle. Namun, tergantung pada cara Anda menggunakan klaster tersebut dan pengaturan lain yang Anda konfigurasikan, nilai yang berbeda mungkin dapat menjadi paling efektif. Pertimbangkan faktor-faktor berikut saat memilih pengaturan kapasitas minimum:

- Tingkat penskalaan untuk instans DB Aurora Serverless v2 tergantung pada kapasitasnya saat ini. Semakin tinggi kapasitas saat ini, semakin cepat instans DB dapat dinaikkan skalanya. Jika Anda memerlukan instans DB menaikkan skalanya dengan cepat ke kapasitas yang sangat tinggi, pertimbangkan untuk mengatur kapasitas minimum ke sebuah nilai sehingga tingkat penskalaannya memenuhi kebutuhan Anda.
- Jika Anda biasanya memodifikasi kelas instans DB Anda untuk mengantisipasi beban kerja yang sangat tinggi atau rendah, Anda dapat menggunakan pengalaman ini untuk membuat perkiraan kasar tentang rentang kapasitas Aurora Serverless v2 yang setara. Untuk menentukan ukuran memori yang akan digunakan pada saat lalu lintas rendah, baca Spesifikasi perangkat keras kelas instans DB untuk Aurora.

  Misalnya, anggaplah Anda menggunakan kelas instans DB db.r6g.xlarge ketika klaster Anda memiliki beban kerja yang rendah. Kelas instans DB tersebut memiliki memori 32 GiB. Dengan demikian, Anda dapat menentukan pengaturan unit kapasitas Aurora minimum (ACU) sebanyak

16 untuk menyiapkan instans DB Aurora Serverless v2 yang dapat menurunkan skalanya hingga mencapai kapasitas yang kira-kira sama. Itu karena setiap ACU setara dengan sekitar 2 GiB memori. Anda dapat menentukan nilai yang agak lebih rendah agar instans DB menurunkan skalanya lebih jauh jika instans DB db.r6g.xlarge Anda terkadang kurang dimanfaatkan.

- Jika aplikasi Anda beroperasi paling efisien ketika instans DB memiliki sejumlah data dalam cache buffer, pertimbangkan untuk menentukan pengaturan ACU minimum sehingga memorinya cukup besar untuk menampung data yang sering diakses. Jika tidak, beberapa data akan dikeluarkan dari cache buffer ketika instans DB Aurora Serverless v2 menurunkan skalanya ke ukuran memori yang lebih rendah. Kemudian, ketika instans DB menaikkan skalanya kembali, informasi dari data akan dibacakan kembali ke cache buffer dari waktu ke waktu. Jika jumlah I/O untuk memasukkan data kembali ke cache buffer cukup besar, mungkin lebih efektif untuk memilih nilai ACU minimum yang lebih tinggi.
- Jika instans DB Aurora Serverless v2 Anda lebih sering berjalan pada kapasitas tertentu, pertimbangkan untuk menentukan pengaturan kapasitas minimum yang lebih rendah dari acuan dasar tersebut, tetapi tidak terlalu rendah. Instans DB Aurora Serverless v2 dapat sangat efektif memperkirakan berapa banyak dan seberapa cepat skala harus dinaikkan ketika kapasitas saat ini tidak secara drastis lebih rendah dari kapasitas yang dibutuhkan.
- Jika beban kerja terprovisi Anda memiliki persyaratan memori yang terlalu tinggi untuk kelas instans DB kecil seperti T3 atau T4g, pilih pengaturan ACU minimum yang menyediakan memori yang sebanding dengan instans DB R5 atau R6g.

  Secara khusus, kami merekomendasikan kapasitas minimum berikut untuk digunakan dengan fitur yang ditentukan (rekomendasi ini dapat berubah):

  - Wawasan Performa – 2 ACU
  - Basis data global Aurora – 8 ACU (hanya berlaku untuk Wilayah AWS primer)
- Dalam beberapa kasus, klaster Anda mungkin berisi instans DB Aurora Serverless v2 pembaca yang diskalakan secara independen dari penulis. Jika demikian, pilih pengaturan kapasitas minimum yang cukup tinggi sehingga ketika instans DB penulis sibuk dengan beban kerja sarat tulis, instans DB pembaca dapat menerapkan perubahan dari penulis tanpa tertinggal. Jika Anda mengamati lag replika pada pembaca yang berada di tingkatan promosi 2–15, pertimbangkan untuk meningkatkan pengaturan kapasitas minimum untuk klaster Anda. Untuk detail tentang cara memilih apakah instans DB pembaca akan diskalakan bersama dengan penulis atau terpisah, lihat Memilih tingkat promosi untuk pembaca Aurora Serverless v2.
- Jika Anda memiliki cluster DB dengan instans DB Aurora Serverless v2 pembaca, pembaca tidak menskalakan bersama dengan instans DB penulis ketika tingkat promosi pembaca bukan 0 atau

1. Dalam hal ini, pengaturan kapasitas minimum yang rendah dapat mengakibatkan lag replikasi yang berlebihan. Hal ini karena pembaca mungkin tidak memiliki kapasitas yang cukup untuk menerapkan perubahan dari penulis ketika basis data sibuk. Kami menyarankan Anda mengatur kapasitas minimum ke nilai yang mewakili jumlah memori dan CPU yang sebanding dengan instans DB penulis.

- Nilai parameter `max_connections` untuk instans DB Aurora Serverless v2 didasarkan pada ukuran memori yang berasal dari ACU maksimum. Namun, ketika Anda menentukan kapasitas minimum 0,5 ACU pada instans DB yang kompatibel dengan PostgreSQL, nilai maksimum `max_connections` dibatasi pada 2.000.

  Jika Anda bermaksud menggunakan klaster Aurora PostgreSQL untuk beban kerja koneksi tinggi, pertimbangkan untuk menggunakan pengaturan ACU minimum 1 atau lebih tinggi. Untuk detail tentang cara Aurora Serverless v2 menangani parameter konfigurasi `max_connections`, lihat Koneksi maksimum untuk Aurora Serverless v2.
- Waktu yang dibutuhkan instans DB Aurora Serverless v2 untuk menskalakan dari kapasitas minimumnya ke kapasitas maksimumnya akan tergantung pada perbedaan antara nilai ACU minimum dan maksimumnya. Ketika kapasitas instans DB saat ini besar, Aurora Serverless v2 akan dinaikkan skalanya dalam inkremen yang lebih besar daripada ketika instans DB dimulai dari kapasitas kecil. Jadi, jika Anda menentukan kapasitas maksimum yang relatif besar dan instans DB menghabiskan sebagian besar waktunya mendekati kapasitas tersebut, pertimbangkan untuk meningkatkan pengaturan ACU minimum. Dengan begitu, instans DB idle dapat menaikkan skalanya kembali ke kapasitas maksimum dengan lebih cepat.

Memilih pengaturan kapasitas Aurora Serverless v2 maksimum untuk klaster

Pengguna cenderung selalu memilih nilai tinggi tertentu untuk pengaturan kapasitas Aurora Serverless v2 maksimum. Kapasitas maksimum yang besar memungkinkan instans DB menaikkan skala secara maksimal saat menjalankan beban kerja intensif. Nilai rendah menghindari kemungkinan biaya tak terduga. Bergantung pada cara Anda menggunakan klaster tersebut dan pengaturan lain yang Anda konfigurasikan, nilai yang paling efektif mungkin lebih tinggi atau lebih rendah dari yang Anda kira. Pertimbangkan faktor-faktor berikut saat memilih pengaturan kapasitas maksimum:

- Kapasitas maksimum harus minimal setinggi kapasitas minimum. Anda dapat mengatur kapasitas minimum dan maksimum agar identik. Namun, dalam hal ini kapasitas tidak pernah naik atau turun. Dengan demikian, menggunakan nilai yang identik untuk kapasitas minimum dan maksimum tidak tepat untuk situasi di luar pengujian.

- Kapasitas maksimum harus lebih tinggi dari 0,5 ACU. Anda dapat mengatur kapasitas minimum dan maksimum agar sama dalam banyak kasus. Namun, Anda tidak dapat menentukan 0,5 untuk minimum dan maksimum. Gunakan nilai 1 atau lebih tinggi untuk kapasitas maksimum.
- Jika Anda biasanya memodifikasi kelas instans DB Anda untuk mengantisipasi beban kerja yang sangat tinggi atau rendah, Anda dapat menggunakan pengalaman ini untuk memperkirakan rentang kapasitas Aurora Serverless v2 yang setara. Untuk menentukan ukuran memori yang akan digunakan pada saat lalu lintas tinggi, baca Spesifikasi perangkat keras kelas instans DB untuk Aurora.

  Misalnya, anggaplah Anda menggunakan kelas instans DB db.r6g.4xlarge ketika klaster Anda memiliki beban kerja yang tinggi. Kelas instans DB tersebut memiliki memori 128 GiB. Dengan demikian, Anda dapat menentukan pengaturan ACU maksimum sebanyak 64 untuk menyiapkan instans DB Aurora Serverless v2 yang dapat menaikkan skalanya hingga mencapai kapasitas yang kira-kira sama. Itu karena setiap ACU setara dengan sekitar 2 GiB memori. Anda dapat menentukan nilai yang agak lebih tinggi agar instans DB menaikkan skalanya lebih jauh jika instans DB db.r6g.4xlarge Anda terkadang tidak memiliki kapasitas yang cukup untuk menangani beban kerja secara efektif.
- Jika Anda memiliki batas anggaran pada penggunaan basis data Anda, pilih nilai yang tetap dalam batas tersebut bahkan jika semua instans DB Aurora Serverless v2 Anda berjalan pada kapasitas maksimum sepanjang waktu. Ingatlah bahwa ketika Anda memiliki n instans DB Aurora Serverless v2 di klaster Anda, kapasitas Aurora Serverless v2 maksimum teoretis yang dapat dikonsumsi klaster setiap saat adalah n kali pengaturan ACU maksimum untuk klaster. (Jumlah sebenarnya yang dikonsumsi mungkin lebih sedikit, misalnya jika beberapa pembaca diskalakan secara terpisah dari penulis.)
- Jika Anda menggunakan instans DB Aurora Serverless v2 pembaca untuk mengalihkan beberapa beban kerja hanya baca dari instans DB penulis, Anda mungkin dapat memilih pengaturan kapasitas maksimum yang lebih rendah. Anda melakukannya agar setiap instans DB pembaca tidak perlu diskalakan sama tingginya saat klaster hanya berisi satu instans DB.
- Misalkan Anda ingin mencegah penggunaan yang berlebihan karena parameter basis data yang salah dikonfigurasi atau kueri yang tidak efisien dalam aplikasi Anda. Dalam hal ini, Anda dapat menghindari penggunaan berlebihan yang tidak disengaja dengan memilih pengaturan kapasitas maksimum yang lebih rendah dari pengaturan tertinggi absolut yang dapat Anda tetapkan.
- Jika lonjakan karena aktivitas pengguna sebenarnya jarang tetapi dapat terjadi, Anda dapat mempertimbangkan situasi tersebut saat memilih pengaturan kapasitas maksimum. Jika prioritasnya adalah agar aplikasi tetap berjalan dengan performa dan skalabilitas penuh, Anda dapat menentukan pengaturan kapasitas maksimum yang lebih tinggi daripada yang Anda amati

dalam penggunaan normal. Jika Anda tidak memiliki masalah dengan aplikasi yang berjalan dengan throughput yang berkurang selama lonjakan aktivitas yang sangat ekstrem, Anda dapat memilih pengaturan kapasitas maksimum yang sedikit lebih rendah. Pastikan Anda memilih pengaturan yang masih memiliki cukup memori dan sumber daya CPU untuk menjaga aplikasi tetap berjalan.

- Jika Anda mengaktifkan pengaturan di klaster Anda yang meningkatkan penggunaan memori untuk setiap instans DB, pertimbangkan memori tersebut saat memutuskan nilai ACU maksimum. Pengaturan tersebut mencakup pengaturan untuk Wawasan Performa, kueri paralel Aurora MySQL, skema performa Aurora MySQL, dan replikasi log biner Aurora MySQL. Pastikan bahwa nilai ACU maksimum memungkinkan instans DB Aurora Serverless v2 menaikkan skalanya secara memadai untuk menangani beban kerja saat fitur tersebut digunakan. Untuk informasi tentang pemecahan masalah yang disebabkan oleh kombinasi pengaturan ACU maksimum rendah dan fitur Aurora yang memerlukan overhead memori, lihat Menghindari out-of-memory kesalahan.

Contoh: Ubah rentang kapasitas Aurora Serverless v2 klaster untuk Aurora MySQL

AWS CLI Contoh berikut menunjukkan cara memperbarui rentang ACU untuk instans Aurora Serverless v2 DB di cluster Aurora MySQL yang ada. Awalnya, rentang kapasitas untuk klaster adalah 8–32 ACU.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

Instans DB idle dan diturunkan skalanya menjadi 8 ACU. Pengaturan terkait kapasitas berikut berlaku untuk instans DB pada saat ini. Untuk merepresentasikan ukuran pool buffer dalam unit yang mudah dibaca, kita membaginya dengan 2 pangkat 30, sehingga menghasilkan pengukuran dalam gibibyte (GiB). Itu karena pengukuran terkait memori untuk Aurora menggunakan unit berdasarkan pangkat 2, bukan pangkat 10.

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              3000 |
```

```
+-------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|                9294577664 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
+-----------+
|   8.65625 |
+-----------+
1 row in set (0.00 sec)
```

Selanjutnya, kita mengubah rentang kapasitas untuk klaster. Setelah perintah `modify-db-cluster` selesai, rentang ACU untuk klaster adalah 12,5–80.

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

Mengubah rentang kapasitas menyebabkan perubahan pada nilai default dari beberapa parameter konfigurasi. Aurora dapat segera menerapkan beberapa nilai default baru tersebut. Namun, beberapa perubahan parameter hanya berlaku setelah boot ulang. Status `pending-reboot` menunjukkan bahwa boot ulang diperlukan untuk menerapkan beberapa perubahan parameter.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
```

```
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

Pada tahap ini, klaster menjadi idle dan instans DB `serverless-v2-instance-1` mengonsumsi 12,5 ACU. Parameter`innodb_buffer_pool_size` sudah disesuaikan berdasarkan kapasitas instans DB saat ini. Parameter `max_connections` masih mencerminkan nilai dari kapasitas maksimum sebelumnya. Jika nilai tersebut direset, instans DB harus di-boot ulang.

> **Note**
>
> Jika Anda mengatur parameter `max_connections` secara langsung dalam grup parameter DB kustom, boot ulang tidak diperlukan.

```
mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              3000 |
+-------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|               15572402176 |
+---------------------------+
1 row in set (0.00 sec)

mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+---------------+
| gibibytes     |
+---------------+
| 14.5029296875 |
+---------------+
```

```
1 row in set (0.00 sec)
```

Sekarang kita mem-boot ulang instans DB dan menunggu hingga instans tersebut tersedia lagi.

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

Status `pending-reboot` dihapus. Nilai `in-sync` mengonfirmasi bahwa Aurora telah menerapkan semua perubahan parameter yang tertunda.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

Parameter `innodb_buffer_pool_size` telah meningkat ke ukuran akhir untuk instans DB idle. Parameter `max_connections` telah meningkat untuk mencerminkan nilai yang berasal dari nilai ACU maksimum. Rumus yang digunakan Aurora untuk `max_connections` menyebabkan peningkatan 1.000 ketika ukuran memori berlipat ganda.

```
mysql> select @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|               16139681792 |
+---------------------------+
1 row in set (0.00 sec)
```

```
mysql> select @@innodb_buffer_pool_size / pow(2,30) as gibibytes;
+-----------+
| gibibytes |
+-----------+
|  15.03125 |
+-----------+
1 row in set (0.00 sec)

mysql> select @@max_connections;
+-------------------+
| @@max_connections |
+-------------------+
|              4000 |
+-------------------+
1 row in set (0.00 sec)
```

Kita mengatur rentang kapasitas menjadi 0,5–128 ACU, dan mem-boot ulang instans DB. Sekarang instans DB idle memiliki ukuran cache buffer yang kurang dari 1 GiB, jadi kita mengukurnya dalam mebibyte (MiB). Nilai `max_connections` 5000 berasal dari ukuran memori pada pengaturan kapasitas maksimum.

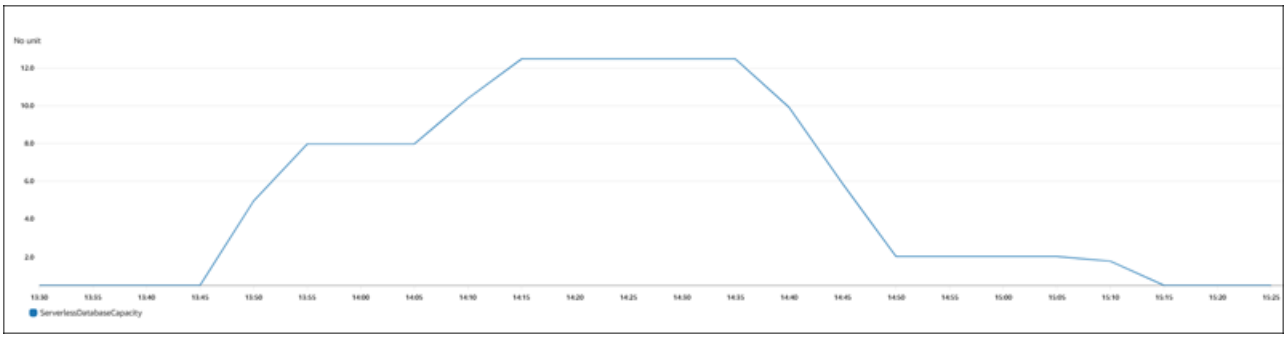
```
mysql> select @@innodb_buffer_pool_size / pow(2,20) as mebibytes, @@max_connections;
+-----------+-------------------+
| mebibytes | @@max_connections |
+-----------+-------------------+
|       672 |              5000 |
+-----------+-------------------+
1 row in set (0.00 sec)
```

Contoh: Ubah rentang kapasitas Aurora Serverless v2 untuk klaster Aurora PostgreSQL

Contoh CLI berikut menunjukkan cara memperbarui rentang ACU untuk instans DB Aurora Serverless v2 di klaster Aurora PostgreSQL yang sudah ada.

1. Rentang kapasitas untuk klaster dimulai pada 0,5–1 ACU.
2. Ubah rentang kapasitas menjadi 8–32 ACU.
3. Ubah rentang kapasitas menjadi 12,5–80 ACU.
4. Ubah rentang kapasitas menjadi 0,5–128 ACU.
5. Kembalikan kapasitas ke rentang awal 0,5–1 ACU.

Gambar berikut menunjukkan perubahan kapasitas di Amazon CloudWatch.



Instans DB idle dan diturunkan skalanya menjadi 0,5 ACU. Pengaturan terkait kapasitas berikut berlaku untuk instans DB pada saat ini.

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

Selanjutnya, kita mengubah rentang kapasitas untuk klaster. Setelah perintah `modify-db-cluster` selesai, rentang ACU untuk klaster adalah 8.0–32.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
    "MinCapacity": 8.0,
    "MaxCapacity": 32.0
}
```

Mengubah rentang kapasitas menyebabkan perubahan pada nilai default dari beberapa parameter konfigurasi. Aurora dapat segera menerapkan beberapa nilai default baru tersebut. Namun, beberapa perubahan parameter hanya berlaku setelah boot ulang. Status `pending-reboot` menunjukkan bahwa Anda harus melakukan boot ulang untuk menerapkan beberapa perubahan parameter.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
```

```
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "pending-reboot"
        }
    ]
}
```

Pada tahap ini, klaster menjadi idle dan instans DB `serverless-v2-instance-1` mengonsumsi 8,0 ACU. Parameter`shared_buffers` sudah disesuaikan berdasarkan kapasitas instans DB saat ini. Parameter `max_connections` masih mencerminkan nilai dari kapasitas maksimum sebelumnya. Jika nilai tersebut direset, instans DB harus di-boot ulang.

> **Note**
>
> Jika Anda mengatur parameter `max_connections` secara langsung dalam grup parameter DB kustom, boot ulang tidak diperlukan.

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 1425408
(1 row)
```

Kita mem-boot ulang instans DB dan menunggu hingga instans tersebut tersedia lagi.

```
aws rds reboot-db-instance --db-instance-identifier serverless-v2-instance-1
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBInstanceStatus": "rebooting"
```

```
}

aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
```

Sekarang setelah instans DB di-boot ulang, status `pending-reboot` dihapus. Nilai `in-sync` mengonfirmasi bahwa Aurora telah menerapkan semua perubahan parameter yang tertunda.

```
aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query '*[].{DBClusterMembers:DBClusterMembers[*].
{DBInstanceIdentifier:DBInstanceIdentifier,DBClusterParameterGroupStatus:DBClusterParameterGrou
[0]'
{
    "DBClusterMembers": [
        {
            "DBInstanceIdentifier": "serverless-v2-instance-1",
            "DBClusterParameterGroupStatus": "in-sync"
        }
    ]
}
```

Setelah boot ulang, `max_connections` menunjukkan nilai dari kapasitas maksimum baru.

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 1425408
(1 row)
```

Selanjutnya, kita mengubah rentang kapasitas untuk klaster menjadi 12,5–80 ACU.

```
aws rds modify-db-cluster --db-cluster-identifier serverless-v2-cluster \
  --serverless-v2-scaling-configuration MinCapacity=12.5,MaxCapacity=80

aws rds describe-db-clusters --db-cluster-identifier serverless-v2-cluster \
  --query 'DBClusters[*].ServerlessV2ScalingConfiguration|[0]'
{
```

```
    "MinCapacity": 12.5,
    "MaxCapacity": 80.0
}
```

Pada tahap ini, klaster menjadi idle dan instans DB `serverless-v2-instance-1` mengonsumsi 12,5 ACU. Parameter`shared_buffers` sudah disesuaikan berdasarkan kapasitas instans DB saat ini. Nilai `max_connections` masih 5000.

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 2211840
(1 row)
```

Kita mem-boot ulang lagi, tetapi nilai parameternya tetap sama. Ini karena `max_connections` memiliki nilai maksimum 5000 untuk klaster DB Aurora Serverless v2 yang menjalankan Aurora PostgreSQL.

```
postgres=> show max_connections;
 max_connections
-----------------
 5000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 2211840
(1 row)
```

Sekarang kita mengatur kisaran kapasitas dari 0,5 menjadi 128 ACU. Klaster DB diturunkan skalanya menjadi 10 ACU, lalu menjadi 2. Kita mem-boot ulang instans DB.

```
postgres=> show max_connections;
 max_connections
```

```
-----------------
 2000
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

Nilai `max_connections` untuk instans DB Aurora Serverless v2 didasarkan pada ukuran memori yang berasal dari ACU maksimum. Namun, ketika Anda menentukan kapasitas minimum 0,5 ACU pada instans DB yang kompatibel dengan PostgreSQL, nilai maksimum `max_connections` dibatasi pada 2.000.

Sekarang kita mengembalikan kapasitas ke rentang awal 0,5–1 ACU dan mem-boot ulang instans DB. Parameter `max_connections` telah kembali ke nilai aslinya.

```
postgres=> show max_connections;
 max_connections
-----------------
 189
(1 row)

postgres=> show shared_buffers;
 shared_buffers
----------------
 16384
(1 row)
```

Menggunakan grup parameter untuk Aurora Serverless v2

Saat Anda membuat klaster DB Aurora Serverless v2, Anda memilih mesin Aurora DB tertentu dan grup parameter klaster DB terkait. Jika Anda tidak memahami cara Aurora menggunakan grup parameter untuk menerapkan pengaturan konfigurasi secara konsisten di seluruh klaster, lihat Bekerja dengan grup parameter. Semua prosedur tersebut untuk membuat, memodifikasi, menerapkan, dan tindakan lain untuk grup parameter akan berlaku untuk Aurora Serverless v2.

Fitur grup parameter beroperasi sama antara klaster terprovisi dan klaster yang berisi instans DB Aurora Serverless v2:

- Nilai parameter default untuk semua instans DB di klaster ditentukan oleh grup parameter klaster.
- Anda dapat mengganti beberapa parameter untuk instans DB tertentu dengan menentukan grup parameter DB kustom untuk instans DB tersebut. Anda dapat melakukannya selama debugging atau penyesuaian performa untuk instans DB tertentu. Misalnya, anggaplah bahwa Anda memiliki klaster yang berisi beberapa instans DB Aurora Serverless v2 dan beberapa instans DB terprovisi. Dalam kasus ini, Anda dapat menentukan beberapa parameter berbeda untuk instans DB terprovisi dengan menggunakan grup parameter DB kustom.
- Untuk Aurora Serverless v2, Anda dapat menggunakan semua parameter yang memiliki nilai `provisioned` dalam atribut `SupportedEngineModes` di grup parameter. Di Aurora Serverless v1, Anda hanya dapat menggunakan subset parameter yang memiliki `serverless` dalam atribut `SupportedEngineModes`.

Topik

- Nilai parameter default
- Koneksi maksimum untuk Aurora Serverless v2
- Parameter yang disesuaikan Aurora saat Aurora Serverless v2 menaikkan dan menurunkan skalanya
- Parameter yang dihitung Aurora berdasarkan kapasitas maksimum Aurora Serverless v2

Nilai parameter default

Perbedaan penting antara instans DB terprovisi dan instans DB Aurora Serverless v2 adalah bahwa Aurora mengganti nilai parameter kustom apa pun untuk parameter tertentu yang terkait dengan kapasitas instans DB. Nilai parameter kustom masih berlaku untuk instans DB terprovisi di klaster Anda. Untuk detail selengkapnya tentang bagaimana instans DB Aurora Serverless v2 menafsirkan parameter dari grup parameter Aurora, lihat Parameter konfigurasi untuk klaster Aurora. Untuk parameter spesifik yang diganti oleh Aurora Serverless v2, lihat Parameter yang disesuaikan Aurora saat Aurora Serverless v2 menaikkan dan menurunkan skalanya dan Parameter yang dihitung Aurora berdasarkan kapasitas maksimum Aurora Serverless v2.

Anda bisa mendapatkan daftar nilai default untuk grup parameter default untuk berbagai mesin Aurora DB dengan menggunakan perintah describe-db-cluster-parametersCLI dan menanyakan file. Wilayah AWS Berikut ini adalah nilai-nilai yang dapat Anda gunakan untuk opsi `--db-parameter-group-family` dan `-db-parameter-group-name` untuk versi mesin yang kompatibel dengan Aurora Serverless v2.

Mesin dan versi basis data	Kelompok grup parameter	Nama grup parameter default
Aurora MySQL versi 3	`aurora-mysql8.0`	`default.aurora-mysql8.0`
Aurora PostgreSQL versi 13.x	`aurora-postgresql13`	`default.aurora-postgresql13`
Aurora PostgreSQL versi 14.x	`aurora-postgresql14`	`default.aurora-postgresql14`
Aurora PostgreSQL versi 15.x	`aurora-postgresql15`	`default.aurora-postgresql15`
Aurora PostgreSQL versi 16.x	`aurora-postgresql16`	`default.aurora-postgresql16`

Contoh berikut mendapatkan daftar parameter dari grup klaster DB default untuk Aurora MySQL versi 3 dan Aurora PostgreSQL 13. Ini adalah versi Aurora MySQL dan Aurora PostgreSQL yang Anda gunakan dengan Aurora Serverless v2.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-mysql8.0 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
  --output text

aws rds describe-db-cluster-parameters \
  --db-cluster-parameter-group-name default.aurora-postgresql13 \
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' \
  --output text
```

Untuk Windows:

```
aws rds describe-db-cluster-parameters ^
```

```
  --db-cluster-parameter-group-name default.aurora-mysql8.0 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text

aws rds describe-db-cluster-parameters ^
  --db-cluster-parameter-group-name default.aurora-postgresql13 ^
  --query 'Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} |
    [?contains(SupportedEngineModes, `provisioned`) == `true`] | [*].[ParameterName]' ^
  --output text
```

Koneksi maksimum untuk Aurora Serverless v2

Untuk Aurora MySQL dan Aurora PostgreSQL, instans DB Aurora Serverless v2 menjaga parameter `max_connections` tetap konstan sehingga koneksi tidak terputus saat instans DB diturunkan skalanya. Nilai default untuk parameter ini berasal dari rumus berdasarkan ukuran memori instans DB. Untuk detail tentang rumus dan nilai default untuk kelas instans DB terprovisi, lihat Koneksi maksimum ke instans DB Aurora MySQL dan Koneksi maksimum ke instans DB Aurora PostgreSQL.

Ketika Aurora Serverless v2 mengevaluasi rumus tersebut, layanan ini menggunakan ukuran memori berdasarkan unit kapasitas Aurora (ACU) maksimum untuk instans DB, bukan nilai ACU saat ini. Jika Anda mengubah nilai default, sebaiknya gunakan variasi rumus alih-alih menentukan nilai konstan. Dengan begitu, Aurora Serverless v2 bisa menggunakan pengaturan yang sesuai berdasarkan kapasitas maksimum.

Saat Anda mengubah kapasitas maksimum klaster DB Aurora Serverless v2, Anda harus mem-boot ulang instans DB Aurora Serverless v2 untuk memperbarui nilai `max_connections`. Ini karena `max_connections` merupakan parameter statis untuk Aurora Serverless v2.

Tabel berikut menunjukkan nilai default untuk `max_connections` bagi Aurora Serverless v2 berdasarkan nilai ACU maksimum.

ACU maksimum	Koneksi maksimum default pada Aurora MySQL	Koneksi maksimum default pada Aurora PostgreSQL
1	90	189
4	135	823

ACU maksimum	Koneksi maksimum default pada Aurora MySQL	Koneksi maksimum default pada Aurora PostgreSQL
8	1.000	1,669
16	2.000	3,360
32	3.000	5.000
64	4.000	5.000
128	5.000	5.000

> **Note**
>
> Nilai `max_connections` untuk instans DB Aurora Serverless v2 didasarkan pada ukuran memori yang berasal dari ACU maksimum. Namun, ketika Anda menentukan kapasitas minimum 0,5 ACU pada instans DB yang kompatibel dengan PostgreSQL, nilai maksimum `max_connections` dibatasi pada 2.000.

Untuk contoh spesifik yang menunjukkan perubahan `max_connections` dengan nilai ACU maksimum, lihat Contoh: Ubah rentang kapasitas Aurora Serverless v2 klaster untuk Aurora MySQL dan Contoh: Ubah rentang kapasitas Aurora Serverless v2 untuk klaster Aurora PostgreSQL.

Parameter yang disesuaikan Aurora saat Aurora Serverless v2 menaikkan dan menurunkan skalanya

Selama penskalaan otomatis, Aurora Serverless v2 harus dapat mengubah parameter agar setiap instans DB beroperasi dengan paling efektif untuk peningkatan atau penurunan kapasitas. Dengan demikian, Anda tidak dapat mengganti beberapa parameter yang terkait dengan kapasitas. Untuk beberapa parameter yang dapat Anda ganti, hindari melakukan hard-coding nilai tetap. Pertimbangan berikut berlaku untuk pengaturan ini yang terkait dengan kapasitas.

Untuk Aurora MySQL, Aurora Serverless v2 mengubah ukuran beberapa parameter secara dinamis selama penskalaan. Untuk parameter berikut, Aurora Serverless v2 tidak menggunakan nilai parameter kustom apa pun yang Anda tentukan:

- innodb_buffer_pool_size
- innodb_purge_threads
- table_definition_cache
- table_open_cache

Untuk Aurora PostgreSQL, Aurora Serverless v2 mengubah ukuran parameter berikut secara dinamis selama penskalaan. Untuk parameter berikut, Aurora Serverless v2 tidak menggunakan nilai parameter kustom apa pun yang Anda tentukan:

- shared_buffers

Untuk semua parameter selain yang tercantum di sini, instans Aurora Serverless v2 DB beroperasi sama dengan instans DB terprovisi. Nilai parameter default diwarisi dari grup parameter klaster. Anda dapat memodifikasi nilai default untuk seluruh klaster dengan menggunakan grup parameter klaster kustom. Atau Anda dapat memodifikasi nilai default untuk instans DB tertentu dengan menggunakan grup parameter DB kustom. Parameter dinamis akan segera diperbarui. Perubahan parameter statis hanya berlaku setelah Anda mem-boot ulang instans DB.

Parameter yang dihitung Aurora berdasarkan kapasitas maksimum Aurora Serverless v2

Untuk parameter berikut, Aurora PostgreSQL menggunakan nilai default yang berasal dari ukuran memori berdasarkan pengaturan ACU maksimum, sama seperti dengan max_connections:

- autovacuum_max_workers
- autovacuum_vacuum_cost_limit
- autovacuum_work_mem
- effective_cache_size
- maintenance_work_mem

Menghindari out-of-memory kesalahan

Jika salah satu instans DB Aurora Serverless v2 Anda secara konsisten mencapai batas kapasitas maksimumnya, Aurora akan menunjukkan kondisi ini dengan mengatur instans DB ke status

`incompatible-parameters`. Saat instans DB memiliki status `incompatible-parameters`, beberapa operasi akan diblokir. Misalnya, Anda tidak dapat meningkatkan versi mesin.

Biasanya, instans DB Anda masuk ke status ini ketika sering restart karena out-of-memory kesalahan. Aurora mencatat sebuah peristiwa ketika jenis pengaktifan ulang ini terjadi. Anda dapat melihat peristiwa ini dengan mengikuti prosedur dalam Melihat peristiwa Amazon RDS. Penggunaan memori yang luar biasa tinggi dapat terjadi karena overhead yang timbul saat mengaktifkan pengaturan seperti Wawasan Performa dan autentikasi IAM. Hal ini juga bisa berasal dari beban kerja yang berat pada instans DB Anda atau pengelolaan metadata yang terkait dengan sejumlah besar objek skema.

Jika tekanan memori menjadi lebih rendah sehingga instans DB sangat sering tidak mencapai kapasitas maksimumnya, Aurora akan secara otomatis mengubah status instans DB kembali ke `available`.

Untuk pulih dari kondisi ini, Anda dapat mengambil beberapa atau semua tindakan berikut:

- Tingkatkan batas bawah kapasitas untuk instans DB Aurora Serverless v2 dengan mengubah nilai unit kapasitas Aurora (ACU) minimum untuk klaster. Tindakan ini menghindari masalah saat basis data idle menurunkan skalanya ke kapasitas dengan memori lebih sedikit daripada yang dibutuhkan untuk fitur yang diaktifkan di klaster Anda. Setelah mengubah pengaturan ACU untuk klaster, boot ulang instans DB Aurora Serverless v2. Tindakan ini mengevaluasi apakah Aurora dapat mereset status kembali ke `available`.
- Tingkatkan batas atas kapasitas instans DB Aurora Serverless v2 dengan mengubah nilai ACU maksimum untuk klaster. Tindakan ini menghindari masalah saat basis data yang sibuk tidak dapat menaikkan skalanya ke kapasitas dengan memori yang cukup untuk fitur yang diaktifkan di klaster Anda dan beban kerja basis data. Setelah mengubah pengaturan ACU untuk klaster, boot ulang instans DB Aurora Serverless v2. Tindakan ini mengevaluasi apakah Aurora dapat mereset status kembali ke `available`.
- Nonaktifkan pengaturan konfigurasi yang memerlukan overhead memori. Misalnya, Anda memiliki fitur seperti AWS Identity and Access Management (IAM), Performance Insights, atau replikasi log biner Aurora MySQL yang diaktifkan tetapi tidak menggunakannya. Jika demikian, Anda dapat menonaktifkannya. Atau Anda dapat menambah nilai kapasitas minimum dan maksimum untuk klaster untuk memperhitungkan memori yang digunakan oleh fitur-fitur tersebut. Untuk panduan tentang memilih pengaturan kapasitas minimum dan maksimum, lihat Memilih rentang kapasitas Aurora Serverless v2 untuk klaster Aurora.
- Kurangi beban kerja pada instans DB. Misalnya, Anda dapat menambahkan instans DB pembaca ke klaster untuk menyebarkan beban dari kueri hanya baca ke lebih banyak instans DB.

- Sesuaikan kode SQL yang digunakan oleh aplikasi Anda untuk menggunakan lebih sedikit sumber daya. Misalnya, Anda dapat memeriksa rencana kueri Anda, memeriksa log kueri yang lambat, atau menyesuaikan indeks pada tabel Anda. Anda juga dapat melakukan jenis penyesuaian SQL standar lainnya.

CloudWatch Metrik Amazon penting untuk Aurora Serverless v2

Untuk memulai Amazon CloudWatch untuk instans Aurora Serverless v2 DB Anda, lihatMelihat Aurora Serverless v2 log di Amazon CloudWatch. Untuk mempelajari lebih lanjut tentang cara memantau klaster Aurora DB, lihat CloudWatch. Memantau peristiwa log di Amazon CloudWatch

Anda dapat melihat instans Aurora Serverless v2 DB CloudWatch untuk memantau kapasitas yang dikonsumsi oleh setiap instans DB dengan `ServerlessDatabaseCapacity` metrik. Anda juga dapat memantau semua CloudWatch metrik Aurora standar, seperti dan. `DatabaseConnections` `Queries` Untuk daftar lengkap CloudWatch metrik yang dapat Anda pantau untuk Aurora, lihat. CloudWatch Metrik Amazon untuk Amazon Aurora Metrik dibagi menjadi metrik tingkat klaster dan tingkat instans, di Metrik tingkat klaster untuk Amazon Aurora dan Metrik tingkat instans untuk Amazon Aurora.

Metrik CloudWatch tingkat instans berikut ini penting untuk dipantau agar Anda memahami bagaimana instans Aurora Serverless v2 DB Anda meningkat dan turun. Semua metrik ini dihitung setiap detik. Dengan begitu, Anda dapat memantau status instans DB Aurora Serverless v2 Anda saat ini. Anda dapat mengatur alarm untuk memberi tahu Anda jika ada instans DB Aurora Serverless v2 yang mendekati ambang batas untuk metrik yang terkait dengan kapasitas. Anda dapat menentukan apakah pengaturan kapasitas minimum dan maksimum sudah sesuai, atau apakah Anda perlu menyesuaikannya. Anda dapat menentukan di mana upaya Anda harus difokuskan untuk mengoptimalkan efisiensi basis data Anda.

- `ServerlessDatabaseCapacity`. Sebagai metrik tingkat instans, metrik ini melaporkan jumlah ACU yang direpresentasikan oleh kapasitas instans DB saat ini. Sebagai metrik tingkat klaster, metrik ini merepresentasikan rata-rata nilai `ServerlessDatabaseCapacity` milik semua instans DB Aurora Serverless v2 di klaster. Metrik ini adalah satu-satunya metrik tingkat klaster di Aurora Serverless v1. Di Aurora Serverless v2, metrik ini tersedia di tingkat instans DB dan di tingkat klaster.
- `ACUUtilization`. Ini adalah metrik baru di Aurora Serverless v2. Nilai ini direpresentasikan sebagai persentase. Metrik ini dihitung sebagai nilai metrik `ServerlessDatabaseCapacity`

dibagi dengan nilai ACU maksimum klaster DB. Pertimbangkan pedoman berikut untuk menafsirkan metrik ini dan mengambil tindakan:

- Jika metrik ini mendekati nilai `100.0`, instans DB telah dinaikkan skalanya setinggi mungkin. Pertimbangkan untuk meningkatkan pengaturan ACU maksimum untuk klaster. Dengan begitu, instans DB penulis dan pembaca dapat diskalakan ke kapasitas yang lebih tinggi.
- Misalkan beban kerja hanya baca menyebabkan instans DB pembaca mendekati `ACUUtilization` sebesar `100.0`, sedangkan instans DB penulis tidak mendekati kapasitas maksimumnya. Dalam hal ini, pertimbangkan untuk menambahkan instans DB pembaca lainnya ke klaster. Dengan begitu, Anda dapat menyebarkan bagian hanya baca dari beban kerja yang tersebar di lebih banyak instans DB, sehingga mengurangi beban pada setiap instans DB pembaca.
- Misalkan Anda menjalankan aplikasi produksi, sehingga performa dan skalabilitas adalah pertimbangan utamanya. Dalam hal ini, Anda dapat mengatur nilai ACU maksimum untuk klaster ke angka yang tinggi. Tujuan Anda adalah agar metrik `ACUUtilization` selalu berada di bawah `100.0`. Dengan nilai ACU maksimum yang tinggi, Anda dapat yakin bahwa ada cukup ruang jika ada lonjakan tak terduga dalam aktivitas basis data. Anda hanya dikenai biaya untuk kapasitas basis data yang benar-benar dikonsumsi.

- `CPUUtilization`. Metrik ini ditafsirkan secara berbeda di Aurora Serverless v2 daripada di instans DB terprovisi. Untuk Aurora Serverless v2, nilai ini adalah persentase yang dihitung sebagai jumlah CPU yang saat ini digunakan dibagi dengan kapasitas CPU yang tersedia di bawah nilai ACU maksimum klaster DB. Aurora memantau nilai ini secara otomatis dan menaikkan skala instans DB Aurora Serverless v2 Anda ketika instans DB secara konsisten menggunakan proporsi kapasitas CPU-nya yang tinggi.

  Jika metrik ini mendekati nilai `100.0`, instans DB telah mencapai kapasitas CPU maksimumnya. Pertimbangkan untuk meningkatkan pengaturan ACU maksimum untuk klaster. Jika metrik ini mendekati nilai `100.0` pada instans DB pembaca, pertimbangkan untuk menambahkan instans DB pembaca lain ke klaster. Dengan begitu, Anda dapat menyebarkan bagian hanya baca dari beban kerja yang tersebar di lebih banyak instans DB, sehingga mengurangi beban pada setiap instans DB pembaca.
- `FreeableMemory`. Nilai ini merepresentasikan jumlah memori yang tidak terpakai yang tersedia ketika instans DB Aurora Serverless v2 diskalakan ke kapasitas maksimumnya. Untuk setiap ACU yang kapasitasnya saat ini di bawah kapasitas maksimum, nilai ini meningkat sekitar 2 GiB. Dengan demikian, metrik ini tidak mendekati nol sampai instans DB dinaikkan skalanya setinggi mungkin.

Jika metrik ini mendekati nilai `0`, instans DB telah dinaikkan skalanya setinggi mungkin dan mendekati batas memori yang tersedia. Pertimbangkan untuk meningkatkan pengaturan ACU maksimum untuk klaster. Jika metrik ini mendekati nilai `0` pada instans DB pembaca, pertimbangkan untuk menambahkan instans DB pembaca lain ke klaster. Dengan begitu, bagian hanya baca dari beban kerja dapat disebarkan ke lebih banyak instans DB, sehingga mengurangi penggunaan memori pada setiap instans DB pembaca.

- `TempStorageIops`. Jumlah IOPS yang dilakukan pada penyimpanan lokal yang dilampirkan ke instans DB. Ini termasuk IOPS untuk baca dan tulis. Metrik ini merepresentasikan hitungan dan diukur sekali per detik. Ini adalah metrik baru untuk Aurora Serverless v2. Untuk detailnya, lihat Metrik tingkat instans untuk Amazon Aurora.
- `TempStorageThroughput`. Jumlah data yang ditransfer ke dan dari penyimpanan lokal yang terkait dengan instans DB. Metrik ini merepresentasikan byte dan diukur sekali per detik. Ini adalah metrik baru untuk Aurora Serverless v2. Untuk detailnya, lihat Metrik tingkat instans untuk Amazon Aurora.

Biasanya, sebagian besar penskalaan untuk instans DB Aurora Serverless v2 disebabkan oleh penggunaan memori dan aktivitas CPU. Metrik `TempStorageIops` dan `TempStorageThroughput` metrik dapat membantu Anda mendiagnosis kasus yang jarang terjadi saat aktivitas jaringan untuk transfer antara instans DB dan perangkat penyimpanan lokal menimbulkan peningkatan kapasitas yang tidak terduga. Untuk memantau aktivitas jaringan lainnya, Anda dapat menggunakan metrik yang ada ini:

- `NetworkReceiveThroughput`
- `NetworkThroughput`
- `NetworkTransmitThroughput`
- `StorageNetworkReceiveThroughput`
- `StorageNetworkThroughput`
- `StorageNetworkTransmitThroughput`

Anda dapat meminta Aurora mempublikasikan beberapa atau semua log database. CloudWatch Anda memilih log yang akan diterbitkan dengan mengaktifkan parameter konfigurasi seperti `general_log` dan `slow_query_log` di grup parameter klaster yang terkait dengan klaster yang berisi instans DB Aurora Serverless v2 Anda. Saat Anda mematikan parameter konfigurasi

log, memublikasikan log tersebut untuk CloudWatch berhenti. Anda juga dapat menghapus log CloudWatch jika tidak lagi diperlukan.

Bagaimana Aurora Serverless v2 metrik berlaku untuk tagihan Anda AWS

Aurora Serverless v2Biaya pada AWS tagihan Anda dihitung berdasarkan `ServerlessDatabaseCapacity` metrik yang sama yang dapat Anda pantau. Mekanisme penagihan dapat berbeda dari CloudWatch rata-rata yang dihitung untuk metrik ini jika Anda menggunakan Aurora Serverless v2 kapasitas hanya sebagian dari satu jam. Ini juga dapat berbeda jika masalah sistem membuat CloudWatch metrik tidak tersedia untuk periode singkat. Dengan demikian, Anda mungkin melihat nilai ACU-jam yang sedikit berbeda pada tagihan Anda daripada jika Anda menghitung sendiri angkanya dari nilai rata-rata `ServerlessDatabaseCapacity`.

Contoh CloudWatch perintah untuk Aurora Serverless v2 metrik

AWS CLI Contoh berikut menunjukkan bagaimana Anda dapat memantau CloudWatch metrik terpenting yang terkait Aurora Serverless v2 dengannya. Dalam setiap kasus, ganti string `Value=` untuk parameter `--dimensions` dengan pengidentifikasi instans DB Aurora Serverless v2 Anda sendiri.

Contoh Linux berikut menampilkan nilai kapasitas minimum, maksimum, dan rata-rata untuk instans DB, yang diukur setiap 10 menit selama satu jam. Perintah `date` Linux menentukan waktu mulai dan akhir secara relatif terhadap tanggal dan waktu saat ini. Fungsi `sort_by` dalam parameter `--query` mengurutkan hasil secara kronologis berdasarkan bidang `Timestamp`.

```
aws cloudwatch get-metric-statistics --metric-name "ServerlessDatabaseCapacity" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

Contoh Linux berikut menunjukkan pemantauan kapasitas setiap instans DB dalam sebuah klaster. Hal ini mengukur pemanfaatan kapasitas minimum, maksimum, dan rata-rata dari setiap instans DB. Pengukuran dilakukan sekali setiap jam selama periode tiga jam. Contoh-contoh ini menggunakan metrik `ACUUtilization` yang merepresentasikan persentase batas atas pada ACU, alih-alih `ServerlessDatabaseCapacity` yang merepresentasikan jumlah ACU tetap. Dengan begitu, Anda tidak perlu mengetahui angka sebenarnya untuk nilai ACU minimum dan maksimum dalam rentang kapasitas. Anda dapat melihat persentase mulai dari 0 hingga 100.

```
aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_writer_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table

aws cloudwatch get-metric-statistics --metric-name "ACUUtilization" \
  --start-time "$(date -d '3 hours ago')" --end-time "$(date -d 'now')" --period 3600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_reader_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

Contoh Linux berikut melakukan pengukuran yang sama seperti yang sebelumnya. Dalam hal ini, pengukuran ditujukan untuk metrik `CPUUtilization`. Pengukuran dilakukan setiap 10 menit selama periode 1 jam. Angka-angka tersebut merepresentasikan persentase CPU yang tersedia yang digunakan, berdasarkan sumber daya CPU yang tersedia untuk pengaturan kapasitas maksimum untuk instans DB.

```
aws cloudwatch get-metric-statistics --metric-name "CPUUtilization" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

Contoh Linux berikut melakukan pengukuran yang sama seperti yang sebelumnya. Dalam hal ini, pengukuran ditujukan untuk metrik `FreeableMemory`. Pengukuran dilakukan setiap 10 menit selama periode 1 jam.

```
aws cloudwatch get-metric-statistics --metric-name "FreeableMemory" \
  --start-time "$(date -d '1 hour ago')" --end-time "$(date -d 'now')" --period 600 \
  --namespace "AWS/RDS" --statistics Minimum Maximum Average \
  --dimensions Name=DBInstanceIdentifier,Value=my_instance \
  --query 'sort_by(Datapoints[*].
{min:Minimum,max:Maximum,avg:Average,ts:Timestamp},&ts)' --output table
```

Memantau performa Aurora Serverless v2 dengan Wawasan Performa

Anda dapat menggunakan Wawasan Performa untuk memantau performa instans DB Aurora Serverless v2. Untuk prosedur Wawasan Performa, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

Penghitung Wawasan Performa baru berikut ini berlaku untuk instans DB Aurora Serverless v2:

- `os.general.serverlessDatabaseCapacity` – Kapasitas instans DB saat ini dalam ACU. Nilai sesuai dengan `ServerlessDatabaseCapacity` CloudWatch metrik untuk instance DB.
- `os.general.acuUtilization` – Persentase kapasitas saat ini dari kapasitas maksimum yang dikonfigurasi. Nilai sesuai dengan `ACUUtilization` CloudWatch metrik untuk instance DB.
- `os.general.maxConfiguredAcu` – Kapasitas maksimum yang Anda konfigurasikan untuk instans DB Aurora Serverless v2 ini. Ini diukur dalam ACU.
- `os.general.minConfiguredAcu` – Kapasitas minimum yang Anda konfigurasikan untuk instans DB Aurora Serverless v2 ini. Ini diukur dalam ACU

Untuk daftar lengkap penghitung Wawasan Performa, lihat Metrik penghitung Wawasan Performa.

Ketika nilai `vCPU` ditampilkan untuk instans DB Aurora Serverless v2 di Wawasan Performa, nilai tersebut merepresentasikan perkiraan berdasarkan nilai ACU untuk instans DB ini. Pada interval default satu menit, nilai vCPU fraksional apa pun dibulatkan ke bilangan bulat terdekat. Untuk interval yang lebih lama, nilai vCPU yang ditampilkan adalah rata-rata nilai vCPU bilangan bulat untuk setiap menit.

Memecahkan masalah kapasitas Aurora Serverless v2

Dalam beberapa kasus, Aurora Serverless v2 tidak diturunkan skalanya ke kapasitas minimum, bahkan tanpa beban pada basis data. Hal ini dapat terjadi karena alasan berikut:

- Fitur tertentu dapat meningkatkan penggunaan sumber daya dan mencegah basis data diturunkan skalanya ke kapasitas minimum. Fitur-fitur ini mencakup hal-hal berikut:
  - Basis data global Aurora
  - Mengekspor CloudWatch Log
  - Mengaktifkan `pg_audit` pada klaster DB yang kompatibel dengan Aurora PostgreSQL
  - Pemantauan yang Ditingkatkan
  - Wawasan Performa

Untuk informasi selengkapnya, lihat Memilih pengaturan kapasitas Aurora Serverless v2 minimum untuk klaster.

- Jika instans pembaca tidak menurunkan skalanya ke nilai minimum dan tetap pada kapasitas yang sama atau lebih tinggi dari instans penulis, periksa tingkat prioritas instans pembaca. Instans DB pembaca Aurora Serverless v2 di tingkat 0 atau 1 akan dipertahankan pada kapasitas minimum setidaknya setinggi instans DB penulis. Ubah tingkat prioritas pembaca menjadi 2 atau lebih tinggi agar dinaikkan dan diturunkan skalanya secara independen dari penulis. Untuk informasi selengkapnya, lihat Memilih tingkat promosi untuk pembaca Aurora Serverless v2.
- Tetapkan parameter basis data apa pun yang memengaruhi ukuran memori bersama ke nilai default-nya. Mengatur nilai yang lebih tinggi dari nilai default akan meningkatkan kebutuhan memori bersama dan mencegah basis data menurunkan skalanya ke kapasitas minimum. Contohnya adalah `max_connections` dan `max_locks_per_transaction`.

  > **Note**
  >
  > Memperbarui parameter memori bersama memerlukan pengaktifan ulang basis data agar perubahan diterapkan.

- Beban kerja basis data yang berat dapat meningkatkan penggunaan sumber daya.
- Volume basis data yang besar dapat meningkatkan penggunaan sumber daya.

  Amazon Aurora menggunakan memori dan sumber daya CPU untuk manajemen klaster DB. Aurora membutuhkan lebih banyak CPU dan memori untuk mengelola klaster DB dengan volume basis data yang lebih besar. Jika kapasitas minimum klaster Anda kurang dari minimum yang diperlukan untuk pengelolaan klaster, klaster Anda tidak akan menurunkan kapasitas minimum.
- Proses latar belakang, seperti purging, juga dapat meningkatkan penggunaan sumber daya.

Jika basis data masih tidak menurunkan skalanya ke kapasitas minimum yang dikonfigurasi, hentikan dan aktifkan ulang basis data untuk mengklaim kembali fragmen memori yang mungkin telah terakumulasi dari waktu ke waktu. Menghentikan dan memulai basis data akan menghasilkan waktu henti, jadi sebaiknya lakukan ini seperlunya.

Migrasi ke Aurora Serverless v2

Untuk mengonversi klaster DB yang ada untuk menggunakan Aurora Serverless v2, Anda dapat melakukan hal berikut:

- Upgrade dari klaster DB Aurora terprovisi.
- Upgrade dari klaster Aurora Serverless v1.
- Migrasikan dari basis data on-premise ke klaster Aurora Serverless v2.

Saat klaster yang di-upgrade menjalankan versi mesin yang sesuai seperti yang tercantum dalam Persyaratan dan batasan untuk Aurora Serverless v2, Anda dapat mulai menambahkan instans DB Aurora Serverless v2 ke dalamnya. Instans DB pertama yang Anda tambahkan ke klaster yang di-upgrade harus berupa instans DB terprovisi. Kemudian, Anda dapat mengalihkan pemrosesan untuk beban kerja tulis, beban kerja baca, atau keduanya ke instans DB Aurora Serverless v2.

Daftar Isi

- Meng-upgrade atau beralih dari klaster yang ada ke Aurora Serverless v2
  - Jalur upgrade untuk klaster yang kompatibel dengan MySQL untuk menggunakan Aurora Serverless v2
  - Jalur upgrade untuk klaster yang kompatibel dengan PostgreSQL untuk menggunakan Aurora Serverless v2
- Beralih dari klaster terprovisi ke Aurora Serverless v2
- Perbandingan Aurora Serverless v2 dan Aurora Serverless v1
  - Perbandingan persyaratan Aurora Serverless v2 dan Aurora Serverless v1
  - Perbandingan penskalaan dan ketersediaan Aurora Serverless v2 dan Aurora Serverless v1
  - Perbandingan dukungan fitur Aurora Serverless v2 dan Aurora Serverless v1
  - Mengadaptasi kasus penggunaan Aurora Serverless v1 ke Aurora Serverless v2
- Upgrade dari klaster Aurora Serverless v1 ke Aurora Serverless v2
  - Klaster DB yang kompatibel dengan Aurora MySQL
  - Klaster DB yang kompatibel dengan Aurora PostgreSQL
- Bermigrasi dari basis data on-premise ke Aurora Serverless v2

> **Note**
>
> Topik-topik ini menjelaskan cara mengonversi klaster DB yang ada. Untuk informasi tentang membuat klaster DB Aurora Serverless v2 baru, lihat Membuat cluster DB yang menggunakan Aurora Serverless v2.

Meng-upgrade atau beralih dari klaster yang ada ke Aurora Serverless v2

Jika klaster terprovisi memiliki versi mesin yang mendukung Aurora Serverless v2, peralihan ke Aurora Serverless v2 tidak memerlukan upgrade. Dalam hal ini, Anda dapat menambahkan instans DB Aurora Serverless v2 ke klaster asli Anda. Anda dapat beralih dari klaster ini ke semua instans DB Aurora Serverless v2. Anda juga dapat menggunakan kombinasi instans DB Aurora Serverless v2 dan instans DB terprovisi di klaster DB yang sama. Untuk versi mesin Aurora yang mendukung Aurora Serverless v2, lihat Aurora Serverless v2.

Jika Anda menjalankan versi mesin yang lebih rendah yang tidak mendukung Aurora Serverless v2, Anda perlu mengambil langkah-langkah umum ini:

1. Upgrade klaster.
2. Buat instans DB penulis terprovisi untuk klaster yang di-upgrade.
3. Ubah klaster untuk menggunakan instans DB Aurora Serverless v2.

> **Important**
>
> Saat Anda melakukan upgrade versi mayor ke versi yang kompatibel dengan Aurora Serverless v2 menggunakan pemulihan atau kloning snapshot, instans DB pertama yang Anda tambahkan ke klaster baru harus berupa instans DB terprovisi. Penambahan ini akan memulai tahap akhir proses upgrade.
> Sampai tahap akhir itu terjadi, klaster tidak memiliki infrastruktur yang diperlukan untuk dukungan Aurora Serverless v2. Dengan demikian, klaster yang di-upgrade ini selalu dimulai dengan instans DB penulis terprovisi. Kemudian, Anda dapat melakukan konversi atau failover instans DB terprovisi ke instans DB Aurora Serverless v2.

Upgrade dari Aurora Serverless v1 ke Aurora Serverless v2 memerlukan pembuatan klaster terprovisi sebagai langkah perantara. Kemudian, Anda melakukan langkah upgrade yang sama seperti ketika Anda memulai dengan klaster terprovisi.

Jalur upgrade untuk klaster yang kompatibel dengan MySQL untuk menggunakan Aurora Serverless v2

Jika klaster asli Anda menjalankan Aurora MySQL, pilih prosedur yang sesuai tergantung pada versi mesin dan mode mesin klaster Anda.

Jika klaster Aurora MySQL asli Anda adalah yang ini	Lakukan tindakan ini untuk beralih ke Aurora Serverless v2
Klaster terprovisi yang menjalankan Aurora MySQL versi 3, kompatibel dengan MySQL 8.0	Ini adalah tahap akhir untuk semua konversi dari klaster Aurora MySQL yang ada. Jika perlu, lakukan upgrade versi minor ke versi 3.02.0 atau lebih tinggi. Gunakan instans DB terprovisi untuk instans DB penulis. Tambahkan satu instans DB pembaca Aurora Serverless v2. Lakukan failover untuk menjadikan instans DB ini sebagai instans DB penulis. (Opsional) Konversi instans DB terprovisi lain di klaster menjadi Aurora Serverless v2. Atau, tambahkan instans DB Aurora Serverless v2 baru dan hapus instans DB terprovisi. Untuk prosedur dan contoh lengkap, lihat Beralih dari klaster terprovisi ke Aurora Serverless v2.
Klaster terprovisi yang menjalankan Aurora MySQL versi 2, kompatibel dengan MySQL 5.7	Lakukan upgrade versi mayor ke Aurora MySQL versi 3.02.0 atau lebih tinggi. Kemudian, ikuti prosedur untuk Aurora MySQL versi 3 untuk beralih dari klaster ini ke instans DB Aurora Serverless v2.
Klaster Aurora Serverless v1 yang menjalank an Aurora MySQL versi 2, kompatibel dengan MySQL 5.7	Untuk membantu merencanakan konversi Anda dari Aurora Serverless v1, baca terlebih dahulu Perbandingan Aurora Serverless v2 dan Aurora Serverless v1. Kemudian, ikuti prosedur dalam Upgrade dari klaster Aurora Serverless v1 ke Aurora Serverless v2.

Jalur upgrade untuk klaster yang kompatibel dengan PostgreSQL untuk menggunakan Aurora Serverless v2

Jika klaster asli Anda menjalankan Aurora PostgreSQL, pilih prosedur yang sesuai tergantung pada versi mesin dan mode mesin klaster Anda.

Jika klaster Aurora PostgreSQL asli Anda adalah yang ini	Lakukan tindakan ini untuk beralih ke Aurora Serverless v2
Klaster terprovisi yang menjalankan Aurora PostgreSQL versi 13	Ini adalah tahap akhir untuk semua konversi dari klaster Aurora PostgreSQL yang ada. Jika perlu, lakukan upgrade versi minor ke versi 13.6 atau lebih tinggi. Tambahkan satu instans DB terprovisi untuk instans DB penulis. Tambahkan satu instans DB pembaca Aurora Serverless v2. Lakukan failover untuk menjadikan instans Aurora Serverless v2 tersebut sebagai instans DB penulis. (Opsional) Konversi instans DB terprovisi lain di klaster menjadi Aurora Serverless v2. Atau, tambahkan instans DB Aurora Serverless v2 baru dan hapus instans DB terprovisi. Untuk prosedur dan contoh lengkap, lihat Beralih dari klaster terprovisi ke Aurora Serverless v2.
Cluster yang disediakan menjalankan Aurora PostgreSQL versi 11 atau 12	Lakukan upgrade versi mayor ke Aurora PostgreSQL versi 13.6 atau lebih tinggi. Kemudian, ikuti prosedur untuk Aurora PostgreSQL versi 13 untuk beralih dari klaster ini ke instans DB Aurora Serverless v2.
Klaster Aurora Serverless v1 yang menjalankan Aurora PostgreSQL versi 11 atau 13	Untuk membantu merencanakan konversi Anda dari Aurora Serverless v1, baca terlebih dahulu

Jika klaster Aurora PostgreSQL asli Anda adalah yang ini	Lakukan tindakan ini untuk beralih ke Aurora Serverless v2
	Perbandingan Aurora Serverless v2 dan Aurora Serverless v1. Kemudian, ikuti prosedur dalam Upgrade dari klaster Aurora Serverless v1 ke Aurora Serverless v2.

Beralih dari klaster terprovisi ke Aurora Serverless v2

Untuk beralih dari klaster terprovisi ke Aurora Serverless v2, ikuti langkah-langkah berikut:

1. Periksa apakah klaster terprovisi perlu di-upgrade untuk digunakan dengan instans DB Aurora Serverless v2. Untuk versi Aurora yang kompatibel dengan Aurora Serverless v2, lihat Persyaratan dan batasan untuk Aurora Serverless v2.

   Jika klaster terprovisi yang menjalankan versi mesin yang tidak tersedia untuk Aurora Serverless v2, upgrade versi mesin klaster:

   - Jika Anda memiliki klaster terprovisi yang kompatibel dengan MySQL 5.7, ikuti petunjuk upgrade untuk Aurora MySQL versi 3. Gunakan prosedur dalam Meningkatkan ke Aurora MySQL versi 3.
   - Jika Anda memiliki klaster terprovisi yang kompatibel dengan PostgreSQL dan menjalankan PostgreSQL versi 11 atau 12, ikuti petunjuk upgrade untuk Aurora PostgreSQL versi 13. Gunakan prosedur dalam Cara melakukan peningkatan versi mayor.
2. Konfigurasikan properti klaster lainnya agar sesuai dengan persyaratan Aurora Serverless v2 dari Persyaratan dan batasan untuk Aurora Serverless v2.
3. Konfigurasikan konfigurasi penskalaan untuk klaster. Ikuti prosedur dalam Mengatur rentang kapasitas Aurora Serverless v2 untuk sebuah klaster.
4. Tambahkan satu atau beberapa instans DB Aurora Serverless v2 ke klaster. Ikuti prosedur umum dalam Menambahkan Replika Aurora ke klaster DB. Untuk setiap instans DB baru, tentukan nama kelas instans DB khusus Tanpa Server di AWS Management Console, atau `db.serverless` di API RDS AWS CLI Amazon atau Amazon.

   Dalam beberapa kasus, Anda mungkin sudah memiliki satu atau beberapa instans DB pembaca terprovisi di klaster. Jika demikian, Anda dapat mengonversi salah satu pembaca ke instans DB

Aurora Serverless v2 alih-alih membuat instans DB baru. Untuk melakukannya, ikuti prosedur dalam Mengonversi penulis atau pembaca terprovisi menjadi Aurora Serverless v2.

5. Lakukan operasi failover untuk menjadikan salah satu instans DB Aurora Serverless v2 sebagai instans DB penulis untuk klaster.
6. (Opsional) Konversi instans DB terprovisi menjadi Aurora Serverless v2, atau hapus dari klaster. Ikuti prosedur umum dalam Mengonversi penulis atau pembaca terprovisi menjadi Aurora Serverless v2 atau Menghapus instans dari klaster DB Aurora.

> **Tip**
>
> Menghapus instans DB terprovisi tidaklah wajib. Anda dapat mengatur klaster yang berisi instans DB Aurora Serverless v2 dan instans DB terprovisi. Namun, jika Anda belum memahami karakteristik performa dan penskalaan instans DB Aurora Serverless v2, sebaiknya konfigurasi klaster Anda dengan instans DB semuanya dari jenis yang sama.

AWS CLI Contoh berikut menunjukkan proses peralihan menggunakan klaster yang disediakan yang menjalankan Aurora MySQL versi 3.02.0. Klasternya bernama `mysql-80`. Klaster ini dimulai dengan dua instans DB terprovisi bernama `provisioned-instance-1` dan `provisioned-instance-2`, satu penulis dan satu pembaca. Keduanya menggunakan kelas instans DB `db.r6g.large`.

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' --output text
mysql-80
provisioned-instance-2     False
provisioned-instance-1     True

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-1 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-1     db.r6g.large

$ aws rds describe-db-instances --db-instance-identifier provisioned-instance-2 \
  --output text --query '*[].[DBInstanceIdentifier,DBInstanceClass]'
provisioned-instance-2     db.r6g.large
```

Kita membuat tabel dengan beberapa data. Dengan demikian, kita dapat mengonfirmasi bahwa data ini dan operasi klaster sama sebelum dan setelah switchover.

```
mysql> create database serverless_v2_demo;
mysql> create table serverless_v2_demo.demo (s varchar(128));
mysql> insert into serverless_v2_demo.demo values ('This cluster started with a
 provisioned writer.');
Query OK, 1 row affected (0.02 sec)
```

Pertama, kita menambahkan rentang kapasitas ke klaster ini. Jika tidak, kita akan mendapatkan kesalahan saat menambahkan instans DB Aurora Serverless v2 apa pun ke klaster ini. Jika kita menggunakan AWS Management Console untuk prosedur ini, langkah itu otomatis ketika kita menambahkan instance Aurora Serverless v2 DB pertama.

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 \
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql

An error occurred (InvalidDBClusterStateFault) when calling the CreateDBInstance
 operation:
Set the Serverless v2 scaling configuration on the parent DB cluster before creating a
 Serverless v2 DB instance.

$ # The blank ServerlessV2ScalingConfiguration attribute confirms that the cluster
 doesn't have a capacity range set yet.
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 --query
 'DBClusters[*].ServerlessV2ScalingConfiguration'
[]

$ aws rds modify-db-cluster --db-cluster-identifier mysql-80 \
  --serverless-v2-scaling-configuration MinCapacity=0.5,MaxCapacity=16
{
  "DBClusterIdentifier": "mysql-80",
  "ServerlessV2ScalingConfiguration": {
    "MinCapacity": 0.5,
    "MaxCapacity": 16
  }
}
```

Kita membuat dua pembaca Aurora Serverless v2 untuk menggantikan instans DB asli. Kita melakukannya dengan menentukan kelas instans DB `db.serverless` untuk instans DB baru.

```
$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-1 --db-
cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-mysql
```

```
{
  "DBInstanceIdentifier": "serverless-v2-instance-1",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceStatus": "creating"
}

$ aws rds create-db-instance --db-instance-identifier serverless-v2-instance-2 \
  --db-cluster-identifier mysql-80 --db-instance-class db.serverless --engine aurora-
mysql
{
  "DBInstanceIdentifier": "serverless-v2-instance-2",
  "DBClusterIdentifier": "mysql-80",
  "DBInstanceClass": "db.serverless",
  "DBInstanceStatus": "creating"
}

$ # Wait for both DB instances to finish being created before proceeding.
$ aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-1
 && \
  aws rds wait db-instance-available --db-instance-identifier serverless-v2-instance-2
```

Kita melakukan failover untuk menjadikan salah satu instans DB Aurora Serverless v2 sebagai penulis baru untuk klaster.

```
$ aws rds failover-db-cluster --db-cluster-identifier mysql-80 \
  --target-db-instance-identifier serverless-v2-instance-1
{
  "DBClusterIdentifier": "mysql-80",
  "DBClusterMembers": [
    {
      "DBInstanceIdentifier": "serverless-v2-instance-1",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "serverless-v2-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
```

```
      "DBInstanceIdentifier": "provisioned-instance-2",
      "IsClusterWriter": false,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    },
    {
      "DBInstanceIdentifier": "provisioned-instance-1",
      "IsClusterWriter": true,
      "DBClusterParameterGroupStatus": "in-sync",
      "PromotionTier": 1
    }
  ],
  "Status": "available"
}
```

Perlu beberapa detik agar perubahan tersebut diterapkan. Pada saat itu, kita akan memiliki satu penulis Aurora Serverless v2 dan satu pembaca Aurora Serverless v2. Jadi, kita tidak memerlukan instans DB asli terprovisi mana pun.

```
$ aws rds describe-db-clusters --db-cluster-identifier mysql-80 \
  --query '*[].[DBClusterIdentifier,DBClusterMembers[*].
[DBInstanceIdentifier,IsClusterWriter]]' \
  --output text
mysql-80
serverless-v2-instance-1        True
serverless-v2-instance-2        False
provisioned-instance-2          False
provisioned-instance-1          False
```

Langkah terakhir dalam prosedur switchover adalah menghapus kedua instans DB terprovisi.

```
$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-2 --skip-
final-snapshot
{
  "DBInstanceIdentifier": "provisioned-instance-2",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}

$ aws rds delete-db-instance --db-instance-identifier provisioned-instance-1 --skip-
final-snapshot
```

```
{
  "DBInstanceIdentifier": "provisioned-instance-1",
  "DBInstanceStatus": "deleting",
  "Engine": "aurora-mysql",
  "EngineVersion": "8.0.mysql_aurora.3.02.0",
  "DBInstanceClass": "db.r6g.large"
}
```

Sebagai pemeriksaan terakhir, kita akan mengonfirmasi bahwa tabel asli dapat diakses dan dapat ditulis dari instans DB penulis Aurora Serverless v2.

```
mysql> select * from serverless_v2_demo.demo;
+--------------------------------------------------+
| s                                                |
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
+--------------------------------------------------+
1 row in set (0.00 sec)

mysql> insert into serverless_v2_demo.demo values ('And it finished with a Serverless
 v2 writer.');
Query OK, 1 row affected (0.01 sec)

mysql> select * from serverless_v2_demo.demo;
+--------------------------------------------------+
| s                                                |
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
| And it finished with a Serverless v2 writer.     |
+--------------------------------------------------+
2 rows in set (0.01 sec)
```

Kita juga menghubungkan ke instans DB pembaca Aurora Serverless v2 dan mengonfirmasi bahwa data yang baru ditulis juga tersedia di sana.

```
mysql> select * from serverless_v2_demo.demo;
+--------------------------------------------------+
| s                                                |
+--------------------------------------------------+
| This cluster started with a provisioned writer.  |
| And it finished with a Serverless v2 writer.     |
+--------------------------------------------------+
```

```
2 rows in set (0.01 sec)
```

Perbandingan Aurora Serverless v2 dan Aurora Serverless v1

Jika Anda sudah menggunakan Aurora Serverless v1, Anda dapat mempelajari perbedaan besar antara Aurora Serverless v1 dan Aurora Serverless v2. Perbedaan arsitektur, seperti dukungan untuk instans DB pembaca, menyediakan jenis kasus penggunaan baru.

Anda dapat menggunakan tabel berikut untuk membantu memahami perbedaan paling penting antara Aurora Serverless v2 dan Aurora Serverless v1.

Topik

- Perbandingan persyaratan Aurora Serverless v2 dan Aurora Serverless v1
- Perbandingan penskalaan dan ketersediaan Aurora Serverless v2 dan Aurora Serverless v1
- Perbandingan dukungan fitur Aurora Serverless v2 dan Aurora Serverless v1
- Mengadaptasi kasus penggunaan Aurora Serverless v1 ke Aurora Serverless v2

Perbandingan persyaratan Aurora Serverless v2 dan Aurora Serverless v1

Tabel berikut merangkum persyaratan yang berbeda untuk menjalankan basis data Anda menggunakan Aurora Serverless v2 atau Aurora Serverless v1. Aurora Serverless v2 menawarkan versi mesin DB Aurora MySQL dan Aurora PostgreSQL yang lebih tinggi daripada Aurora Serverless v1.

Fitur	Persyaratan Aurora Serverless v2	Persyaratan Aurora Serverless v1
Mesin DB	Aurora MySQL, Aurora PostgreSQL	Aurora MySQL, Aurora PostgreSQL
Versi Aurora MySQL yang didukung	Lihat Aurora Serverless v2 dengan Aurora MySQL.	Lihat Aurora Serverless v1 dengan Aurora MySQL.
Versi Aurora PostgreSQL yang didukung	Lihat Aurora Serverless v2 dengan Aurora PostgreSQL.	Lihat Aurora Serverless v1 dengan Aurora PostgreSQL.
Meng-upgrade klaster DB	Demikian pula dengan klaster DB terprovisi, Anda dapat	Upgrade versi minor diterapkan secara otomatis saat

Fitur	Persyaratan Aurora Serverless v2	Persyaratan Aurora Serverless v1
	melakukan upgrade secara manual tanpa menunggu Aurora meng-upgrade klaster DB untuk Anda. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Amazon Aurora. **Note** Untuk melakukan upgrade versi mayor dari 13.x ke 14.x atau 15.x untuk klaster DB yang kompatibel dengan Aurora PostgreSQL, kapasitas maksimum klaster Anda harus minimal 2 ACU.	tersedia. Untuk informasi selengkapnya, lihat Aurora Serverless v1 dan versi mesin basis data Aurora. Anda dapat melakukan upgrade versi mayor secara manual. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Aurora Serverless v1.

Fitur	Persyaratan Aurora Serverless v2	Persyaratan Aurora Serverless v1
Mengonversi dari klaster DB terprovisi	Anda dapat menggunakan metode berikut: • Tambahkan satu atau beberapa instans DB pembaca Aurora Serverless v2 ke klaster terprovisi yang sudah ada. Untuk menggunakan Aurora Serverless v2 untuk penulis, lakukan failover ke salah satu instans DB Aurora Serverless v2. Agar seluruh klaster menggunakan instans DB Aurora Serverless v2, hapus instans DB penulis terprovisi setelah mempromosikan instans DB Aurora Serverless v2 menjadi penulis. • Buat klaster baru dengan mesin DB dan versi mesin yang sesuai. Gunakan salah satu metode standar. Misalnya, pulihkan snapshot klaster atau buat klon dari klaster yang ada. Pilih Aurora Serverless v2 untuk beberapa atau semua instans DB di klaster baru. Jika Anda membuat klaster baru melalui kloning, Anda tidak dapat meng-upgr	Pulihkan snapshot dari klaster terprovisi untuk membuat klaster Aurora Serverless v1 baru.

Fitur	Persyaratan Aurora Serverless v2	Persyaratan Aurora Serverless v1
	ade versi mesin secara bersamaan. Pastikan bahwa klaster asli sudah menjalankan versi mesin yang kompatibel dengan Aurora Serverless v2.	
Mengonversi dari klaster Aurora Serverless v1	Ikuti prosedur dalam Upgrade dari klaster Aurora Serverless v1 ke Aurora Serverless v2.	Tidak berlaku
Kelas instans DB yang tersedia	Kelas instans DB khusus `db.serverless` . Di AWS Management Console, itu diberi label sebagai Tanpa Server.	Tidak berlaku. Aurora Serverless v1 menggunakan mode mesin `serverless` .
Port	Port apa pun yang kompatibel dengan MySQL atau PostgreSQL	Port MySQL atau PostgreSQL default saja
Alamat IP publik diizinkan?	Ya	Tidak
Cloud privat virtual (VPC) diperlukan?	Ya	Ya. Setiap klaster Aurora Serverless v1 menggunakan 2 antarmuka dan titik akhir Penyeimbang Beban Gateway yang dialokasikan ke VPC Anda.

Perbandingan penskalaan dan ketersediaan Aurora Serverless v2 dan Aurora Serverless v1

Tabel berikut merangkum perbedaan antara Aurora Serverless v2 dan Aurora Serverless v1 dari segi skalabilitas dan ketersediaan.

Penskalaan Aurora Serverless v2 lebih responsif, lebih terperinci, dan tidak terlalu mengganggu daripada penskalaan Aurora Serverless v1. Aurora Serverless v2 dapat diskalakan dengan mengubah ukuran instans DB dan dengan menambahkan lebih banyak instans DB ke klaster DB. Hal ini juga dapat skala dengan menambahkan cluster lain Wilayah AWS ke database global Aurora. Sebaliknya, Aurora Serverless v1 diskalakan hanya dengan menambahkan atau mengurangi kapasitas penulis. Semua komputasi untuk klaster Aurora Serverless v1 berjalan di satu Zona Ketersediaan dan satu Wilayah AWS.

Fitur penskalaan dan ketersediaan tinggi	Aurora Serverless v2perilaku	Aurora Serverless v1perilaku
Unit kapasitas Aurora (ACU) minimum (Aurora MySQL)	0,5	1 saat klaster berjalan, 0 saat klaster dijeda.
ACU minimum (Aurora PostgreSQL)	0,5	2 saat klaster berjalan, 0 saat klaster dijeda.
ACU maksimum (Aurora MySQL)	128	256
ACU maksimum (Aurora PostgreSQL)	128	384
Menghentikan klaster	Anda dapat menghentikan dan memulai klaster secara manual dengan menggunakan fitur hentikan dan mulai klaster yang sama dengan klaster terprovisi.	Klaster berhenti secara otomatis setelah batas waktu habis. Klaster memerlukan beberapa waktu untuk tersedia ketika aktivitas dilanjutkan.
Penskalaan untuk instans DB	Skala naik dan turun dengan inkremen minimum 0,5 ACU.	Skala naik dan turun dengan menggandakan atau mengurangi separuh ACU.
Jumlah instans DB	Sama seperti klaster terprovisi: 1 instans DB penulis, maksimal 15 instans DB pembaca.	1 instans DB menangani baca dan tulis.

Fitur penskalaan dan ketersediaan tinggi	Aurora Serverless v2perilaku	Aurora Serverless v1perilaku
Penskalaan dapat terjadi saat pernyataan SQL berjalan?	Ya. Aurora Serverless v2 tidak perlu menunggu titik diam.	Tidak. Misalnya, penskalaan menunggu penyelesaian transaksi, tabel sementara, dan kunci tabel yang berjalan lama.
Instans DB pembaca diskalakan bersama dengan penulis	Opsional.	Tidak berlaku.
Penyimpanan maksimum	128 TiB	128 TiB atau 64 TiB, tergantung pada mesin basis data dan versi.
Cache buffer dipertahankan saat penskalaan	Ya. Cache buffer diubah ukurannya secara dinamis.	Tidak. Rewarming cache buffer dilakukan setelah penskalaan.
Failover	Ya, sama seperti untuk klaster terprovisi.	Upaya terbaik saja, tergantung pada ketersediaan kapasitas. Lebih lambat daripada di Aurora Serverless v2.
Kemampuan Multi-AZ	Ya, sama seperti untuk klaster terprovisi. Klaster Multi-AZ memerlukan instans DB pembaca di Zona Ketersediaan (AZ) kedua. Untuk klaster Multi-AZ, Aurora melakukan failover Multi-AZ jika terjadi kegagalan AZ.	Klaster Aurora Serverless v1 menjalankan semua komputasinya di satu AZ. Pemulihan jika terjadi kegagalan AZ adalah upaya terbaik saja dan tergantung pada ketersediaan kapasitas.
Basis data global Aurora	Ya	Tidak

Fitur penskalaan dan ketersediaan tinggi	Aurora Serverless v2perilaku	Aurora Serverless v1perilaku
Penskalaan berdasarkan tekanan memori	Ya	Tidak
Penskalaan berdasarkan beban CPU	Ya	Ya
Penskalaan berdasarkan lalu lintas jaringan	Ya, berdasarkan overhead memori dan CPU untuk lalu lintas jaringan. Parameter `max_connections` tetap konstan untuk menghindari terputusnya koneksi saat menurunkan skala.	Ya, berdasarkan jumlah koneksi.
Tindakan batas waktu untuk peristiwa penskalaan	Tidak	Ya
Menambahkan instance DB baru ke cluster melalui AWS Auto Scaling	Tidak berlaku. Anda dapat membuat instans DB pembaca Aurora Serverless v2 di tingkat promosi 2–15 dan biarkan instans tersebut diturunkan skalanya ke kapasitas rendah.	Tidak. Instans DB pembaca tidak tersedia.

Perbandingan dukungan fitur Aurora Serverless v2 dan Aurora Serverless v1

Tabel berikut merangkum hal ini:

- Fitur yang tersedia di Aurora Serverless v2, tetapi tidak di Aurora Serverless v1
- Fitur yang berfungsi secara berbeda antara Aurora Serverless v1 dan Aurora Serverless v2
- Fitur yang saat ini tidak tersedia di Aurora Serverless v2

Aurora Serverless v2 mencakup banyak fitur dari klaster terprovisi yang tidak tersedia untuk Aurora Serverless v1.

Fitur	Dukungan Aurora Serverless v2	Dukungan Aurora Serverless v1
Topologi klaster	Aurora Serverless v2 adalah properti dari instans DB individu. Klaster dapat berisi beberapa instans DB Aurora Serverless v2, atau kombinasi instans DB Aurora Serverless v2 dan instans DB terprovisi.	Klaster Aurora Serverless v1 tidak menggunakan karakteristik instans DB. Anda tidak dapat mengubah properti Aurora Serverless v1 setelah membuat klaster.
Parameter konfigurasi	Hampir semua parameter yang sama dapat dimodifikasi seperti pada klaster terprovisi. Untuk detailnya, lihat Menggunakan grup parameter untuk Aurora Serverless v2.	Hanya sebagian parameter yang dapat dimodifikasi.
Grup parameter	Grup parameter klaster dan grup parameter DB. Parameter dengan nilai `provisioned` dalam atribut `SupportedEngineModes` tersedia. Parameter tersebut lebih banyak daripada di Aurora Serverless v1.	Grup parameter klaster saja. Parameter dengan nilai `serverless` dalam atribut `SupportedEngineModes` tersedia.
Enkripsi untuk volume klaster	Opsional	Wajib. Batasan di berlaku untuk semua klaster Aurora Serverless v1.
Snapshot lintas Wilayah	Ya	Snapshot harus dienkripsi dengan kunci AWS Key Management Service () AWS KMS Anda sendiri.

Fitur	Dukungan Aurora Serverless v2	Dukungan Aurora Serverless v1
Cadangan otomatis dipertahankan setelah penghapusan klaster DB	Ya	Tidak
TLS/SSL	Ya. Dukungannya sama dengan klaster terprovisi. Untuk informasi penggunaan, lihat Menggunakan TLS/SSL dengan Aurora Serverless v2.	Ya. Ada beberapa perbedaan dari dukungan TLS untuk klaster terprovisi. Untuk informasi penggunaan, lihat Menggunakan TLS/SSL dengan Aurora Serverless v1.
Kloning	Hanya dari dan ke versi mesin DB yang kompatibel dengan Aurora Serverless v2. Anda tidak dapat menggunakan kloning untuk meng-upgrade dari klaster Aurora Serverless v1 atau dari klaster terprovisi versi sebelumnya.	Hanya dari dan ke versi mesin DB yang kompatibel dengan Aurora Serverless v1.
Integrasi dengan Amazon S3	Ya	Ya
Integrasi dengan AWS Secrets Manager	Tidak	Tidak
Mengekspor snapshot klaster DB ke S3	Ya	Tidak
Mengaitkan peran IAM	Ya	Tidak
Mengunggah log ke Amazon CloudWatch	Opsional. Anda memilih log mana yang akan diaktifkan dan log mana yang akan diunggah CloudWatch.	Semua log yang diaktifkan diunggah secara CloudWatch otomatis.
Data API tersedia	Ya	Ya

Fitur	Dukungan Aurora Serverless v2	Dukungan Aurora Serverless v1
Editor kueri tersedia	Ya	Ya
Wawasan Performa	Ya	Tidak
Proksi Amazon RDS tersedia	Ya	Tidak
Babelfish for Aurora PostgreSQL tersedia	Ya. Didukung untuk versi Aurora PostgreSQL yang kompatibel dengan Babelfish dan Aurora Serverless v2.	Tidak

Mengadaptasi kasus penggunaan Aurora Serverless v1 ke Aurora Serverless v2

Bergantung pada kasus penggunaan Anda untuk Aurora Serverless v1, Anda dapat menyesuaikan pendekatan tersebut untuk memanfaatkan fitur Aurora Serverless v2 sebagai berikut.

Misalkan Anda memiliki klaster Aurora Serverless v1 yang diberi beban ringan dan prioritas Anda adalah menjaga ketersediaan yang berkelanjutan sambil meminimalkan biaya. Dengan Aurora Serverless v2, Anda dapat mengonfigurasi pengaturan ACU minimum yang lebih kecil sebesar 0,5, dibandingkan dengan minimal 1 ACU untuk Aurora Serverless v1. Anda dapat meningkatkan ketersediaan dengan membuat konfigurasi Multi-AZ, dengan instans DB pembaca yang juga memiliki minimal 0,5 ACU.

Misalkan Anda memiliki klaster Aurora Serverless v1 yang Anda gunakan dalam skenario pengembangan dan pengujian. Dalam hal ini, biaya juga merupakan prioritas tinggi, tetapi klaster tidak perlu tersedia setiap saat. Saat ini, Aurora Serverless v2 tidak secara otomatis dijeda saat klaster benar-benar idle. Sebagai gantinya, Anda dapat menghentikan klaster secara manual saat tidak diperlukan, dan memulainya saat waktunya untuk siklus pengujian atau pengembangan berikutnya.

Misalkan Anda memiliki klaster Aurora Serverless v1 dengan beban kerja yang berat. Sebuah klaster setara yang menggunakan Aurora Serverless v2 dapat diskalakan dengan lebih banyak granularitas. Misalnya, Aurora Serverless v1 diskalakan dengan menggandakan kapasitas, misalnya dari 64 menjadi 128 ACU. Sebaliknya, instans DB Aurora Serverless v2 Anda dapat diskalakan ke sebuah nilai di antara angka-angka tersebut.

Misalkan beban kerja Anda membutuhkan kapasitas total yang lebih tinggi daripada yang tersedia di Aurora Serverless v1. Anda dapat menggunakan beberapa instans DB pembaca Aurora Serverless v2 untuk mengambil alih bagian beban kerja yang sarat baca dari instans DB penulis. Anda juga dapat membagi beban kerja yang sarat baca di antara beberapa instans DB pembaca.

Untuk beban kerja sarat tulis, Anda dapat mengonfigurasi klaster dengan instans DB besar terprovisi sebagai penulis. Anda dapat melakukannya bersama satu atau beberapa instans DB pembaca Aurora Serverless v2.

Upgrade dari klaster Aurora Serverless v1 ke Aurora Serverless v2

Proses meng-upgrade klaster DB dari Aurora Serverless v1 ke Aurora Serverless v2 memiliki beberapa langkah. Hal ini karena Anda tidak dapat mengonversi langsung dari Aurora Serverless v1 ke Aurora Serverless v2. Selalu ada langkah perantara yang memerlukan konversi klaster DB Aurora Serverless v1 ke klaster terprovisi.

Klaster DB yang kompatibel dengan Aurora MySQL

Anda dapat mengonversi cluster Aurora Serverless v1 DB Anda ke cluster DB yang disediakan, lalu menggunakan penerapan biru/hijau untuk memutakhirkannya dan mengubahnya menjadi cluster DB. Aurora Serverless v2 Kami merekomendasikan prosedur ini untuk lingkungan produksi. Untuk informasi selengkapnya, lihat Menggunakan Deployment Blue/Green Amazon RDS untuk pembaruan basis data.

Untuk menggunakan penerapan biru/hijau untuk meningkatkan cluster yang Aurora Serverless v1 menjalankan Aurora MySQL versi 2 (MySQL 5.7—kompatibel)

1. Konversikan klaster DB Aurora Serverless v1 ke klaster Aurora MySQL versi 2 terprovisi. Ikuti prosedur di Mengonversi dari Aurora Serverless v1 menjadi terprovisi.
2. Buat penyebaran biru/hijau. Ikuti prosedur di Membuat deployment blue/green.
3. Pilih versi Aurora MySQL untuk cluster hijau yang kompatibel dengan, misalnya 3.04.1. Aurora Serverless v2

   Untuk versi yang kompatibel, lihat Aurora Serverless v2 dengan Aurora MySQL.
4. Ubah instance DB penulis dari cluster hijau untuk menggunakan kelas instans DB Serverless v2 (db.serverless).

   Untuk detailnya, lihat Mengonversi penulis atau pembaca terprovisi menjadi Aurora Serverless v2.

5. Saat kluster Aurora Serverless v2 DB Anda yang ditingkatkan tersedia, beralih dari cluster biru ke cluster hijau.

Klaster DB yang kompatibel dengan Aurora PostgreSQL

Anda dapat mengonversi cluster Aurora Serverless v1 DB Anda ke cluster DB yang disediakan, lalu menggunakan penerapan biru/hijau untuk memutakhirkannya dan mengubahnya menjadi cluster DB. Aurora Serverless v2 Kami merekomendasikan prosedur ini untuk lingkungan produksi. Untuk informasi selengkapnya, lihat Menggunakan Deployment Blue/Green Amazon RDS untuk pembaruan basis data.

Untuk menggunakan penerapan biru/hijau untuk meningkatkan cluster yang Aurora Serverless v1 menjalankan Aurora PostgreSQL versi 11

1. Konversi klaster DB Aurora Serverless v1 ke klaster Aurora PostgreSQL terprovisi. Ikuti prosedur dalam Mengonversi dari Aurora Serverless v1 menjadi terprovisi.
2. Buat penyebaran biru/hijau. Ikuti prosedur di Membuat deployment blue/green.
3. Pilih versi Aurora PostgreSQL untuk cluster hijau yang kompatibel dengan, misalnya 15.3. Aurora Serverless v2

   Untuk versi yang kompatibel, lihat Aurora Serverless v2 dengan Aurora PostgreSQL.
4. Ubah instance DB penulis dari cluster hijau untuk menggunakan kelas instans DB Serverless v2 (db.serverless).

   Untuk detailnya, lihat Mengonversi penulis atau pembaca terprovisi menjadi Aurora Serverless v2.
5. Saat kluster Aurora Serverless v2 DB Anda yang ditingkatkan tersedia, beralih dari cluster biru ke cluster hijau.

Anda juga dapat memutakhirkan cluster Aurora Serverless v1 DB Anda langsung dari Aurora PostgreSQL versi 11 ke versi 13, mengubahnya menjadi cluster DB yang disediakan, dan kemudian mengonversi cluster yang disediakan menjadi cluster DB. Aurora Serverless v2

Untuk memutakhirkan, lalu konversi Aurora Serverless v1 cluster yang menjalankan Aurora PostgreSQL versi 11

1. Upgrade Aurora Serverless v1 cluster ke Aurora PostgreSQL versi 13 versi yang kompatibel dengan, misalnya, 13.12. Aurora Serverless v2 Ikuti prosedur dalam Meningkatkan versi mayor.

   Untuk versi yang kompatibel, lihat Aurora Serverless v2 dengan Aurora PostgreSQL.
2. Konversi klaster DB Aurora Serverless v1 ke klaster Aurora PostgreSQL terprovisi. Ikuti prosedur dalam Mengonversi dari Aurora Serverless v1 menjadi terprovisi.
3. Tambahkan instance DB Aurora Serverless v2 pembaca ke cluster. Untuk informasi selengkapnya, lihat Menambahkan pembaca Aurora Serverless v2.
4. Gagal ke instans Aurora Serverless v2 DB:

   a. Pilih instance DB penulis dari cluster DB.

   b. Untuk Tindakan, pilih Failover.

   c. Pada halaman konfirmasi, pilih Failover.

Untuk cluster Aurora Serverless v1 DB yang menjalankan Aurora PostgreSQL versi 13, Anda Aurora Serverless v1 mengonversi cluster menjadi cluster DB yang disediakan, dan kemudian mengonversi cluster yang disediakan menjadi cluster DB. Aurora Serverless v2

Untuk meng-upgrade klaster Aurora Serverless v1 yang menjalankan Aurora PostgreSQL versi 13

1. Konversi klaster DB Aurora Serverless v1 ke klaster Aurora PostgreSQL terprovisi. Ikuti prosedur dalam Mengonversi dari Aurora Serverless v1 menjadi terprovisi.
2. Tambahkan instance DB Aurora Serverless v2 pembaca ke cluster. Untuk informasi selengkapnya, lihat Menambahkan pembaca Aurora Serverless v2.
3. Gagal ke instans Aurora Serverless v2 DB:

   a. Pilih instance DB penulis dari cluster DB.

   b. Untuk Tindakan, pilih Failover.

   c. Pada halaman konfirmasi, pilih Failover.

Bermigrasi dari basis data on-premise ke Aurora Serverless v2

Anda dapat memigrasikan basis data on-premise Anda ke Aurora Serverless v2, seperti halnya dengan Aurora MySQL dan Aurora PostgreSQL terprovisi.

- Untuk basis data MySQL, Anda dapat menggunakan perintah `mysqldump`. Untuk informasi selengkapnya, lihat Mengimpor data ke instans DB MySQL atau MariaDB dengan waktu henti yang lebih singkat.
- Untuk basis data PostgreSQL, Anda dapat menggunakan perintah `pg_dump` dan `pg_restore`. Untuk informasi selengkapnya, lihat postingan blog Praktik terbaik untuk memigrasi basis data PostgreSQL ke Amazon RDS dan Amazon Aurora.

Menggunakan Amazon Aurora Serverless v1

Amazon Aurora Serverless v1 (Amazon Aurora Nirserver versi 1) adalah konfigurasi penskalaan otomatis sesuai permintaan untuk Amazon Aurora. DB klaster Aurora Serverless v1 adalah klaster DB yang meningkatkan dan menurunkan kapasitas komputasi berdasarkan kebutuhan aplikasi Anda. Hal ini berbeda dari klaster DB terprovisi Aurora, dengan kapasitas yang Anda kelola secara manual. Aurora Serverless v1 menyediakan opsi yang relatif sederhana dan hemat biaya untuk beban kerja yang jarang, intermiten, atau tidak dapat diprediksi. Konfigurasi ini hemat biaya karena secara otomatis dimulai, menskalakan kapasitas komputasi sesuai dengan penggunaan aplikasi Anda, dan dinonaktifkan saat tidak digunakan.

Untuk mempelajari selengkapnya tentang harga, lihat Harga Serverless di bagian MySQL-Compatible Edition atau PostgreSQL-Compatible Edition di halaman Amazon Aurora pricing.

Klaster Aurora Serverless v1 memiliki jenis volume penyimpanan yang berkapasitas tinggi, terdistribusi, dan memiliki ketersediaan tinggi yang sama dengan yang digunakan oleh klaster DB terprovisi.

Untuk klaster Aurora Serverless v2, Anda dapat memilih apakah akan mengenkripsi volume klaster.

Untuk klaster Aurora Serverless v1, volume klaster selalu dienkripsi. Anda dapat memilih kunci enkripsi, tetapi tidak dapat menonaktifkan enkripsi. Artinya, Anda dapat melakukan operasi yang sama pada Aurora Serverless v1 seperti pada snapshot terenkripsi. Untuk informasi selengkapnya, lihat Aurora Serverless v1 dan snapshot.

Topik

- Ketersediaan wilayah dan versi
- Keuntungan Aurora Serverless v1
- Kasus penggunaan untuk Aurora Serverless v1
- Batasan Aurora Serverless v1
- Persyaratan konfigurasi untuk Aurora Serverless v1
- Menggunakan TLS/SSL dengan Aurora Serverless v1
- Cara kerja Aurora Serverless v1
- Membuat klaster DB Aurora Serverless v1
- Memulihkan klaster DB Aurora Serverless v1

- Memodifikasi klaster DB Aurora Serverless v1
- Menskalakan kapasitas klaster DB Aurora Serverless v1 secara manual
- Melihat klaster DB Aurora Serverless v1
- Menghapus klaster DB Aurora Serverless v1
- Aurora Serverless v1 dan versi mesin basis data Aurora

> **Important**
>
> Aurora memiliki dua generasi teknologi nirserver, Aurora Serverless v2 dan Aurora Serverless v1. Jika aplikasi Anda dapat berjalan di MySQL 8.0 atau PostgreSQL 13, kami sarankan Anda menggunakan Aurora Serverless v2. Aurora Serverless v2 diskalakan lebih cepat dan dengan cara yang lebih granular. Aurora Serverless v2 juga memiliki lebih banyak kompatibilitas dengan fitur Aurora lainnya seperti instans DB pembaca. Jadi, jika Anda sudah terbiasa dengan Aurora, Anda tidak perlu mempelajari banyak prosedur atau batasan baru untuk menggunakan Aurora Serverless v2 dan juga Aurora Serverless v1.
> Anda dapat mempelajari tentang Aurora Serverless v2 dalam Menggunakan Aurora Serverless v2.

Ketersediaan wilayah dan versi

Ketersediaan fitur dan dukungan bervariasi di seluruh versi spesifik dari setiap mesin basis data Aurora, dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang versi dan ketersediaan Wilayah dengan Aurora dan Aurora Serverless v1, lihat Aurora Serverless v1.

Keuntungan Aurora Serverless v1

Aurora Serverless v1 memberikan keuntungan berikut:

- Lebih sederhana dari terprovisi – Aurora Serverless v1 menghilangkan banyak kerumitan dalam mengelola instans dan kapasitas DB.
- Dapat diskalakan – Aurora Serverless v1 menskalakan kapasitas komputasi dan memori dengan lancar sesuai kebutuhan, tanpa gangguan pada koneksi klien.
- Hemat biaya – Saat menggunakan Aurora Serverless v1, Anda hanya membayar sumber daya basis data yang Anda konsumsi, per detik.

- Penyimpanan yang memiliki ketersediaan tinggi – Aurora Serverless v1 menggunakan sistem penyimpanan terdistribusi dengan toleransi kesalahan yang sama dengan replikasi enam arah seperti Aurora untuk melindungi dari kehilangan data.

Kasus penggunaan untuk Aurora Serverless v1

Aurora Serverless v1 dirancang untuk kasus penggunaan berikut:

- Aplikasi yang jarang digunakan – Anda memiliki aplikasi yang hanya digunakan selama beberapa menit beberapa kali per hari atau minggu, seperti situs blog volume rendah. Dengan Aurora Serverless v1, Anda hanya membayar sumber daya basis data yang Anda konsumsi per detik.
- Aplikasi baru – Anda men-deploy aplikasi baru dan Anda tidak yakin dengan ukuran instans yang Anda butuhkan. Dengan Aurora Serverless v1, Anda dapat membuat titik akhir basis data dan meminta basis data melakukan penskalaan otomatis berdasarkan persyaratan kapasitas aplikasi Anda.
- Beban kerja variabel – Anda menjalankan aplikasi yang jarang digunakan, dengan beban puncak selama 30 menit hingga beberapa jam beberapa kali setiap hari, atau beberapa kali per tahun. Contohnya adalah aplikasi untuk sumber daya manusia, penganggaran, dan aplikasi pelaporan operasional. Dengan Aurora Serverless v1, Anda tidak perlu lagi menyediakan kapasitas puncak atau rata-rata.
- Beban kerja yang tidak dapat diprediksi – Anda menjalankan beban kerja harian yang mengalami peningkatan aktivitas secara tiba-tiba dan tidak dapat diprediksi. Contohnya adalah situs info lalu lintas yang mengalami lonjakan aktivitas saat hujan mulai turun. Dengan Aurora Serverless v1, kapasitas basis data Anda diskalakan secara otomatis untuk memenuhi kebutuhan beban puncak aplikasi dan menurunkan skala kembali saat lonjakan aktivitas berakhir.
- Basis data pengembangan dan pengujian – Developer Anda menggunakan basis data selama jam kerja, tetapi tidak memerlukannya pada malam hari atau akhir pekan. Dengan Aurora Serverless v1, basis data Anda secara otomatis dinonaktifkan saat tidak digunakan.
- Aplikasi multi-penghuni – Dengan Aurora Serverless v1, Anda tidak perlu mengelola kapasitas basis data satu per satu untuk setiap aplikasi dalam armada Anda. Aurora Serverless v1 akan mengelola kapasitas basis data individu untuk Anda.

Batasan Aurora Serverless v1

Batasan berikut berlaku untuk Aurora Serverless v1:

- Aurora Serverless v1 tidak mendukung fitur-fitur berikut:
  - Basis data global Aurora
  - Replika Aurora
  - Autentikasi basis data AWS Identity and Access Management (IAM)
  - Pelacakan mundur di Aurora
  - Stream aktivitas basis data
  - Autentikasi Kerberos
  - Wawasan Performa
  - Proksi RDS
  - Melihat log di AWS Management Console
- Koneksi ke klaster DB Aurora Serverless v1 tertutup secara otomatis jika dibiarkan terbuka selama lebih dari satu hari.
- Semua klaster DB Aurora Serverless v1 memiliki batasan berikut:
  - Anda tidak dapat mengekspor snapshot Aurora Serverless v1 ke bucket Amazon S3.
  - Anda tidak dapat menggunakan AWS Database Migration Service dan Change Data Capture (CDC) dengan klaster DB Aurora Serverless v1. Hanya klaster DB Aurora terprovisi yang mendukung CDC dengan AWS DMS sebagai sumber.
  - Anda tidak dapat menyimpan data ke file teks di Amazon S3 atau memuat data file teks ke Aurora Serverless v1 dari S3.
  - Anda tidak dapat melampirkan peran IAM ke klaster DB Aurora Serverless v1. Namun, Anda dapat memuat data ke Aurora Serverless v1 dari Amazon S3 menggunakan ekstensi `aws_s3` dengan fungsi `aws_s3.table_import_from_s3` dan parameter `credentials`. Untuk informasi selengkapnya, lihat Mengimpor data dari Amazon S3 ke klaster DB Aurora PostgreSQL.
  - Saat menggunakan editor kueri, rahasia Secrets Manager dibuat untuk kredensial DB untuk mengakses basis data. Jika Anda menghapus kredensial dari editor kueri, rahasia terkait juga dihapus dari Secrets Manager. Anda tidak dapat memulihkan rahasia ini setelah dihapus.
- Klaster DB berbasis Aurora MySQL yang menjalankan Aurora Serverless v1 tidak mendukung hal berikut:
  - Menginvokasi fungsi AWS Lambda dari dalam klaster DB Aurora MySQL Anda. Namun, fungsi AWS Lambda dapat membuat panggilan ke klaster DB Aurora Serverless v1 Anda.
  - Memulihkan snapshot dari instans DB yang bukan merupakan instans DB Aurora MySQL atau RDS for MySQL.

- Mereplikasi data menggunakan replikasi berdasarkan log biner (binlog). Batasan ini berlaku terlepas dari apakah klaster DB Aurora Serverless v1 berbasis Aurora MySQL Anda adalah sumber atau target replikasi. Untuk mereplikasi data ke klaster DB Aurora Serverless v1 dari instans DB MySQL di luar Aurora, seperti yang berjalan di Amazon EC2, pertimbangkan untuk menggunakan AWS Database Migration Service. Untuk informasi selengkapnya, lihat Panduan Pengguna AWS Database Migration Service.
- Membuat pengguna dengan akses berbasis host (`'username'@'IP_address'`). Hal ini karena Aurora Serverless v1 menggunakan armada router antara host klien dan basis data untuk penskalaan yang mulus. Alamat IP yang dilihat klaster DB Aurora Serverless adalah alamat host router dan bukan klien Anda. Untuk informasi selengkapnya, lihat Arsitektur Aurora Serverless v1.

  Sebagai gantinya, gunakan wildcard (`'username'@'%'`).

- Klaster DB berbasis Aurora PostgreSQL yang menjalankan Aurora Serverless v1 memiliki batasan berikut:
  - Manajemen rencana kueri Aurora PostgreSQL (ekstensi `apg_plan_management`) tidak didukung.
  - Fitur replikasi logis yang tersedia di Amazon RDS PostgreSQL dan Aurora PostgreSQL tidak didukung.
  - Komunikasi keluar seperti yang dimungkinkan oleh ekstensi Amazon RDS for PostgreSQL tidak didukung. Misalnya, Anda tidak dapat mengakses data eksternal dengan ekstensi `postgres_fdw/dblink`. Untuk informasi selengkapnya tentang ekstensi RDS PostgreSQL, lihat PostgreSQL di Amazon RDS dalam Panduan Pengguna RDS.
  - Saat ini, kueri dan perintah SQL tertentu tidak disarankan. Hal ini termasuk kunci advisory tingkat sesi, relasi sementara, notifikasi asinkron (`LISTEN`), dan kursor dengan penahanan (`DECLARE name ... CURSOR WITH HOLD FOR query`). Selain itu, perintah `NOTIFY` akan mencegah penskalaan dan tidak disarankan.

    Untuk informasi selengkapnya, lihat Penskalaan otomatis untuk Aurora Serverless v1.

- Anda tidak dapat mengatur periode pencadangan otomatis yang diinginkan untuk klaster DB Aurora Serverless v1.
- Anda dapat mengatur periode pemeliharaan untuk klaster DB Aurora Serverless v1. Untuk informasi selengkapnya, lihat Menyesuaikan periode pemeliharaan klaster DB yang dinginkan.

Persyaratan konfigurasi untuk Aurora Serverless v1

Saat Anda membuat klaster DB Aurora Serverless v1, pastikan Anda memperhatikan persyaratan berikut:

- Gunakan nomor port spesifik ini untuk setiap mesin DB:
  - Aurora MySQL – `3306`
  - Aurora PostgreSQL – `5432`
- Buat klaster DB Aurora Serverless v1 Anda dalam cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. Saat membuat klaster DB Aurora Serverless v1 di VPC, Anda mengonsumsi dua (2) dari lima puluh (50) titik akhir Penyeimbang Beban Antarmuka dan Gateway yang dialokasikan untuk VPC Anda. Titik akhir ini dibuat secara otomatis untuk Anda. Untuk menambah kuota, Anda dapat menghubungi AWS Support. Untuk informasi selengkapnya, lihat kuota Amazon VPC.
- Anda tidak dapat memberikan alamat IP publik untuk klaster DB Aurora Serverless v1. Anda dapat mengakses klaster DB Aurora Serverless v1 hanya dari dalam VPC.
- Buat subnet di Zona Ketersediaan yang berbeda-beda untuk grup subnet DB yang Anda gunakan untuk klaster DB Aurora Serverless v1 Anda. Dengan kata lain, Anda tidak dapat memiliki lebih dari satu subnet di Zona Ketersediaan yang sama.
- Perubahan pada grup subnet yang digunakan oleh klaster DB Aurora Serverless v1 tidak diterapkan pada klaster.
- Anda dapat mengakses klaster DB Aurora Serverless v1 dari AWS Lambda. Untuk melakukannya, Anda harus mengonfigurasi fungsi Lambda agar berjalan dalam VPC yang sama seperti klaster DB Aurora Serverless v1 Anda. Untuk informasi selengkapnya tentang menggunakan AWS Lambda, lihat Mengonfigurasi fungsi Lambda untuk mengakses sumber daya di Amazon VPC dalam Panduan Developer AWS Lambda.

Menggunakan TLS/SSL dengan Aurora Serverless v1

Secara default, Aurora Serverless v1 menggunakan protokol Keamanan Lapisan Pengangkutan/Lapisan Soket Aman (TLS/SSL) untuk mengenkripsi komunikasi antara klien dan klaster DB Aurora Serverless v1 Anda. Layanan ini mendukung protokol TLS/SSL versi 1.0, 1.1, dan 1.2. Anda tidak perlu mengonfigurasi klaster DB Aurora Serverless v1 Anda untuk menggunakan TLS/SSL.

Namun, batasan berikut berlaku:

- Dukungan TLS/SSL untuk klaster DB Aurora Serverless v1 saat ini tidak tersedia di Wilayah AWS Tiongkok (Beijing).
- Ketika Anda membuat pengguna basis data untuk klaster DB Aurora Serverless v1 berbasis Aurora MySQL, jangan gunakan klausa `REQUIRE` untuk izin SSL. Tindakan ini akan mencegah pengguna terhubung ke instans Aurora DB.
- Untuk utilitas Klien MySQL dan Klien PostgreSQL, variabel sesi yang dapat Anda gunakan di lingkungan lain tidak berpengaruh ketika menggunakan TLS/SSL antara klien dan Aurora Serverless v1.
- Untuk Klien MySQL, ketika menghubungkan dengan mode `VERIFY_IDENTITY` TLS/SSL, saat ini Anda perlu menggunakan perintah `mysql` yang kompatibel dengan MySQL 8.0. Untuk informasi selengkapnya, lihat Menghubungkan ke instans DB yang menjalankan mesin basis data MySQL.

Bergantung pada klien yang Anda gunakan untuk terhubung ke klaster DB Aurora Serverless v1, Anda mungkin tidak perlu menetapkan TLS/SSL untuk mendapatkan koneksi terenkripsi. Sebagai contoh, untuk menggunakan Klien PostgreSQL agar terhubung ke klaster DB Aurora Serverless v1 yang menjalankan Aurora PostgreSQL-Compatible Edition, hubungkan seperti biasanya.

```
psql -h endpoint -U user
```

Setelah Anda memasukkan kata sandi, Klien PostgreSQL akan menunjukkan detail koneksi, termasuk versi dan cipher TLS/SSL.

```
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1), server 10.12)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
 compression: off)
Type "help" for help.
```

Important

Aurora Serverless v1 menggunakan protokol Keamanan Lapisan Pengangkutan/Lapisan Soket Aman (TLS/SSL) untuk mengenkripsi koneksi secara default kecuali jika SSL/TLS dinonaktifkan oleh aplikasi klien. Koneksi TLS/SSL diterminasi pada armada router. Komunikasi antara armada router dan klaster DB Aurora Serverless v1 Anda terjadi dalam batas jaringan internal layanan.

Anda dapat memeriksa status koneksi klien untuk mengetahui apakah koneksi ke Aurora Serverless v1 dienkripsi TLS/SSL. Tabel PostgreSQL `pg_stat_ssl` dan

`pg_stat_activity` serta fungsi `ssl_is_used`-nya tidak menunjukkan status TLS/SSL untuk komunikasi antara aplikasi klien dan Aurora Serverless v1. Demikian pula, status TLS/SSL tidak dapat diperoleh dari pernyataan `status` MySQL.
Parameter klaster Aurora `force_ssl` untuk PostgreSQL dan `require_secure_transport` untuk MySQL sebelumnya tidak didukung untuk Aurora Serverless v1. Parameter ini sekarang tersedia untuk Aurora Serverless v1. Untuk daftar lengkap parameter yang didukung oleh Aurora Serverless v1, hubungi operasi API. DescribeEngineDefaultClusterParameters Untuk informasi tentang grup parameter dan Aurora Nirserver v1, lihat Grup parameter untuk Aurora Serverless v1.

Untuk menggunakan Klien MySQL agar terhubung ke klaster DB Aurora Serverless v1 yang menjalankan Aurora MySQL-Compatible Edition, Anda menetapkan TLS/SSL dalam permintaan Anda. Contoh berikut mencakup trust store Amazon root CA 1 yang diunduh dari Amazon Trust Services, yang diperlukan agar koneksi ini berhasil.

```
mysql -h endpoint -P 3306 -u user -p --ssl-ca=amazon-root-CA-1.pem --ssl-mode=REQUIRED
```

Saat diminta, masukkan kata sandi Anda. Kemudian, monitor MySQL terbuka. Anda dapat mengonfirmasi bahwa sesi dienkripsi menggunakan perintah `status`.

```
mysql> status
--------------
mysql  Ver 14.14 Distrib 5.5.62, for Linux (x86_64) using readline 5.1
Connection id:          19
Current database:
Current user:           ***@*******
SSL:                    Cipher in use is ECDHE-RSA-AES256-SHA
...
```

Untuk mempelajari selengkapnya tentang menghubungkan ke basis data Aurora MySQL dengan Klien MySQL, lihat Menghubungkan ke instans DB yang menjalankan mesin basis data MySQL.

Aurora Serverless v1 mendukung semua mode TLS/SSL yang tersedia untuk Klien MySQL (`mysql`) dan Klien PostgreSQL (`psql`), termasuk yang tercantum dalam tabel berikut.

Deskripsi mode TLS/SSL	mysql	psql
Hubungkan tanpa menggunakan TLS/SSL.	DISABLED	disable
Mencoba koneksi menggunakan TLS/SSL terlebih dahulu, tetapi kembali ke non-SSL jika diperlukan.	PREFERRED	prefer (default)
Memberlakukan penggunaan TLS/SSL.	REQUIRED	require
memberlakukan TLS/SSL dan memverifikasi CA.	VERIFY_CA	verify-ca
Memberlakukan TLS/SSL, memverifikasi CA, dan memverifikasi nama host CA.	VERIFY_IDENTITY	verify-full

Aurora Serverless v1 menggunakan sertifikat wildcard. Jika Anda menentukan opsi "memverifikasi CA" atau "memverifikasi CA dan nama host CA" saat menggunakan TLS/SSL, pertama-tama unduh trust store Amazon root CA 1 dari Amazon Trust Services. Setelah melakukannya, Anda dapat mengidentifikasi file berformat PEM ini dalam perintah klien Anda. Untuk melakukannya menggunakan Klien PostgreSQL:

Untuk Linux, macOS, atau Unix:

```
psql 'host=endpoint user=user sslmode=require sslrootcert=amazon-root-CA-1.pem
 dbname=db-name'
```

Untuk mempelajari selengkapnya tentang mengoperasikan basis data Aurora PostgreSQL menggunakan Klien Postgres, lihat Menghubungkan ke instans DB yang menjalankan mesin basis data PostgreSQL.

Untuk informasi selengkapnya tentang menghubungkan ke klaster DB Aurora secara umum, lihat Menghubungkan ke klaster DB Amazon Aurora.

Cipher suite yang didukung untuk koneksi ke klaster DB Aurora Serverless v1

Dengan menggunakan cipher suite yang dapat dikonfigurasi, Anda dapat memiliki kontrol lebih besar atas keamanan koneksi basis data Anda. Anda dapat menentukan daftar cipher suite yang ingin Anda izinkan untuk mengamankan koneksi TLS/SSL klien ke basis data Anda. Dengan cipher suite yang dapat dikonfigurasi, Anda dapat mengontrol enkripsi koneksi yang diterima server basis data Anda. Tindakan ini mencegah penggunaan cipher yang tidak aman atau yang sudah ditiadakan.

Klaster DB Aurora Serverless v1 yang didasarkan pada Aurora MySQL mendukung cipher suite yang sama dengan klaster DB terprovisi Aurora MySQL. Untuk informasi tentang cipher suite ini, lihat Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora MySQL.

Klaster DB Aurora Serverless v1 yang didasarkan pada Aurora PostgreSQL tidak mendukung cipher suite.

Cara kerja Aurora Serverless v1

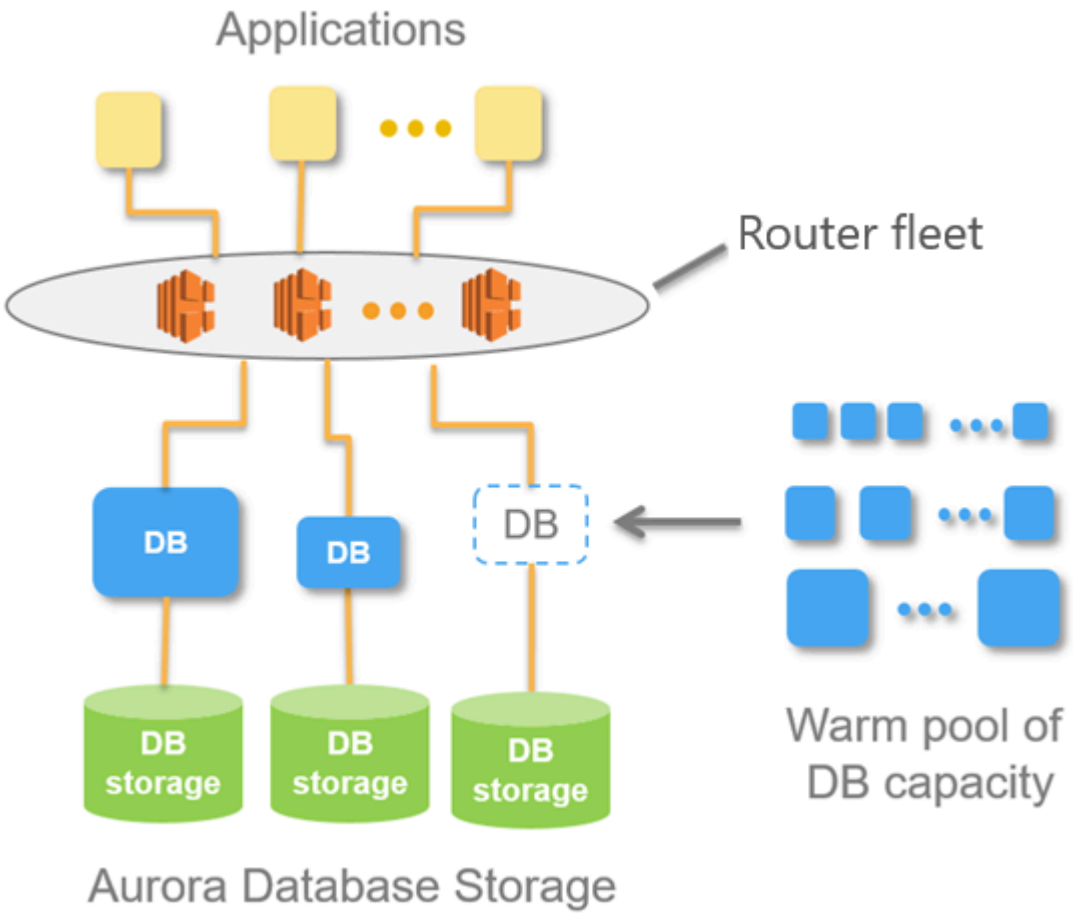
Setelahnya, Anda dapat mempelajari cara kerja Aurora Serverless v1.

Topik

- Arsitektur Aurora Serverless v1
- Penskalaan otomatis untuk Aurora Serverless v1
- Tindakan batas waktu habis untuk perubahan kapasitas
- Jeda dan lanjutkan untuk Aurora Serverless v1
- Mengatur jumlah maksimum koneksi basis data untuk Aurora Serverless v1
- Grup parameter untuk Aurora Serverless v1
- Logging untuk Aurora Serverless v1
- Aurora Serverless v1 dan pemeliharaan
- Aurora Serverless v1 dan failover
- Aurora Serverless v1 dan snapshot

Arsitektur Aurora Serverless v1

Gambar berikut menunjukkan gambaran umum arsitektur Aurora Serverless v1.



Alih-alih menyediakan dan mengelola server basis data, Anda menentukan unit kapasitas Aurora (ACU). Setiap ACU adalah kombinasi dari sekitar 2 gigabyte (GB) memori, CPU yang sesuai, dan jaringan. Penyimpanan basis data secara otomatis menskalakan dari 10 gibibyte (GiB) hingga 128 tebibyte (TiB), sama seperti penyimpanan dalam klaster DB Aurora standar.

Anda dapat menentukan ACU minimum dan maksimum. Unit kapasitas Aurora minimum adalah ACU terendah yang dapat dicapai saat menurunkan skala klaster DB. Unit kapasitas Aurora maksimum adalah ACU tertinggi yang dapat dicapai saat menaikkan skala klaster DB. Berdasarkan pengaturan Anda, Aurora Serverless v1 secara otomatis membuat aturan penskalaan untuk ambang batas pemanfaatan CPU, koneksi, dan memori yang tersedia.

Aurora Serverless v1 mengelola kumpulan sumber daya aktif di Wilayah AWS untuk meminimalkan waktu penskalaan. Saat Aurora Serverless v1 menambahkan sumber daya baru ke klaster DB Aurora, layanan ini menggunakan armada router untuk mengalihkan koneksi klien aktif ke sumber daya baru. Pada waktu tertentu, Anda hanya dikenai biaya untuk ACU yang secara aktif digunakan dalam klaster DB Aurora Anda.

Penskalaan otomatis untuk Aurora Serverless v1

Kapasitas yang dialokasikan ke klaster DB Aurora Serverless v1 akan dinaikkan dan diturunkan skalakan dengan lancar berdasarkan beban yang dihasilkan oleh aplikasi klien Anda. Di sini, beban adalah pemanfaatan CPU dan jumlah koneksi. Ketika kapasitas dibatasi oleh salah satu dari hal ini, Aurora Serverless v1 menaikkan skala. Aurora Serverless v1 juga menaikkan skala ketika mendeteksi masalah performa yang dapat diatasi dengan penaikan skala.

Anda dapat melihat peristiwa penskalaan untuk klaster Aurora Serverless v1 Anda di AWS Management Console. Selama penskalaan otomatis, Aurora Serverless v1 mengatur ulang metrik `EngineUptime`. Nilai metrik pengaturan ulang ini tidak berarti bahwa penskalaan yang lancar mengalami masalah atau Aurora Serverless v1 memutus koneksi. Ini hanyalah titik awal untuk waktu aktif pada kapasitas baru. Untuk mempelajari selengkapnya tentang metrik, lihat Memantau metrik di klaster Amazon Aurora.

Jika tidak memiliki koneksi aktif, klaster DB Aurora Serverless v1 Anda dapat menurunkan skalanya ke kapasitas nol (0 ACU). Untuk mempelajari selengkapnya, lihat Jeda dan lanjutkan untuk Aurora Serverless v1.

Jika memang perlu melakukan operasi penskalaan, Aurora Serverless v1 pertama-tama mencoba mengidentifikasi titik penskalaan, waktu ketika tidak ada kueri yang sedang diproses. Aurora Serverless v1 mungkin tidak dapat menemukan titik penskalaan karena alasan berikut:

- Kueri yang berjalan lama
- Transaksi yang sedang berlangsung
- Tabel atau kunci tabel sementara

Untuk meningkatkan tingkat keberhasilan klaster DB Aurora Serverless v1 dalam menemukan titik penskalaan, kami sarankan Anda menghindari kueri dan transaksi yang berjalan lama. Untuk mempelajari selengkapnya tentang operasi yang memblokir penskalaan dan cara menghindarinya, lihat Praktik terbaik untuk menggunakan Aurora Serverless v1.

Secara default, Aurora Serverless v1 mencoba menemukan titik penskalaan selama 5 menit (300 detik). Anda dapat menentukan periode waktu tunggu yang berbeda saat Anda membuat atau memodifikasi klaster. Periode batas waktu bisa antara 60 detik dan 10 menit (600 detik). Jika Aurora Serverless v1 tidak dapat menemukan titik penskalaan dalam periode yang ditentukan, batas waktu operasi penskalaan otomatis habis.

Secara default, jika penskalaan otomatis tidak dapat menemukan titik penskalaan sebelum batas waktu habis, Aurora Serverless v1 akan mempertahankan klaster pada kapasitas saat ini. Anda dapat mengubah perilaku default ini saat Anda membuat atau memodifikasi klaster DB Aurora Serverless v1 dengan memilih opsi Paksa perubahan kapasitas. Untuk informasi selengkapnya, lihat Tindakan batas waktu habis untuk perubahan kapasitas.

Tindakan batas waktu habis untuk perubahan kapasitas

Jika batas waktu penskalaan otomatis habis tanpa menemukan titik penskalaan, Aurora akan secara default mempertahankan kapasitas saat ini. Anda dapat memilih agar Aurora memaksa perubahan dengan memilih opsi Paksa perubahan kapasitas. Opsi ini tersedia di bagian Waktu habis dan tindakan Penskalaan Otomatis pada halaman Buat basis data saat Anda membuat klaster.

Secara default, opsi Paksa perubahan kapasitas tidak dipilih. Jangan pilih opsi ini agar kapasitas klaster DB Aurora Serverless v1 Anda tetap tidak berubah jika waktu operasi penskalaan habis tanpa menemukan titik penskalaan.

Memilih opsi ini akan menyebabkan klaster DB Aurora Serverless v1 Anda menerapkan perubahan kapasitas, bahkan tanpa titik penskalaan. Sebelum memilih opsi ini, ketahui konsekuensinya:

- Setiap transaksi dalam proses akan terinterupsi, dan pesan kesalahan berikut akan muncul.

  Aurora MySQL versi 2 – ERROR 1105 (HY000): Transaksi terakhir dibatalkan karena Seamless Scaling. Silakan coba lagi.

  Anda dapat mengirim kembali transaksi segera setelah klaster DB Aurora Serverless v1 Anda tersedia.
- Koneksi ke tabel dan kunci sementara terputus.

  Kami menyarankan Anda memilih opsi Paksa perubahan kapasitas hanya jika aplikasi Anda dapat dipulihkan dari koneksi yang terputus atau transaksi yang tidak selesai.

Pilihan yang Anda buat di AWS Management Console saat membuat klaster DB Aurora Serverless v1 akan disimpan di objek `ScalingConfigurationInfo`, pada properti `SecondsBeforeTimeout` dan `TimeoutAction`. Nilai properti `TimeoutAction` diatur ke salah satu nilai berikut saat Anda membuat klaster:

- `RollbackCapacityChange` – Nilai ini diatur saat Anda memilih opsi Kembalikan perubahan kapasitas. Ini adalah perilaku default.

- `ForceApplyCapacityChange` – Nilai ini diatur saat Anda memilih opsi Paksa perubahan kapasitas.

Anda bisa mendapatkan nilai properti ini pada cluster Aurora Serverless v1 DB yang ada dengan menggunakan describe-db-clustersAWS CLIperintah, seperti yang ditunjukkan berikut.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-db-clusters --region region \
  --db-cluster-identifier your-cluster-name \
  --query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'
```

Untuk Windows:

```
aws rds describe-db-clusters --region region ^
  --db-cluster-identifier your-cluster-name ^
  --query "*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}"
```

Misalnya, berikut ini adalah kueri dan respons untuk klaster DB Aurora Serverless v1 bernama `west-coast-sles` di Wilayah AS Barat (California Utara).

```
$ aws rds describe-db-clusters --region us-west-1 --db-cluster-identifier west-coast-
sles
--query '*[].{ScalingConfigurationInfo:ScalingConfigurationInfo}'

[
    {
        "ScalingConfigurationInfo": {
            "MinCapacity": 1,
            "MaxCapacity": 64,
            "AutoPause": false,
            "SecondsBeforeTimeout": 300,
            "SecondsUntilAutoPause": 300,
            "TimeoutAction": "RollbackCapacityChange"
        }
    }
]
```

Seperti ditunjukkan oleh responsnya, klaster DB Aurora Serverless v1 ini menggunakan pengaturan default.

Untuk informasi selengkapnya, lihat Membuat klaster DB Aurora Serverless v1. Setelah membuat Aurora Serverless v1, Anda juga dapat mengubah tindakan batas waktu habis dan pengaturan kapasitas lainnya kapan saja. Untuk mempelajari caranya, lihat Memodifikasi klaster DB Aurora Serverless v1.

Jeda dan lanjutkan untuk Aurora Serverless v1

Anda dapat memilih untuk menjeda klaster DB Aurora Serverless v1 setelah jumlah waktu tertentu tanpa aktivitas. Anda menentukan jumlah waktu tanpa aktivitas sebelum klaster DB dijeda. Saat Anda memilih opsi ini, waktu tanpa aktivitas default adalah lima menit, tetapi Anda dapat mengubah nilai ini. Ini adalah pengaturan opsional.

Ketika klaster DB dijeda, tidak ada aktivitas komputasi atau memori yang terjadi, dan Anda hanya dikenai biaya untuk penyimpanan. Jika koneksi basis data diminta saat klaster DB Aurora Serverless v1 dijeda, klaster DB secara otomatis beroperasi kembali dan melayani permintaan koneksi.

Saat klaster DB melanjutkan aktivitas, kapasitasnya sama dengan saat Aurora menjeda klaster tersebut. Jumlah ACU akan bergantung pada seberapa banyak Aurora menaikkan atau menurunkan skala klaster sebelum menjedanya.

> **Note**
>
> Jika klaster DB dijeda selama lebih dari tujuh hari, klaster DB ini mungkin akan dicadangkan dengan snapshot. Dalam kasus ini, Aurora memulihkan klaster DB dari snapshot ketika ada permintaan untuk terhubung ke klaster DB tersebut.

Mengatur jumlah maksimum koneksi basis data untuk Aurora Serverless v1

Contoh berikut ditujukan untuk klaster DB Aurora Serverless v1 yang kompatibel dengan MySQL 5.7. Anda dapat menggunakan klien MySQL atau editor kueri, jika aksesnya telah Anda konfigurasi. Untuk informasi selengkapnya, lihat Menjalankan kueri di editor kueri.

Untuk menemukan jumlah maksimum koneksi basis data

1. Temukan rentang kapasitas untuk klaster DB Aurora Serverless v1 Anda menggunakan AWS CLI.

```
aws rds describe-db-clusters \
```

```
    --db-cluster-identifier my-serverless-57-cluster \
    --query 'DBClusters[*].ScalingConfigurationInfo|[0]'
```

Hasilnya menunjukkan bahwa rentang kapasitasnya adalah 1–4 ACU.

```
{
    "MinCapacity": 1,
    "AutoPause": true,
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

2. Jalankan kueri SQL berikut untuk menemukan jumlah maksimum koneksi.

```
select @@max_connections;
```

Hasil yang ditampilkan ditujukan untuk kapasitas minimum klaster, 1 ACU.

```
@@max_connections
90
```

3. Skalakan klaster menjadi 8–32 ACU.

Untuk informasi selengkapnya tentang penskalaan, lihat Memodifikasi klaster DB Aurora Serverless v1.

4. Konfirmasikan rentang kapasitas.

```
{
    "MinCapacity": 8,
    "AutoPause": true,
    "MaxCapacity": 32,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

5. Temukan jumlah maksimum koneksi.

```
select @@max_connections;
```

Hasil yang ditampilkan ditujukan untuk kapasitas minimum klaster, 8 ACU.

```
@@max_connections
1000
```

6. Skalakan klaster ke nilai maksimum yang mungkin, 256–256 ACU.
7. Konfirmasikan rentang kapasitas.

```
{
    "MinCapacity": 256,
    "AutoPause": true,
    "MaxCapacity": 256,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

8. Temukan jumlah maksimum koneksi.

```
select @@max_connections;
```

Hasil yang ditampilkan ditujukan untuk 256 ACU.

```
@@max_connections
6000
```

> **Note**
>
> Nilai`max_connections` tidak diskalakan secara linier dengan jumlah ACU.

9. Skalakan klaster kembali ke 1–4 ACU.

```
{
    "MinCapacity": 1,
    "AutoPause": true,
    "MaxCapacity": 4,
    "TimeoutAction": "RollbackCapacityChange",
    "SecondsUntilAutoPause": 3600
}
```

Kali ini, nilai `max_connections` ditujukan untuk 4 ACU.

```
@@max_connections
270
```

10. Biarkan klaster diturunkan skalanya menjadi 2 ACU.

    ```
    @@max_connections
    180
    ```

    Jika Anda telah mengonfigurasi klaster untuk dijeda setelah jumlah waktu idle tertentu, klaster akan diturunkan skalanya menjadi 0 ACU. Namun, `max_connections` tidak akan turun di bawah nilai untuk 1 ACU.

    ```
    @@max_connections
    90
    ```

Grup parameter untuk Aurora Serverless v1

Saat Anda membuat klaster DB Aurora Serverless v1, Anda memilih mesin Aurora DB tertentu dan grup parameter klaster DB terkait. Tidak seperti klaster DB Aurora terprovisi, klaster DB Aurora Serverless v1 memiliki satu instans DB baca/tulis tunggal yang dikonfigurasi hanya dengan grup parameter klaster DB—instans tersebut tidak memiliki grup parameter DB terpisah. Selama penskalaan otomatis, Aurora Serverless v1 harus dapat mengubah parameter agar klaster beroperasi dengan paling efektif untuk peningkatan atau penurunan kapasitas. Jadi, dengan klaster DB Aurora Serverless v1, beberapa perubahan yang dapat Anda terapkan pada parameter untuk jenis mesin DB tertentu mungkin tidak berlaku.

Misalnya, klaster DB Aurora Serverless v1 berbasis Aurora PostgreSQL tidak dapat menggunakan `apg_plan_mgmt.capture_plan_baselines` dan parameter lain yang mungkin digunakan pada \klaster DB Aurora PostgreSQL terprovisi untuk manajemen rencana kueri.

Anda bisa mendapatkan daftar nilai default untuk grup parameter default untuk berbagai mesin Aurora DB dengan menggunakan perintah describe-engine-default-cluster-parameter CLI dan menanyakan file. Wilayah AWS Berikut ini adalah nilai yang dapat Anda gunakan untuk opsi `--db-parameter-group-family`.

Aurora MySQL versi 2	`aurora-mysql5.7`

Aurora PostgreSQL versi 11	`aurora-postgresql11`
Aurora PostgreSQL versi 13	`aurora-postgresql13`

Kami menyarankan Anda mengonfigurasi AWS CLI dengan ID kunci akses AWS dan kunci akses rahasia AWS, serta mengatur Wilayah AWS sebelum menggunakan perintah AWS CLI. Dengan menyediakan Wilayah untuk konfigurasi CLI, Anda tidak perlu memasukkan parameter `--region` saat menjalankan perintah. Untuk mempelajari selengkapnya tentang mengonfigurasi AWS CLI, lihat Dasar-dasar konfigurasi dalam Panduan Pengguna AWS Command Line Interface.

Contoh berikut mendapatkan daftar parameter dari grup klaster DB default untuk Aurora MySQL versi 2.

Untuk Linux, macOS, atau Unix:

```
aws rds describe-engine-default-cluster-parameters \
  --db-parameter-group-family aurora-mysql5.7 --query \
  'EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, `serverless`) == `true`] | [*].{param:ParameterName}' \
  --output text
```

Untuk Windows:

```
aws rds describe-engine-default-cluster-parameters ^
   --db-parameter-group-family aurora-mysql5.7 --query ^
   "EngineDefaults.Parameters[*].
{ParameterName:ParameterName,SupportedEngineModes:SupportedEngineModes} | [?
contains(SupportedEngineModes, 'serverless') == `true`] | [*].{param:ParameterName}" ^
   --output text
```

Memodifikasi nilai parameter untuk Aurora Serverless v1

Seperti yang dijelaskan dalam Bekerja dengan grup parameter, Anda tidak dapat langsung mengubah nilai dalam grup parameter default, terlepas dari jenisnya (grup parameter klaster DB, grup parameter DB). Sebagai gantinya, Anda membuat grup parameter kustom berdasarkan grup parameter klaster DB default untuk mesin Aurora DB Anda dan mengubah pengaturan yang diperlukan pada grup parameter tersebut. Misalnya, Anda mungkin ingin mengubah beberapa

pengaturan untuk cluster Aurora Serverless v1 DB Anda untuk mencatat kueri atau mengunggah log khusus mesin DB ke Amazon CloudWatch.

Untuk membuat grup parameter klaster DB kustom

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Grup parameter.
3. Pilih Buat grup parameter untuk membuka panel detail Grup parameter.
4. Pilih grup klaster DB default yang sesuai untuk mesin DB yang ingin Anda gunakan untuk klaster DB Aurora Serverless v1. Pastikan Anda memilih opsi berikut:

    a. Untuk Rangkaian grup parameter, pilih rangkaian grup parameter yang sesuai untuk mesin DB pilihan Anda. Pastikan bahwa pilihan Anda memiliki awalan `aurora-` dalam namanya.

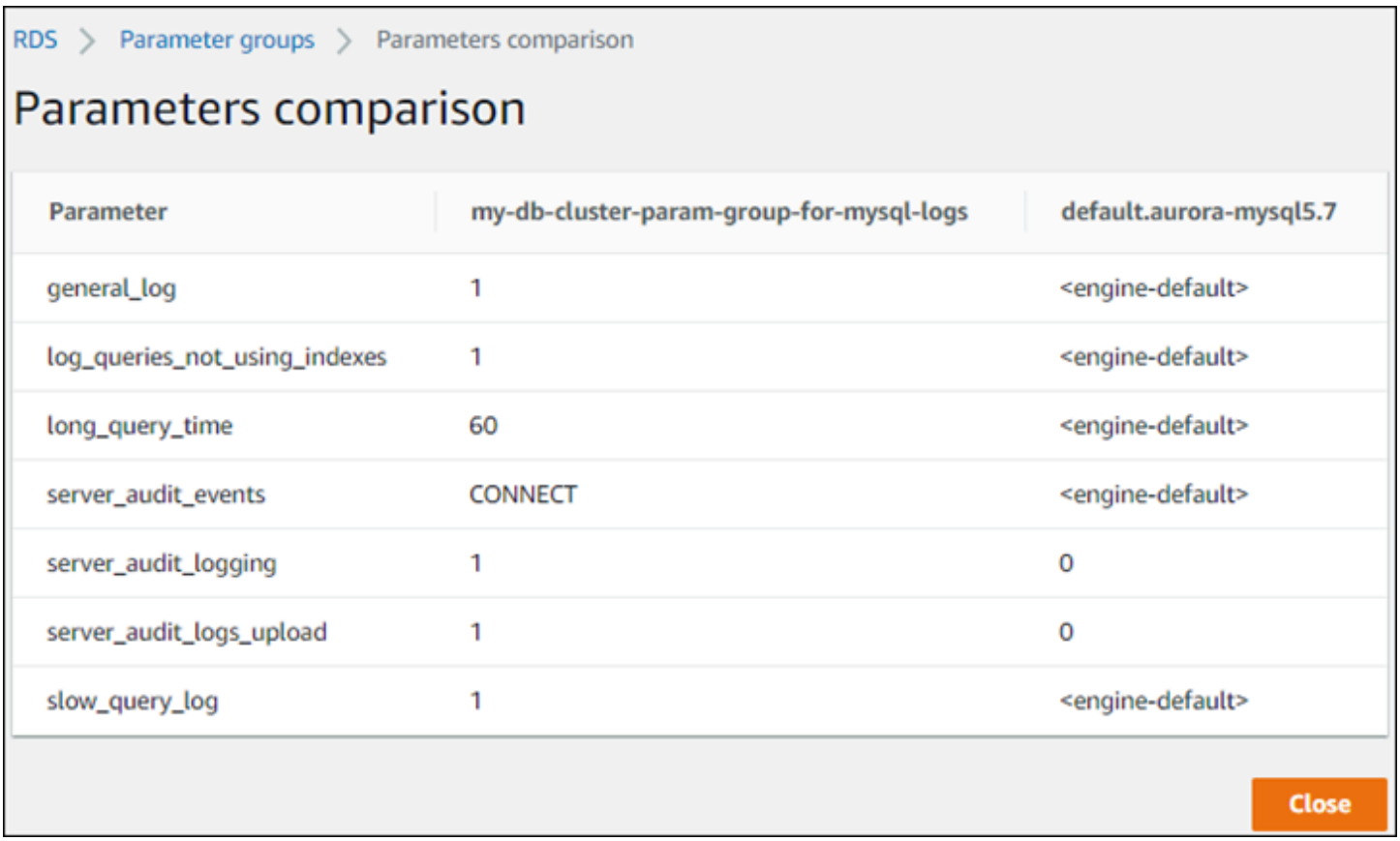
    b. Untuk Jenis, pilih Grup Parameter Klaster DB.

    c. Untuk Nama grup dan Deskripsi, masukkan nama yang bermakna untuk Anda atau orang lain yang mungkin perlu menangani klaster DB Aurora Serverless v1 Anda dan parameternya.

    d. Pilih Buat.

Grup parameter klaster DB kustom Anda ditambahkan ke daftar grup parameter yang tersedia di Wilayah AWS Anda. Anda dapat menggunakan grup parameter klaster DB kustom Anda saat membuat klaster DB Aurora Serverless v1 baru. Anda juga dapat memodifikasi klaster DB Aurora Serverless v1 yang ada untuk menggunakan grup parameter klaster DB kustom Anda. Setelah klaster DB Aurora Serverless v1 Anda dimulai menggunakan grup parameter klaster DB kustom, Anda dapat mengubah nilai untuk parameter dinamis menggunakan AWS Management Console atau AWS CLI.

Anda juga dapat menggunakan konsol untuk melihat side-by-side perbandingan nilai dalam grup parameter cluster DB kustom Anda dibandingkan dengan grup parameter cluster DB default, seperti yang ditunjukkan pada gambar berikut.



Jika Anda mengubah nilai parameter pada klaster DB aktif, Aurora Serverless v1 akan memulai penskalaan ke dalam yang konstans untuk menerapkan perubahan parameter. Jika klaster DB Aurora Serverless v1 Anda dalam keadaan "dijeda", klaster DB ini akan dilanjutkan dan memulai penskalaan agar dapat menerapkan perubahan. Operasi penskalaan untuk perubahan grup parameter selalu memaksa perubahan kapasitas, jadi ketahui bahwa memodifikasi parameter dapat mengakibatkan koneksi terputus jika titik penskalaan tidak dapat ditemukan selama periode penskalaan.

Logging untuk Aurora Serverless v1

Secara default, log kesalahan untuk Aurora Serverless v1 diaktifkan dan diunggah secara otomatis ke Amazon CloudWatch. Anda juga dapat meminta cluster Aurora Serverless v1 DB Anda mengunggah log khusus mesin database Aurora ke. CloudWatch Untuk melakukannya, aktifkan parameter konfigurasi di grup parameter klaster DB kustom Anda. Cluster Aurora Serverless v1 DB Anda kemudian mengunggah semua log yang tersedia ke Amazon CloudWatch. Pada titik ini, Anda dapat menggunakan CloudWatch untuk menganalisis data log, membuat alarm, dan melihat metrik.

Untuk Aurora MySQL, tabel berikut menunjukkan log yang dapat Anda aktifkan. Saat diaktifkan, mereka secara otomatis diunggah dari cluster Aurora Serverless v1 DB Anda ke Amazon CloudWatch.

Log Aurora MySQL	Deskripsi
`general_log`	Membuat log umum. Atur ke 1 untuk mengaktif kan. Default-nya adalah nonaktif (0).
`log_queries_not_using_indexes`	Mencatat setiap kueri ke log kueri lambat yang tidak menggunakan indeks. Default-nya adalah nonaktif (0). Atur ke 1 untuk mengaktifkan log ini.
`long_query_time`	Mencegah kueri yang berjalan cepat agar tidak dicatat dalam log kueri lambat. Dapat diatur ke angka float antara 0 dan 31536000. Default-nya adalah 0 (tidak aktif).
`server_audit_events`	Daftar peristiwa untuk dicatat dalam log. Nilai yang didukung adalah `CONNECT`, `QUERY`, `QUERY_DCL` , `QUERY_DDL` , `QUERY_DML` , dan `TABLE`.
`server_audit_logging`	Atur ke 1 untuk mengaktifkan logging audit server. Jika Anda mengaktifkan ini, Anda dapat menentukan peristiwa audit yang akan dikirim CloudWatch dengan mencantumkannya di `server_audit_events` parameter.
`slow_query_log`	Membuat log kueri lambat. Atur ke 1 untuk mengaktifkan log kueri lambat. Default-nya adalah nonaktif (0).

Untuk informasi selengkapnya, lihat Menggunakan Audit Lanjutan dengan klaster DB Amazon Aurora MySQL.

Untuk Aurora PostgreSQL, tabel berikut menunjukkan log yang dapat Anda aktifkan. Saat diaktifkan, mereka secara otomatis diunggah dari cluster Aurora Serverless v1 DB Anda ke Amazon CloudWatch bersama dengan log kesalahan biasa.

Log Aurora PostgreSQL	Deskripsi
`log_connections`	Diaktifkan secara default dan tidak dapat diubah. Log ini mencatat log detail untuk semua koneksi klien baru.
`log_disconnections`	Diaktifkan secara default dan tidak dapat diubah. Mencatat log semua pemutusan koneksi klien.
`log_hostname`	Dimatikan secara default dan tidak dapat diubah. Nama host tidak dicatat.
`log_lock_waits`	Default-nya adalah 0 (nonaktif). Atur ke 1 untuk mencatat log peristiwa tunggu kunci.
`log_min_duration_statement`	Durasi minimum (dalam milidetik) untuk menjalankan pernyataan sebelum dicatat lognya.
`log_min_messages`	Mengatur tingkat pesan yang dicatat lognya. Nilai yang didukung adalah `debug5`, `debug4`, `debug3`, `debug2`, `debug1`, `info`, `notice`, `warning`, `error`, `log`, `fatal`, `panic`. Untuk mencatat data performa ke log `postgres`, tetapkan nilainya ke `debug1`.
`log_temp_files`	Mencatat log penggunaan file sementara yang melampaui kilobyte (kB) yang ditentukan.
`log_statement`	Mengontrol pernyataan SQL tertentu yang dicatat lognya. Nilai yang didukung adalah `none`, `ddl`, `mod`, dan `all`. Default-nya adalah `none`.

Setelah Anda mengaktifkan log untuk Aurora MySQL atau Aurora PostgreSQL untuk cluster DB Anda, Anda dapat melihat log masuk. Aurora Serverless v1 CloudWatch

Melihat Aurora Serverless v1 log dengan Amazon CloudWatch

Aurora Serverless v1secara otomatis mengunggah ("menerbitkan") ke Amazon CloudWatch semua log yang diaktifkan di grup parameter cluster DB kustom Anda. Anda tidak perlu memilih atau menentukan jenis log. Mengunggah log dimulai segera setelah Anda mengaktifkan parameter konfigurasi log. Jika Anda menonaktifkan parameter log nanti, pengunggahan lainnya akan berhenti. Namun, semua log yang telah diterbitkan CloudWatch tetap ada sampai Anda menghapusnya.

Untuk informasi lebih lanjut tentang penggunaan CloudWatch dengan log Aurora MySQL, lihat. Memantau peristiwa log di Amazon CloudWatch

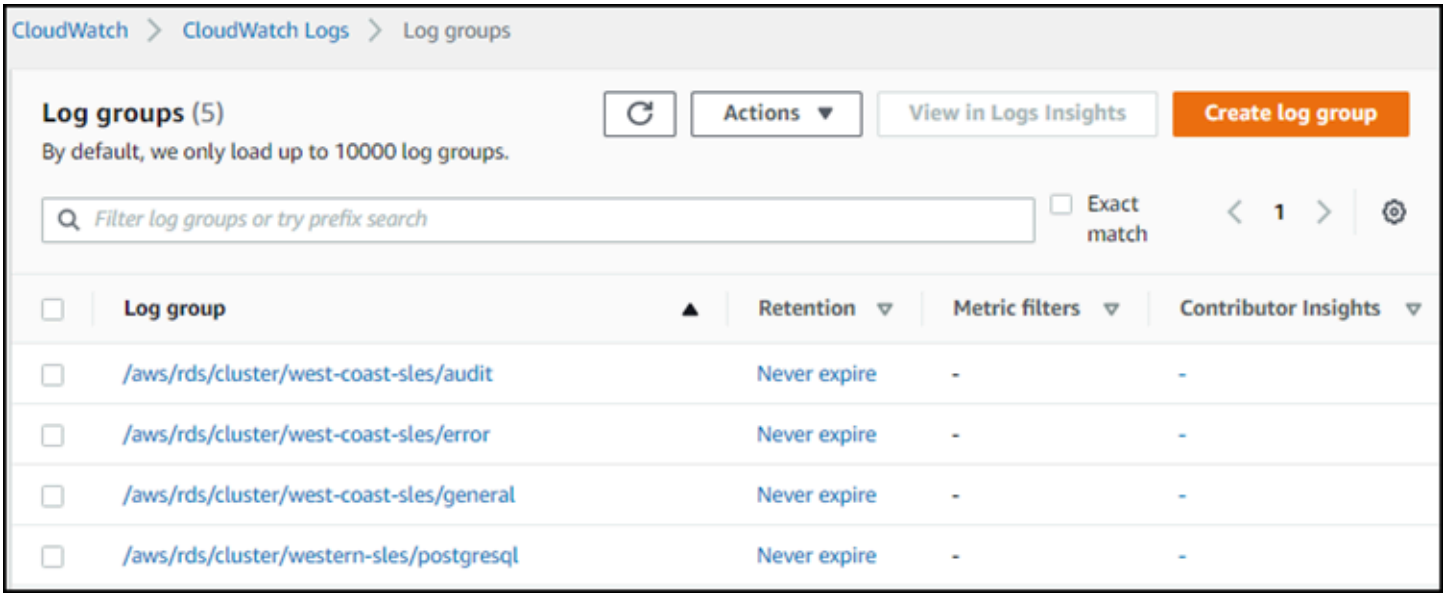
Untuk informasi lebih lanjut tentang CloudWatch dan Aurora PostgreSQL, lihat. Menerbitkan log Aurora PostgreSQL ke Amazon Logs CloudWatch

Untuk melihat log untuk klaster DB Aurora Serverless v1 Anda

1. Buka CloudWatch konsol di https://console.aws.amazon.com/cloudwatch/.
2. Pilih Wilayah AWS Anda.
3. Pilih Grup log.
4. Pilih log klaster DB Aurora Serverless v1 Anda dari daftar. Untuk log kesalahan, pola penamaannya adalah sebagai berikut.

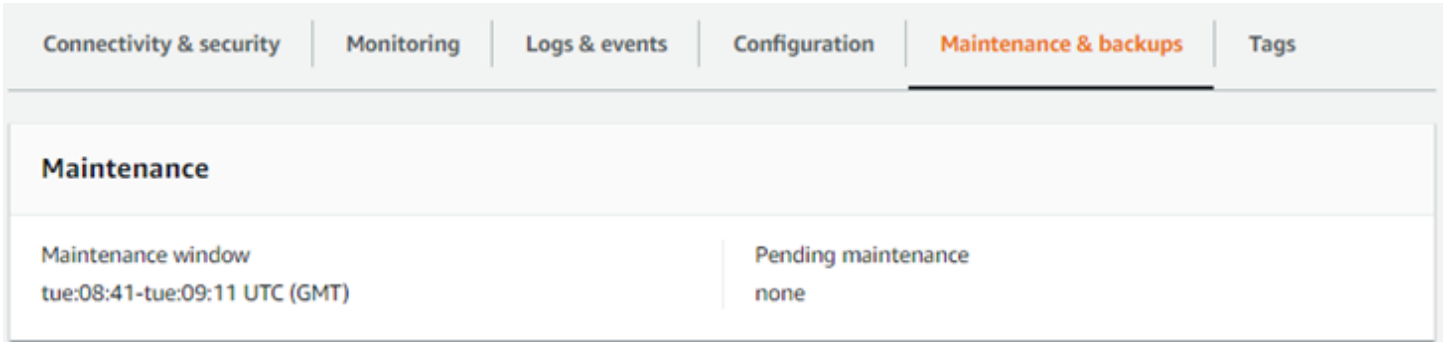
   ```
   /aws/rds/cluster/cluster-name/error
   ```

Misalnya, pada tangkapan layar berikut Anda dapat menemukan daftar untuk log yang diterbitkan untuk klaster DB Aurora Serverless v1 Aurora PostgreSQL bernama `western-sles`. Anda juga dapat menemukan beberapa entri untuk klaster DB Aurora Serverless v1 Aurora MySQL, `west-coast-sles`. Pilih log yang Anda minati untuk mulai mengkaji kontennya.



Aurora Serverless v1 dan pemeliharaan

Pemeliharaan untuk klaster DB Aurora Serverless v1, seperti menerapkan fitur terbaru, perbaikan, dan pembaruan keamanan, dilakukan secara otomatis untuk Anda. Aurora Serverless v1 memiliki periode pemeliharaan yang dapat Anda lihat di AWS Management Console pada bagian Pemeliharaan & pencadangan untuk klaster DB Aurora Serverless v1 Anda. Anda dapat menemukan tanggal dan waktu pemeliharaan yang mungkin dilakukan dan jika ada pemeliharaan yang tertunda untuk cluster Aurora Serverless v1 DB Anda, seperti yang ditunjukkan pada gambar berikut.



Anda dapat mengatur periode pemeliharaan ketika Anda membuat klaster DB Aurora Serverless v1, dan Anda dapat memodifikasi periode ini nanti. Untuk informasi selengkapnya, lihat Menyesuaikan periode pemeliharaan klaster DB yang dinginkan.

Jendela pemeliharaan digunakan untuk upgrade versi mayor terjadwal. Upgrade versi minor dan patch diterapkan segera selama penskalaan. Penskalaan terjadi sesuai dengan pengaturan Anda untuk`TimeoutAction`:

- `ForceApplyCapacityChange`— Perubahan diterapkan segera.
- `RollbackCapacityChange`— Aurora secara paksa memperbarui cluster setelah 3 hari dari upaya patch pertama.

Seperti halnya perubahan apa pun yang dipaksakan tanpa titik penskalaan yang tepat, hal ini dapat mengganggu beban kerja Anda.

Jika memungkinkan, Aurora Serverless v1 melakukan pemeliharaan dengan cara yang tidak mengganggu. Jika pemeliharaan diperlukan, klaster DB Aurora Serverless v1 Anda menskalakan kapasitasnya untuk menangani operasi yang diperlukan. Sebelum menskalakan, Aurora Serverless v1 mencari titik penskalaan. Itu dilakukan hingga tiga hari jika perlu.

Pada akhirnya jika Aurora Serverless v1 tidak dapat menemukan titik penskalaan, layanan tersebut akan membuat peristiwa klaster. Peristiwa ini memberi tahu Anda tentang pemeliharaan yang tertunda dan kebutuhan penskalaan untuk melakukan pemeliharaan. Notifikasi ini mencakup tanggal kapan Aurora Serverless v1 dapat memaksa klaster DB untuk diskalakan.

Untuk informasi selengkapnya, lihat Tindakan batas waktu habis untuk perubahan kapasitas.

Aurora Serverless v1 dan failover

Jika instans DB untuk klaster DB Aurora Serverless v1 menjadi tidak tersedia atau Zona Ketersediaan (AZ) tempat klaster DB ini berada mengalami kegagalan, Aurora akan membuat ulang instans DB di AZ yang berbeda. Namun, klaster Aurora Serverless v1 bukan klaster Multi-AZ. Hal ini karena klaster tersebut terdiri dari satu instans DB dalam satu AZ. Dengan demikian, mekanisme failover ini membutuhkan waktu lebih lama daripada untuk klaster Aurora dengan instans terprovisi atau instans Aurora Serverless v2. Waktu Aurora Serverless v1 failover tidak ditentukan karena bergantung pada permintaan dan ketersediaan kapasitas di AZ lain dalam Wilayah AWS yang ditentukan.

Karena Aurora memisahkan kapasitas komputasi dan penyimpanan, volume penyimpanan untuk klaster tersebar di beberapa AZ. Data Anda tetap tersedia meskipun pemadaman memengaruhi instans DB atau AZ terkait.

Aurora Serverless v1 dan snapshot

Volume klaster untuk klaster Aurora Serverless v1 selalu dienkripsi. Anda dapat memilih kunci enkripsi, tetapi tidak dapat menonaktifkan enkripsi. Untuk menyalin atau membagikan snapshot klaster Aurora Serverless v1, enkripsi snapshot menggunakan AWS KMS key Anda sendiri. Untuk

informasi selengkapnya, lihat Menyalin snapshot klaster DB. Untuk mempelajari selengkapnya tentang enkripsi dan Amazon Aurora, lihat Membuat klaster DB Amazon Aurora

Membuat klaster DB Aurora Serverless v1

Prosedur berikut membuat klaster Aurora Serverless v1 tanpa objek skema atau data Anda. Jika Anda ingin membuat klaster Aurora Serverless v1 yang merupakan duplikat dari klaster terprovisi atau klaster Aurora Serverless v1 yang ada, Anda dapat melakukan operasi pemulihan atau kloning snapshot. Untuk detailnya, lihat Memulihkan dari snapshot klaster DB dan Mengkloning volume untuk klaster DB Amazon Aurora. Anda tidak dapat mengonversi klaster terprovisi yang sudah ada ke Aurora Serverless v1. Anda juga tidak dapat mengonversi klaster Aurora Serverless v1 yang ada kembali ke klaster terprovisi.

Saat membuat klaster DB Aurora Serverless v1, Anda dapat mengatur kapasitas minimum dan maksimum untuk klaster tersebut. Unit kapasitas setara dengan konfigurasi komputasi dan memori tertentu. Aurora Serverless v1 membuat aturan penskalaan untuk ambang batas pemanfaatan CPU, koneksi, dan memori yang tersedia, serta menskalakan secara mulus ke berbagai unit kapasitas sesuai kebutuhan untuk aplikasi Anda. Untuk informasi selengkapnya, lihat Arsitektur Aurora Serverless v1.

Anda dapat mengatur nilai spesifik berikut untuk klaster DB Aurora Serverless v1 Anda:

- Unit kapasitas Aurora minimum – Aurora Serverless v1 dapat menurunkan kapasitas hingga mencapai unit kapasitas ini.
- Unit kapasitas Aurora maksimum – Aurora Serverless v1 dapat meningkatkan kapasitas hingga mencapai unit kapasitas ini.

Anda juga dapat memilih opsi konfigurasi penskalaan opsional berikut:

- Paksa penskalaan kapasitas ke nilai yang ditentukan saat batas waktu tercapai – Anda dapat memilih pengaturan ini jika ingin Aurora Serverless v1 memaksa Aurora Serverless v1 untuk menskalakan meskipun tidak dapat menemukan titik penskalaan sebelum batas waktu habis. Jika Anda ingin Aurora Serverless v1 membatalkan perubahan kapasitas jika tidak dapat menemukan titik penskalaan, jangan pilih pengaturan ini. Untuk informasi selengkapnya, lihat Tindakan batas waktu habis untuk perubahan kapasitas.
- Jeda kapasitas komputasi setelah tidak ada aktivitas dalam beberapa menit berturut-turut – Anda dapat memilih pengaturan ini jika Anda ingin Aurora Serverless v1 menskalakan ke nol saat tidak

ada aktivitas di klaster DB Anda selama jumlah waktu yang Anda tentukan. Dengan mengaktifkan pengaturan ini, klaster DB Aurora Serverless v1 Anda secara otomatis melanjutkan pemrosesan dan menskalakan ke kapasitas yang diperlukan untuk menangani beban kerja saat lalu lintas basis data berlanjut. Untuk mempelajari selengkapnya, lihat Jeda dan lanjutkan untuk Aurora Serverless v1.

Sebelum Anda dapat membuat cluster Aurora Serverless v1 DB, Anda memerlukan AWS akun. Anda juga perlu menyelesaikan tugas penyiapan untuk menggunakan Amazon Aurora. Untuk informasi selengkapnya, lihat Menyiapkan lingkungan Anda untuk Amazon Aurora. Anda juga perlu menyelesaikan langkah awal lainnya untuk membuat klaster DB Aurora. Untuk mempelajari selengkapnya, lihat Membuat klaster DB Amazon Aurora.

Aurora Serverless v1tersedia dalam versi tertentu Wilayah AWS dan untuk Aurora MySQL dan Aurora PostgreSQL tertentu saja. Untuk informasi selengkapnya, lihat Aurora Serverless v1.

> **Note**
>
> Volume klaster untuk klaster Aurora Serverless v1 selalu dienkripsi. Saat membuat klaster DB Aurora Serverless v1, Anda tidak dapat menonaktifkan enkripsi, tetapi Anda dapat memilih untuk menggunakan kunci enkripsi Anda sendiri. Dengan Aurora Serverless v2, Anda dapat memilih apakah akan mengenkripsi volume klaster.

Anda dapat membuat cluster Aurora Serverless v1 DB dengan AWS Management Console, the AWS CLI, atau RDS API.

> **Note**
>
> Jika Anda menerima pesan kesalahan berikut saat mencoba membuat klaster Anda, akun Anda memerlukan izin tambahan.
> `Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.`
> Untuk informasi selengkapnya, lihat Menggunakan peran tertaut layanan untuk Amazon Aurora.

Anda tidak dapat langsung terhubung ke instans DB di klaster DB Aurora Serverless v1 Anda. Untuk terhubung ke klaster DB Aurora Serverless v1, Anda perlu menggunakan titik akhir basis data.

Anda dapat menemukan titik akhir untuk klaster DB Aurora Serverless v1 Anda di tab Konektivitas & keamanan untuk klaster Anda di AWS Management Console. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster DB Amazon Aurora.

Konsol

Gunakan prosedur umum berikut. Untuk informasi lebih lanjut tentang membuat cluster Aurora DB menggunakan AWS Management Console, lihat. Membuat klaster DB Amazon Aurora

Untuk membuat klaster DB Aurora Serverless v1 baru

1. Masuk ke AWS Management Console.
2. Pilih Wilayah AWS yang mendukungAurora Serverless v1.
3. Pilih Amazon RDS dari daftar AWS Layanan.
4. Pilih Buat basis data.
5. Di halaman Buat basis data:
   a. Pilih Pembuatan Standar untuk metode pembuatan basis data.
   b. Lanjutkan membuat klaster DB Aurora Serverless v1 menggunakan langkah-langkah dari contoh berikut.

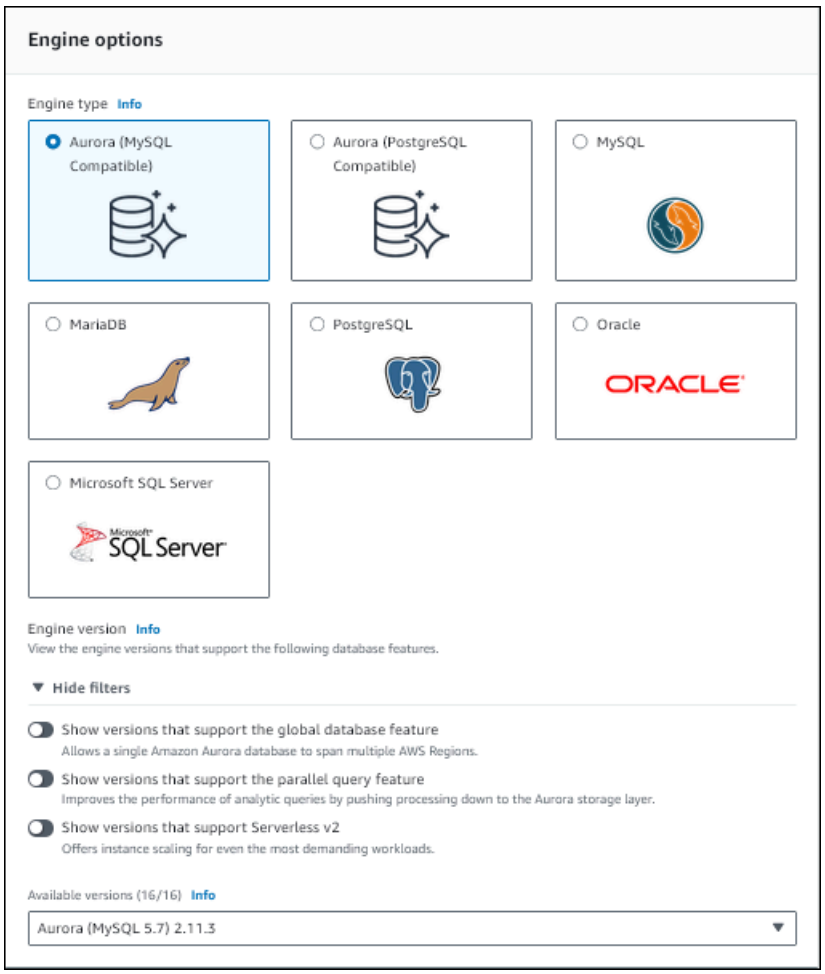
> **Note**
>
> Jika Anda memilih versi mesin DB yang tidak mendukung Aurora Serverless v1, opsi Nirserver tidak ditampilkan untuk kelas instans DB.

Contoh untuk Aurora MySQL

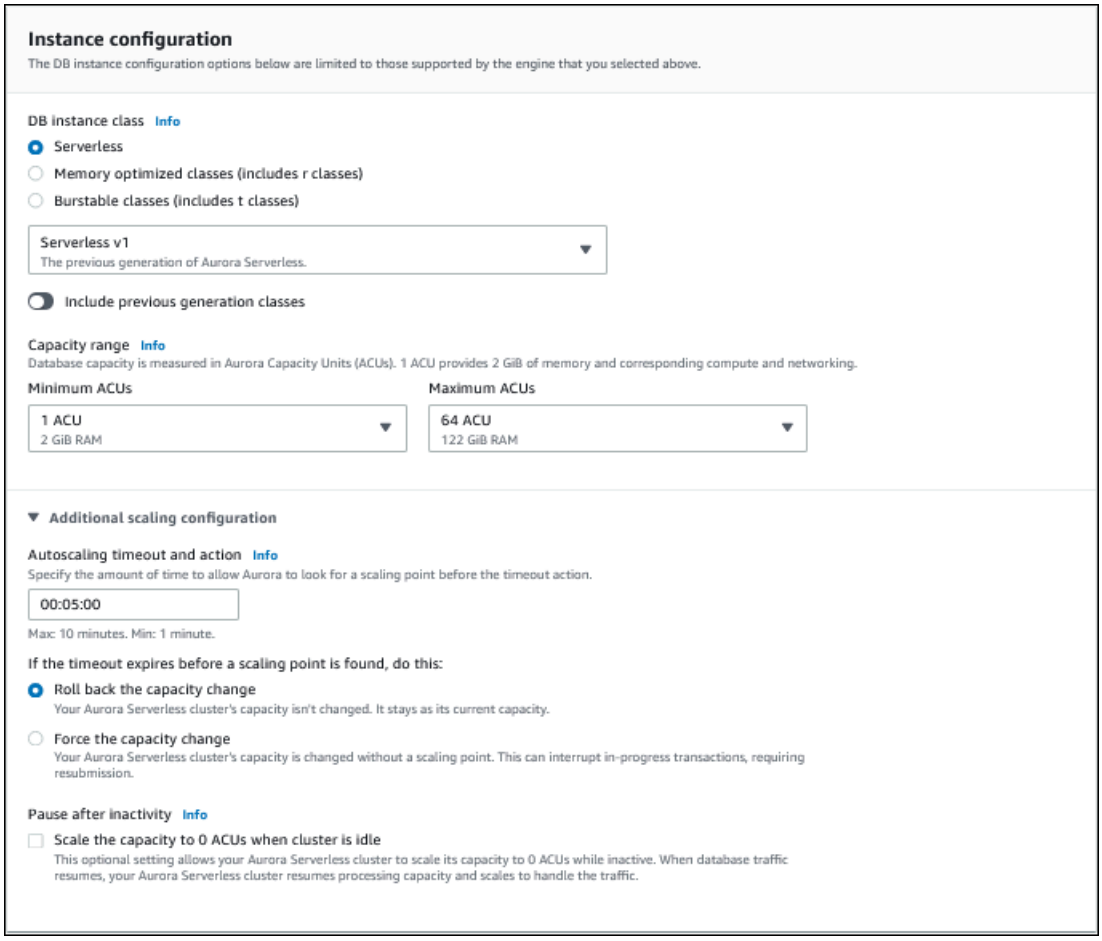
Gunakan prosedur berikut:

Untuk membuat klaster DB Aurora Serverless v1 untuk Aurora MySQL

1. Untuk Jenis mesin, pilih Aurora (Kompatibel dengan MySQL).
2. Pilih versi Aurora MySQL, yang kompatibel dengan Aurora Serverless v1, yang Anda inginkan untuk klaster DB Anda. Versi yang didukung ditampilkan di sisi kanan halaman.



3. Untuk Kelas instans DB, pilih Nirserver.
4. Atur Rentang kapasitas untuk klaster DB.
5. Sesuaikan nilai sesuai kebutuhan di bagian Konfigurasi penskalaan tambahan pada halaman. Untuk mempelajari selengkapnya tentang pengaturan kapasitas, lihat Penskalaan otomatis untuk Aurora Serverless v1.



6. Untuk mengaktifkan Data API untuk klaster DB Aurora Serverless v1 Anda, pilih kotak centang Data API dalam Konfigurasi tambahan pada bagian Konektivitas.

   Untuk mempelajari selengkapnya tentang Data API, lihat Menggunakan RDS Data API.

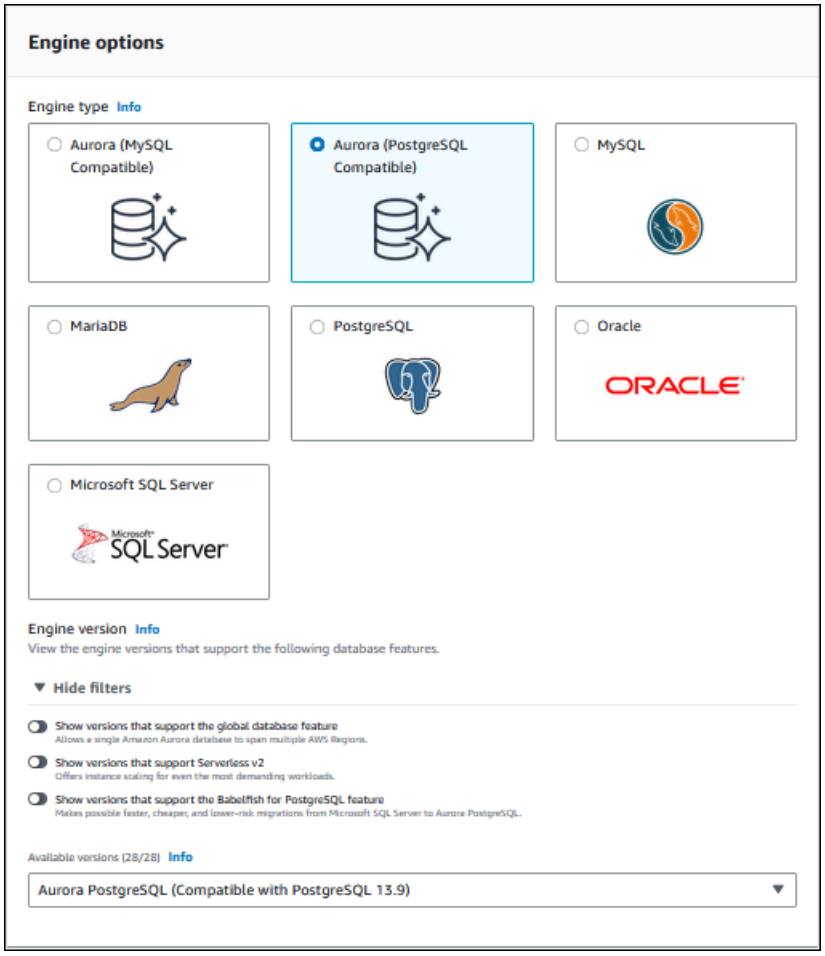
7. Pilih pengaturan basis data lain sesuai kebutuhan, lalu pilih Buat basis data.

Contoh untuk Aurora PostgreSQL

Gunakan prosedur berikut:

Untuk membuat klaster DB Aurora Serverless v1 untuk Aurora PostgreSQL

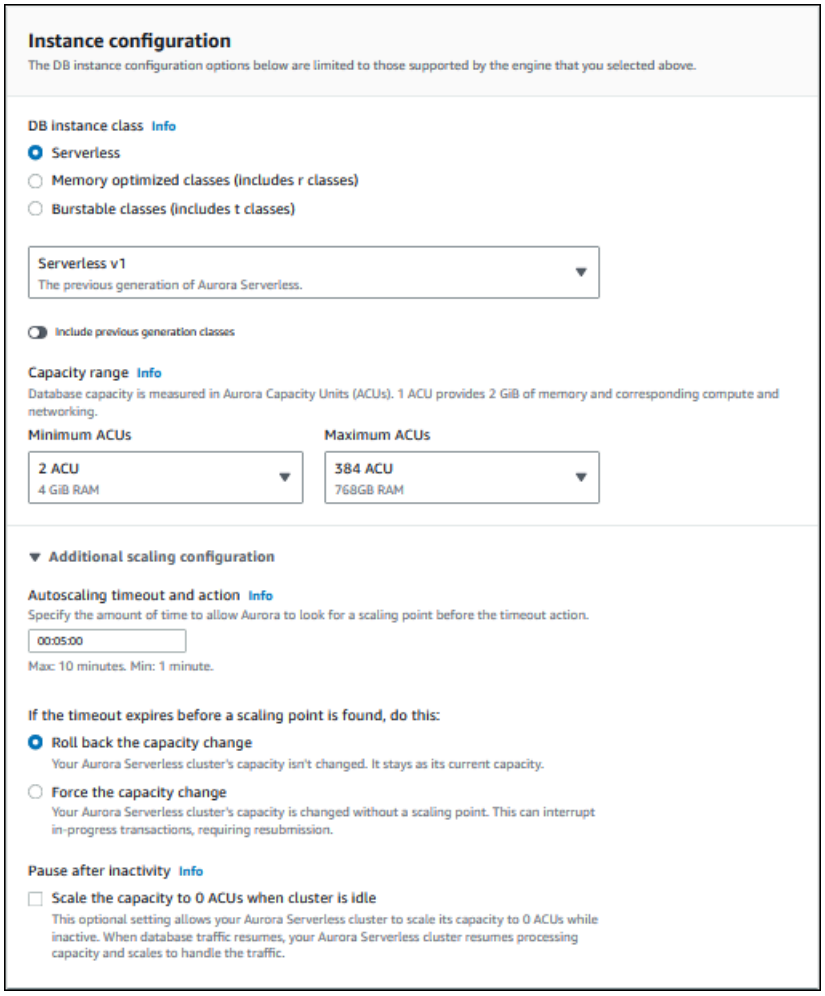
1. Untuk Jenis mesin, pilih Aurora (Kompatibel dengan PostgreSQL).
2. Pilih versi Aurora PostgreSQL, yang kompatibel dengan Aurora Serverless v1, yang Anda inginkan untuk klaster DB Anda. Versi yang didukung ditampilkan di sisi kanan halaman.



3. Untuk Kelas instans DB, pilih Nirserver.
4. Jika Anda memilih Aurora PostgreSQL versi 13 versi minor, pilih Nirserver v1 dari menu.

> **Note**
>
> Aurora PostgreSQL versi 13 juga mendukung Aurora Serverless v2.

5. Atur Rentang kapasitas untuk klaster DB.
6. Sesuaikan nilai sesuai kebutuhan di bagian Konfigurasi penskalaan tambahan pada halaman. Untuk mempelajari selengkapnya tentang pengaturan kapasitas, lihat Penskalaan otomatis untuk Aurora Serverless v1.



7. Untuk menggunakan Data API dengan klaster DB Aurora Serverless v1 Anda, pilih kotak centang Data API dalam Konfigurasi tambahan pada bagian Konektivitas.

   Untuk mempelajari selengkapnya tentang Data API, lihat Menggunakan RDS Data API.

8. Pilih pengaturan basis data lain sesuai kebutuhan, lalu pilih Buat basis data.

AWS CLI

Untuk membuat cluster Aurora Serverless v1 DB baru dengan AWS CLI, jalankan create-db-clusterperintah dan tentukan `serverless` untuk `--engine-mode` opsi.

Secara opsional, Anda dapat menentukan opsi `--scaling-configuration` untuk mengonfigurasi kapasitas minimum, kapasitas maksimum, dan jeda otomatis saat tidak ada koneksi.

Contoh perintah berikut membuat klaster DB Nirserver baru dengan mengatur opsi `--engine-mode` ke `serverless`. Contoh-contoh tersebut juga menentukan nilai untuk opsi `--scaling-configuration`.

Contoh untuk Aurora MySQL

Perintah berikut membuat klaster DB Nirserver yang kompatibel dengan Aurora MySQL baru. Nilai kapasitas yang valid untuk Aurora MySQL adalah 1, 2, 4, 8, 16, 32, 64, 128, dan 256.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

Untuk Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-mysql --engine-version 5.7.mysql_aurora.2.11.4 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=4,MaxCapacity=32,SecondsUntilAutoPause=1000,AutoPause=true ^
    --master-username username --master-user-password password
```

Contoh untuk Aurora PostgreSQL

Perintah berikut membuat klaster DB Nirserver yang kompatibel dengan PostgreSQL 13.9 baru. Nilai kapasitas yang valid untuk Aurora PostgreSQL adalah 2, 4, 8, 16, 32, 64, 192, dan 384.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster \
    --engine aurora-postgresql --engine-version 13.9 \
    --engine-mode serverless \
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true \
    --master-username username --master-user-password password
```

Untuk Windows:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster ^
    --engine aurora-postgresql --engine-version 13.9 ^
    --engine-mode serverless ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=1000,AutoPause=true ^
    --master-username username --master-user-password password
```

API RDS

Untuk membuat klaster DB Aurora Serverless v1 dengan API RDS, jalankan operasi CreateDBCluster dan tetapkan `serverless` untuk parameter `EngineMode`.

Secara opsional, Anda dapat menentukan parameter `ScalingConfiguration` untuk mengonfigurasi kapasitas minimum, kapasitas maksimum, dan jeda otomatis saat tidak ada koneksi. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, dan 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, dan 384.

Memulihkan klaster DB Aurora Serverless v1

Anda dapat mengonfigurasi klaster DB Aurora Serverless v1 saat memulihkan snapshot klaster DB terprovisi dengan AWS Management Console, AWS CLI, atau API RDS.

Saat Anda memulihkan snapshot ke klaster DB Aurora Serverless v1, Anda dapat mengatur nilai spesifik berikut:

- Unit kapasitas Aurora minimum – Aurora Serverless v1 dapat menurunkan kapasitas hingga mencapai unit kapasitas ini.
- Unit kapasitas Aurora maksimum – Aurora Serverless v1 dapat meningkatkan kapasitas hingga mencapai unit kapasitas ini.
- Tindakan batas waktu – Tindakan yang harus diambil jika batas waktu modifikasi kapasitas habis karena tidak dapat menemukan titik penskalaan. Klaster DB Aurora Serverless v1 dapat memaksa klaster DB Anda ke pengaturan kapasitas baru jika mengatur opsi Paksa penskalaan kapasitas ke nilai yang ditentukan..... Atau, hal ini dapat melakukan rollback perubahan kapasitas untuk membatalkannya jika Anda tidak memilih opsi tersebut. Untuk informasi selengkapnya, lihat Tindakan batas waktu habis untuk perubahan kapasitas.

- Jeda setelah tidak aktif – Jumlah waktu tanpa lalu lintas basis data untuk menskalakan ke kapasitas pemrosesan nol. Saat lalu lintas basis data dilanjutkan, Aurora secara otomatis melanjutkan kapasitas dan skala pemrosesan untuk menangani lalu lintas.

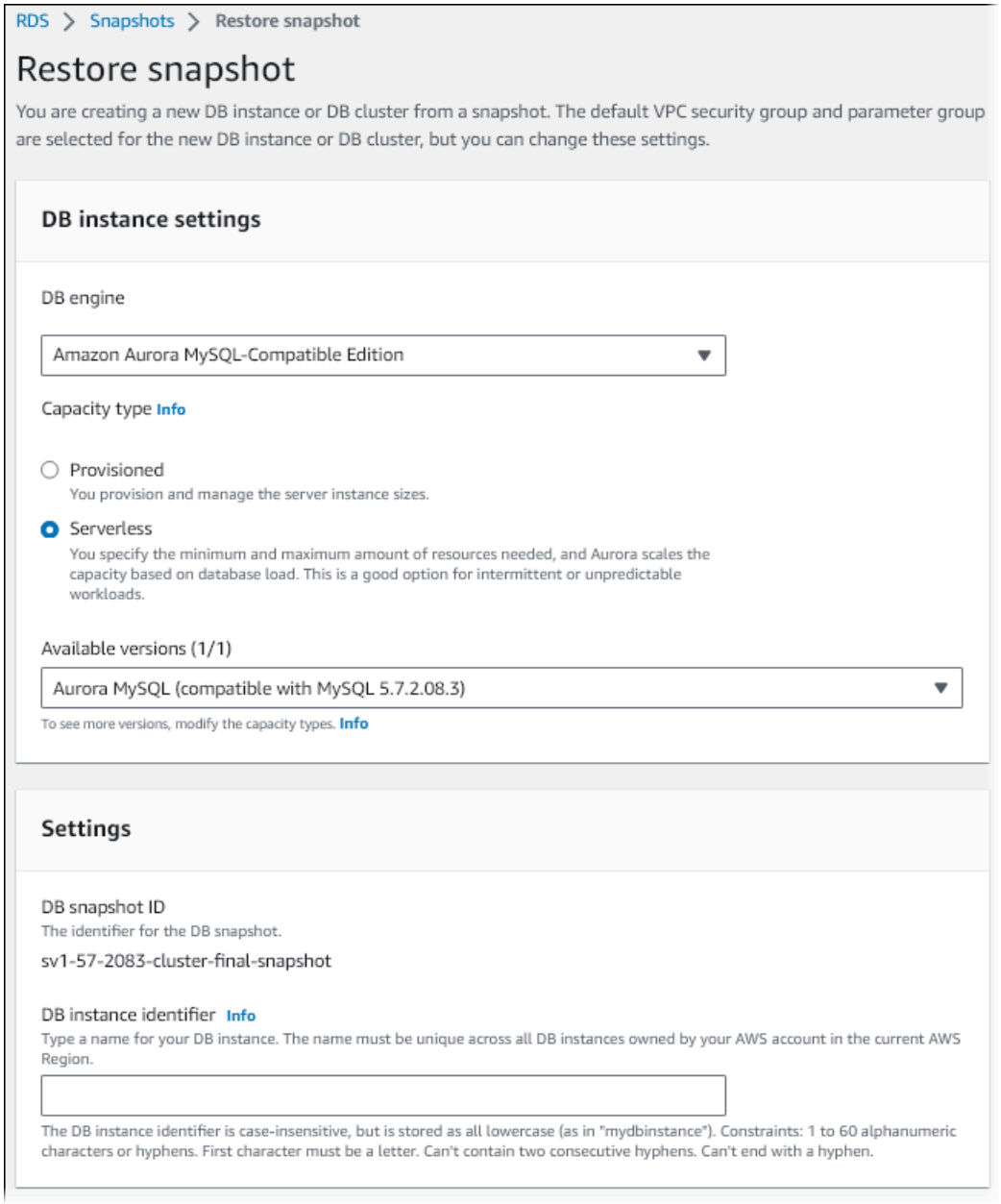
Untuk informasi umum tentang memulihkan klaster DB dari snapshot, lihat Memulihkan dari snapshot klaster DB.

Konsol

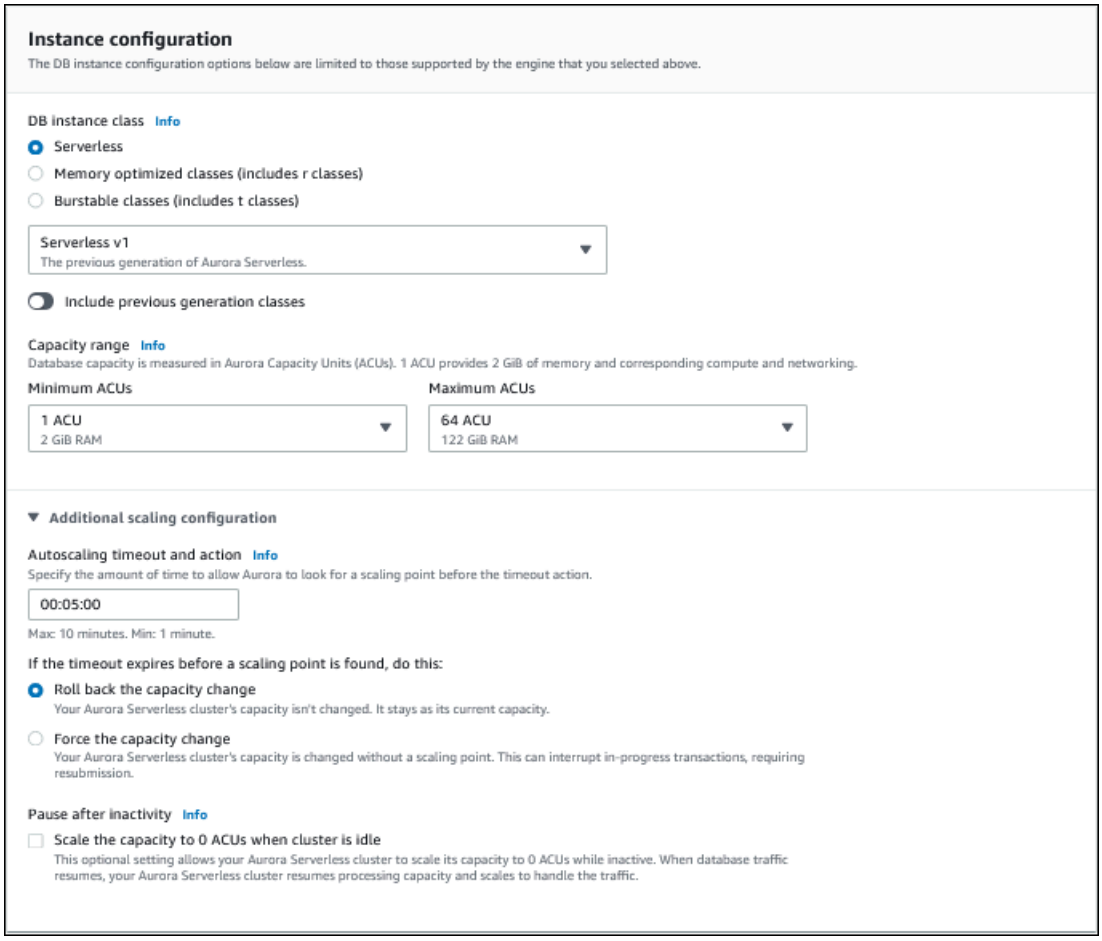
Anda dapat memulihkan snapshot klaster DB ke klaster DB Aurora dengan AWS Management Console.

Untuk memulihkan snapshot klaster DB ke klaster DB Aurora

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas AWS Management Console, pilih Wilayah AWS yang meng-host klaster DB sumber Anda.
3. Di panel navigasi, pilih Snapshot, lalu pilih snapshot klaster DB yang ingin Anda pulihkan.
4. Untuk Tindakan, pilih Pulihkan Snapshot.
5. Di halaman Pulihkan Klaster DB, pilih Nirserver untuk Tipe kapasitas.



6. Di bidang Pengidentifikasi klaster DB, ketik nama untuk klaster DB Anda yang dipulihkan, dan lengkapi bidang lainnya.
7. Di bagian Pengaturan kapasitas, ubah konfigurasi penskalaan.



8. Pilih Pulihkan Klaster DB.

Untuk terhubung ke klaster DB Aurora Serverless v1, gunakan titik akhir basis data. Untuk detailnya, lihat petunjuk dalam Menghubungkan ke klaster DB Amazon Aurora.

> **Note**
>
> Jika Anda menemukan pesan kesalahan berikut, akun Anda memerlukan izin tambahan: `Unable to create the resource. Verify that you have permission to create service linked role. Otherwise wait and try again later.` Untuk informasi selengkapnya, lihat Menggunakan peran tertaut layanan untuk Amazon Aurora.

AWS CLI

Anda dapat mengonfigurasi klaster DB Aurora Serverless saat memulihkan snapshot klaster DB terprovisi dengan AWS Management Console, AWS CLI, atau API RDS.

Saat Anda memulihkan snapshot ke klaster DB Aurora Serverless, Anda dapat mengatur nilai spesifik berikut:

- Unit kapasitas Aurora minimum – Aurora Serverless dapat menurunkan kapasitas hingga mencapai unit kapasitas ini.
- Unit kapasitas Aurora maksimum – Aurora Serverless dapat meningkatkan kapasitas hingga mencapai unit kapasitas ini.
- Tindakan batas waktu – Tindakan yang harus diambil jika batas waktu modifikasi kapasitas habis karena tidak dapat menemukan titik penskalaan. Klaster DB Aurora Serverless v1 dapat memaksa klaster DB Anda ke pengaturan kapasitas baru jika mengatur opsi Paksa penskalaan kapasitas ke nilai yang ditentukan..... Atau, hal ini dapat melakukan rollback perubahan kapasitas untuk membatalkannya jika Anda tidak memilih opsi tersebut. Untuk informasi selengkapnya, lihat Tindakan batas waktu habis untuk perubahan kapasitas.
- Jeda setelah tidak aktif – Jumlah waktu tanpa lalu lintas basis data untuk menskalakan ke kapasitas pemrosesan nol. Saat lalu lintas basis data dilanjutkan, Aurora secara otomatis melanjutkan kapasitas dan skala pemrosesan untuk menangani lalu lintas.

> **Note**
>
> Versi snapshot klaster DB harus kompatibel dengan Aurora Serverless v1. Untuk daftar versi yang didukung, lihat Aurora Serverless v1.

Untuk memulihkan snapshot ke klaster Aurora Serverless v1 dengan kompatibilitas MySQL 5.7, sertakan parameter tambahan berikut:

- `--engine` **`aurora-mysql`**
- `--engine-version` **`5.7`**

Parameter `--engine` dan `--engine-version` memungkinkan Anda membuat klaster Aurora Serverless v1 yang kompatibel dengan MySQL 5.7 dari Aurora yang kompatibel dengan MySQL

5.6 atau snapshot Aurora Serverless v1. Contoh berikut memulihkan snapshot dari klaster yang kompatibel dengan MySQL 5.6 bernama *mydbclustersnapshot* ke klaster Aurora Serverless v1 yang kompatibel dengan MySQL 5.7 bernama *mynewdbcluster*.

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless \
    --engine aurora-mysql \
    --engine-version 5.7
```

Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine aurora-mysql ^
    --engine-version 5.7
```

Secara opsional, Anda dapat menentukan opsi `--scaling-configuration` untuk mengonfigurasi kapasitas minimum, kapasitas maksimum, dan jeda otomatis saat tidak ada koneksi. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, dan 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, dan 384.

Dalam contoh berikut, Anda memulihkan dari snapshot klaster DB yang dibuat sebelumnya bernama *mydbclustersnapshot* ke klaster DB baru bernama *mynewdbcluster*. Anda mengatur `--scaling-configuration` sehingga klaster DB Aurora Serverless v1 baru dapat menskalakan dari 8 ACU ke 64 ACU (unit kapasitas Aurora) sesuai kebutuhan untuk memproses beban kerja. Setelah pemrosesan selesai dan setelah 1000 detik tanpa koneksi yang didukung, klaster akan dinonaktifkan hingga permintaan koneksi memintanya untuk diaktifkan ulang.

Untuk Linux, macOS, atau Unix:

```
aws rds restore-db-cluster-from-snapshot \
    --db-cluster-identifier mynewdbcluster \
```

```
    --snapshot-identifier mydbclustersnapshot \
    --engine-mode serverless --scaling-configuration
MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

Untuk Windows:

```
aws rds restore-db-cluster-from-snapshot ^
    --db-instance-identifier mynewdbcluster ^
    --db-snapshot-identifier mydbclustersnapshot ^
    --engine-mode serverless --scaling-configuration
MinCapacity=8,MaxCapacity=64,TimeoutAction='ForceApplyCapacityChange',SecondsUntilAutoPause=10
```

API RDS

Untuk mengonfigurasi cluster Aurora Serverless v1 DB saat Anda memulihkan dari cluster DB menggunakan RDS API, jalankan ClusterFromSnapshot operasi RestoreDB dan tentukan `serverless` parameternya. `EngineMode`

Secara opsional, Anda dapat menentukan parameter `ScalingConfiguration` untuk mengonfigurasi kapasitas minimum, kapasitas maksimum, dan jeda otomatis saat tidak ada koneksi. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, dan 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, dan 384.

Memodifikasi klaster DB Aurora Serverless v1

Setelah mengonfigurasi cluster Aurora Serverless v1 DB, Anda dapat memodifikasi properti tertentu dengan AWS CLI, API, atau RDS. AWS Management Console Sebagian besar properti yang dapat Anda ubah sama dengan jenis klaster Aurora lainnya.

Perubahan yang paling relevan untuk Aurora Serverless v1 adalah sebagai berikut:

- Memodifikasi konfigurasi penskalaan
- Meningkatkan versi mayor
- Mengonversi dari Aurora Serverless v1 menjadi terprovisi

Memodifikasi konfigurasi penskalaan klaster DB Aurora Serverless v1

Anda dapat mengatur kapasitas minimum dan maksimum untuk klaster DB. Setiap unit kapasitas setara dengan konfigurasi komputasi dan memori tertentu. Aurora Serverless secara otomatis membuat aturan penskalaan untuk ambang batas pemanfaatan CPU, koneksi, dan memori yang tersedia. Anda juga dapat menentukan apakah Aurora Serverless menjeda basis data saat tidak ada aktivitas lalu melanjutkannya saat aktivitas dimulai lagi.

Anda dapat mengatur nilai spesifik berikut untuk konfigurasi penskalaan:

- Unit kapasitas Aurora minimum – Aurora Serverless dapat menurunkan kapasitas hingga mencapai unit kapasitas ini.
- Unit kapasitas Aurora maksimum – Aurora Serverless dapat meningkatkan kapasitas hingga mencapai unit kapasitas ini.
- Waktu habis dan tindakan Penskalaan Otomatis – Bagian ini menentukan berapa lama Aurora Serverless menunggu untuk menemukan titik penskalaan sebelum batas waktu habis. Bagian ini juga menentukan tindakan yang harus diambil jika batas waktu modifikasi kapasitas habis karena tidak dapat menemukan titik penskalaan. Aurora dapat memaksa perubahan kapasitas untuk mengatur kapasitas ke nilai yang ditentukan sesegera mungkin. Atau, layanan ini dapat melakukan rollback perubahan kapasitas untuk membatalkannya. Untuk informasi selengkapnya, lihat Tindakan batas waktu habis untuk perubahan kapasitas.
- Jeda setelah tidak aktif – Gunakan pengaturan opsional Skalakan kapasitas ke 0 ACU saat klaster idle untuk menskalakan basis data ke nol kapasitas pemrosesan saat tidak aktif. Saat lalu lintas basis data dilanjutkan, Aurora secara otomatis melanjutkan kapasitas dan skala pemrosesan untuk menangani lalu lintas.

Konsol

Anda dapat mengubah konfigurasi penskalaan klaster DB Aurora dengan AWS Management Console.

Untuk memodifikasi klaster DB Aurora Serverless v1

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB Aurora Serverless v1 yang ingin Anda modifikasi.
4. Untuk Tindakan, pilih Ubah klaster.

5. Di bagian Pengaturan kapasitas, ubah konfigurasi penskalaan.
6. Pilih Lanjutkan.
7. Pada halaman Modifikasi klaster DB, tinjau modifikasi Anda, lalu pilih kapan harus menerapkannya.
8. Pilih Ubah klaster.

AWS CLI

Untuk memodifikasi konfigurasi penskalaan cluster Aurora Serverless v1 DB menggunakan AWS CLI, jalankan modify-db-cluster AWS CLI perintah. Tentukan opsi `--scaling-configuration` untuk mengonfigurasi kapasitas minimum, kapasitas maksimum, dan jeda otomatis saat tidak ada koneksi. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, dan 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, dan 384.

Dalam contoh ini, Anda memodifikasi konfigurasi penskalaan klaster DB Aurora Serverless v1 bernama *`sample-cluster`*.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --scaling-configuration
 MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange
```

API RDS

Anda dapat mengubah konfigurasi penskalaan klaster DB Aurora dengan operasi API ModifyDBCluster. Tentukan parameter `ScalingConfiguration` untuk mengonfigurasi kapasitas

minimum, kapasitas maksimum, dan jeda otomatis saat tidak ada koneksi. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, dan 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, dan 384.

Meningkatkan versi mayor klaster DB Aurora Serverless v1

Anda dapat meningkatkan versi mayor untuk klaster DB Aurora Serverless v1 yang kompatibel dengan PostgreSQL 11 ke versi sesuai yang kompatibel dengan PostgreSQL 13.

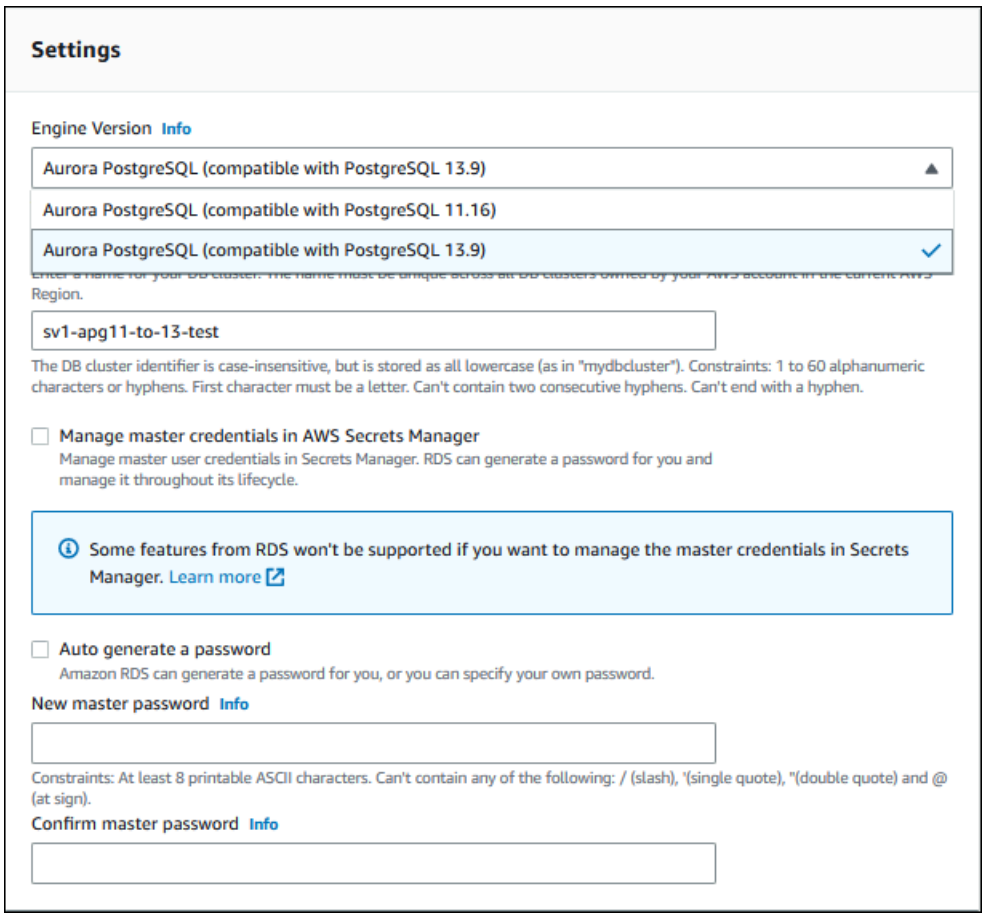
Konsol

Anda dapat melakukan peningkatan di tempat terhadap klaster DB Aurora Serverless v1 menggunakan AWS Management Console.

Untuk meningkatkan klaster DB Aurora Serverless v1

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB Aurora Serverless v1 yang ingin Anda tingkatkan.
4. Untuk Tindakan, pilih Ubah klaster.
5. Untuk Versi, pilih nomor versi Aurora PostgreSQL versi 13.

   Contoh berikut menunjukkan peningkatan di tempat dari Aurora PostgreSQL 11.16 ke 13.9.



Jika Anda melakukan peningkatan versi mayor, biarkan semua properti lainnya tetap sama. Untuk mengubah properti lainnya, lakukan operasi Modifikasi lain setelah peningkatan selesai.

6. Pilih Lanjutkan.
7. Pada halaman Modifikasi klaster DB, tinjau modifikasi Anda, lalu pilih kapan harus menerapkannya.
8. Pilih Ubah klaster.

AWS CLI

Untuk melakukan peningkatan di tempat terhadap klaster DB Aurora Serverless v1 yang kompatibel dengan PostgreSQL 11 ke klaster DB yang kompatibel dengan PostgreSQL 13, tentukan parameter `--engine-version` dengan nomor versi Aurora PostgreSQL versi 13 yang kompatibel dengan Aurora Serverless v1. Sertakan juga parameter `--allow-major-version-upgrade`.

Dalam contoh ini, Anda memodifikasi versi mayor klaster DB Aurora Serverless v1 yang kompatibel dengan PostgreSQL 11 bernama `sample-cluster`. Tindakan ini akan memulai peningkatan di tempat terhadap klaster DB Aurora Serverless v1 yang kompatibel dengan PostgreSQL 13.

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --engine-version 13.9 \
    --allow-major-version-upgrade
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine-version 13.9 ^
    --allow-major-version-upgrade
```

API RDS

Untuk melakukan peningkatan di tempat terhadap klaster DB Aurora Serverless v1 yang kompatibel dengan PostgreSQL 11 ke klaster DB yang kompatibel dengan PostgreSQL 13, tentukan parameter `EngineVersion` dengan nomor versi Aurora PostgreSQL versi 13 yang kompatibel dengan Aurora Serverless v1. Sertakan juga parameter `AllowMajorVersionUpgrade`.

Mengonversi klaster DB Aurora Serverless v1 menjadi klaster DB terprovisi

Anda dapat mengonversi klaster DB Aurora Serverless v1 ke klaster DB terprovisi. Untuk melakukan konversi, Anda perlu mengubah kelas instans DB menjadi Terprovisi. Anda dapat menggunakan konversi ini sebagai bagian dari peningkatan klaster DB Anda dari Aurora Serverless v1 ke Aurora Serverless v2. Untuk informasi selengkapnya, lihat Upgrade dari klaster Aurora Serverless v1 ke Aurora Serverless v2.

Proses konversi membuat instans DB pembaca di klaster DB, mempromosikan instans pembaca ke instans penulis, lalu menghapus instans Aurora Serverless v1 asli. Saat Anda mengonversi klaster DB, Anda tidak dapat melakukan modifikasi lain secara bersamaan, seperti mengubah versi mesin DB atau grup parameter klaster DB. Operasi konversi akan diterapkan segera, dan tidak dapat dibatalkan.

Selama konversi, snapshot klaster DB cadangan akan diambil dari klaster DB jika terjadi kesalahan. Pengidentifikasi untuk snapshot klaster DB memiliki form `pre-modify-engine-mode-`*`DB_cluster_identifier`*`-`*`timestamp`*.

Aurora menggunakan versi mesin minor DB default saat ini untuk klaster DB terprovisi.

Jika Anda tidak menyediakan kelas instans DB untuk klaster DB yang dikonversi, Aurora akan merekomendasikan kelas berdasarkan kapasitas maksimum klaster DB Aurora Serverless v1 asli. Kapasitas yang disarankan untuk pemetaan kelas instans ditunjukkan dalam tabel berikut.

Kapasitas maksimum Serverless (ACU)	Kelas instans DB terprovisi
1	db.t3.small
2	db.t3.medium
4	db.t3.large
8	db.r5.large
16	db.r5.xlarge
32	db.r5.2xlarge
64	db.r5.4xlarge
128	db.r5.8xlarge
192	db.r5.12xlarge
256	db.r5.16xlarge
384	db.r5.24xlarge

> **Note**
>
> Bergantung pada kelas instans DB yang Anda pilih, dan penggunaan basis data Anda, Anda mungkin melihat biaya yang berbeda untuk klaster DB terprovisi dibandingkan dengan Aurora Serverless v1.
>
> Jika Anda mengonversi klaster DB Aurora Serverless v1 Anda ke kelas instans DB (db.t*) yang dapat melonjak, Anda mungkin dikenai biaya tambahan untuk menggunakan klaster DB. Untuk informasi selengkapnya, lihat Jenis kelas instans DB.

AWS CLI

Untuk mengonversi cluster Aurora Serverless v1 DB ke cluster yang disediakan, jalankan perintah. modify-db-cluster AWS CLI

Parameter-parameter berikut diperlukan:

- `--db-cluster-identifier` – Klaster DB Aurora Serverless v1 yang Anda konversi menjadi klaster terprovisi.
- `--engine-mode` – Gunakan nilai `provisioned`.
- `--allow-engine-mode-change`
- `--db-cluster-instance-class` – Memilih kelas instans DB untuk klaster DB terprovisi berdasarkan kapasitas klaster DB Aurora Serverless v1.

Dalam contoh ini, Anda mengonversi klaster DB Aurora Serverless v1 bernama `sample-cluster` dan menggunakan kelas instans DB `db.r5.xlarge`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --engine-mode provisioned \
    --allow-engine-mode-change \
    --db-cluster-instance-class db.r5.xlarge
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --engine-mode provisioned ^
    --allow-engine-mode-change ^
    --db-cluster-instance-class db.r5.xlarge
```

API RDS

Untuk mengonversi klaster DB Aurora Serverless v1 ke klaster terprovisi, gunakan operasi API ModifyDBCluster.

Parameter berikut diperlukan:

- `DBClusterIdentifier` – Klaster DB Aurora Serverless v1 yang Anda konversi menjadi klaster terprovisi.
- `EngineMode` – Gunakan nilai `provisioned`.
- `AllowEngineModeChange`
- `DBClusterInstanceClass` – Memilih kelas instans DB untuk klaster DB terprovisi berdasarkan kapasitas klaster DB Aurora Serverless v1.

Menskalakan kapasitas klaster DB Aurora Serverless v1 secara manual

Klaster DB Aurora Serverless v1 biasanya diskalakan dengan lancar berdasarkan beban kerja. Namun, kapasitas mungkin tidak selalu diskalakan cukup cepat untuk memenuhi kondisi ekstrem yang tiba-tiba, seperti peningkatan transaksi secara eksponensial. Dalam kasus seperti itu, Anda dapat memulai operasi penskalaan secara manual dengan mengatur nilai kapasitas baru. Setelah Anda mengatur kapasitas secara eksplisit, Aurora Serverless v1 secara otomatis menskalakan klaster DB. Hal ini dilakukan berdasarkan periode pendinginan untuk menurunkan skala.

Anda dapat secara eksplisit mengatur kapasitas klaster DB Aurora Serverless v1 ke nilai tertentu dengan AWS Management Console, AWS CLI, atau API RDS.

Konsol

Anda dapat mengatur kapasitas klaster DB Aurora dengan AWS Management Console.

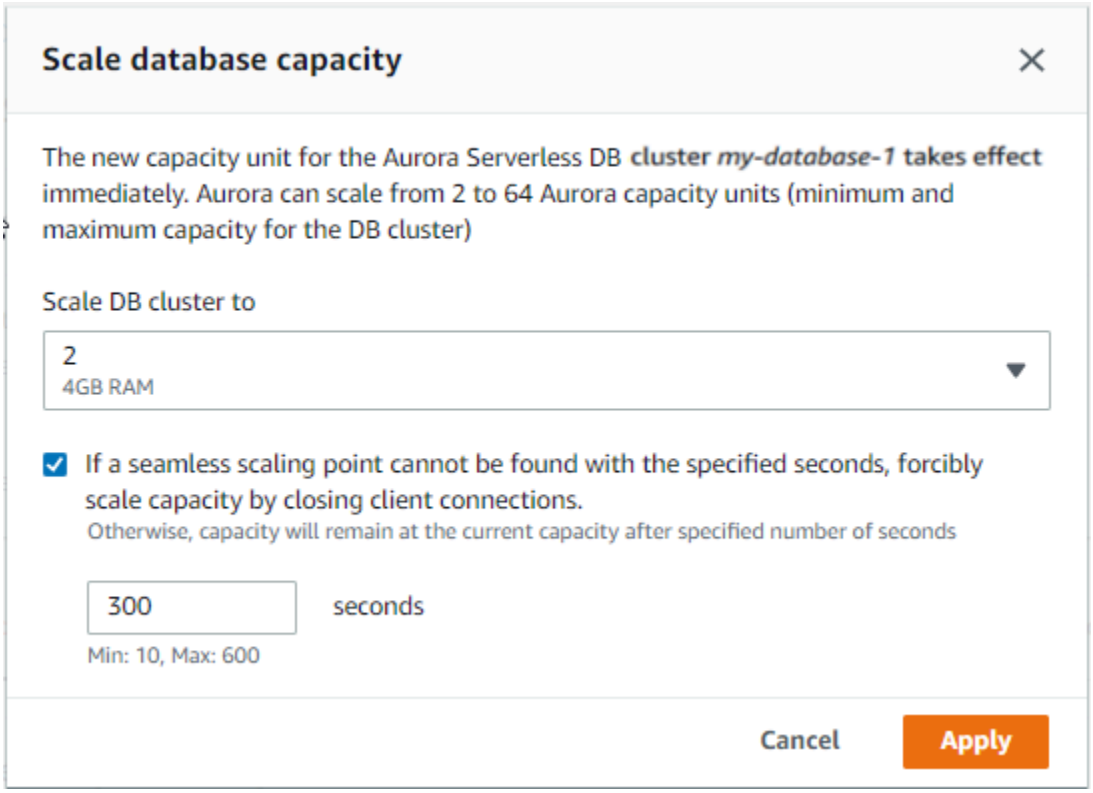
Untuk memodifikasi klaster DB Aurora Serverless v1

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih klaster DB Aurora Serverless v1 yang ingin Anda modifikasi.
4. Untuk Tindakan, pilih Atur kapasitas.
5. Di jendela Skalakan kapasitas basis data, pilih yang berikut:
    a. Untuk pemilih drop-down Skalakan klaster DB ke, pilih kapasitas baru yang Anda inginkan untuk klaster DB Anda.

b. Untuk kotak centang Jika titik penskalaan yang lancar tidak dapat ditemukan, pilih perilaku yang Anda inginkan untuk pengaturan `TimeoutAction` klaster DB Aurora Serverless v1 Anda, seperti berikut:

   - Hapus opsi ini jika Anda ingin kapasitas Anda tidak berubah saat Aurora Serverless v1 tidak dapat menemukan titik penskalaan sebelum batas waktu habis.
   - Pilih opsi ini jika Anda ingin memaksa klaster DB Aurora Serverless v1 Anda mengubah kapasitasnya meskipun tidak dapat menemukan titik penskalaan sebelum batas waktu habis. Opsi ini dapat mengakibatkan pemutusan koneksi Aurora Serverless v1 yang membuatnya tidak dapat menemukan titik penskalaan.

c. Untuk detik, masukkan jumlah waktu yang Anda inginkan untuk mengizinkan klaster DB Aurora Serverless v1 Anda mencari titik penskalaan sebelum batas waktu habis. Anda dapat menentukan nilai berapa pun dari 10 detik hingga 600 detik (10 menit). Default-nya adalah lima menit (300 detik). Contoh berikut memaksa klaster DB Aurora Serverless v1 untuk menurunkan skala ke 2 ACU meskipun tidak dapat menemukan titik penskalaan dalam lima menit.



6. Pilih Terapkan.

Untuk mempelajari selengkapnya tentang titik penskalaan, `TimeoutAction`, dan periode pendinginan, lihat Penskalaan otomatis untuk Aurora Serverless v1.

AWS CLI

Untuk mengatur kapasitas klaster DB Aurora Serverless v1 menggunakan AWS CLI, jalankan perintah modify-current-db-cluster-capacity AWS CLI, lalu tentukan opsi `--capacity`. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, dan 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, dan 384.

Dalam contoh ini, Anda mengatur kapasitas klaster DB Aurora Serverless v1 bernama *`sample-cluster`* menjadi *`64`*.

```
aws rds modify-current-db-cluster-capacity --db-cluster-identifier sample-cluster --
capacity 64
```

API RDS

Anda dapat mengatur kapasitas klaster DB Aurora dengan operasi API ModifyCurrentDBClusterCapacity. Tentukan parameter `Capacity`. Nilai kapasitas yang valid mencakup hal berikut:

- Aurora MySQL: 1, 2, 4, 8, 16, 32, 64, 128, dan 256.
- Aurora PostgreSQL: 2, 4, 8, 16, 32, 64, 192, dan 384.

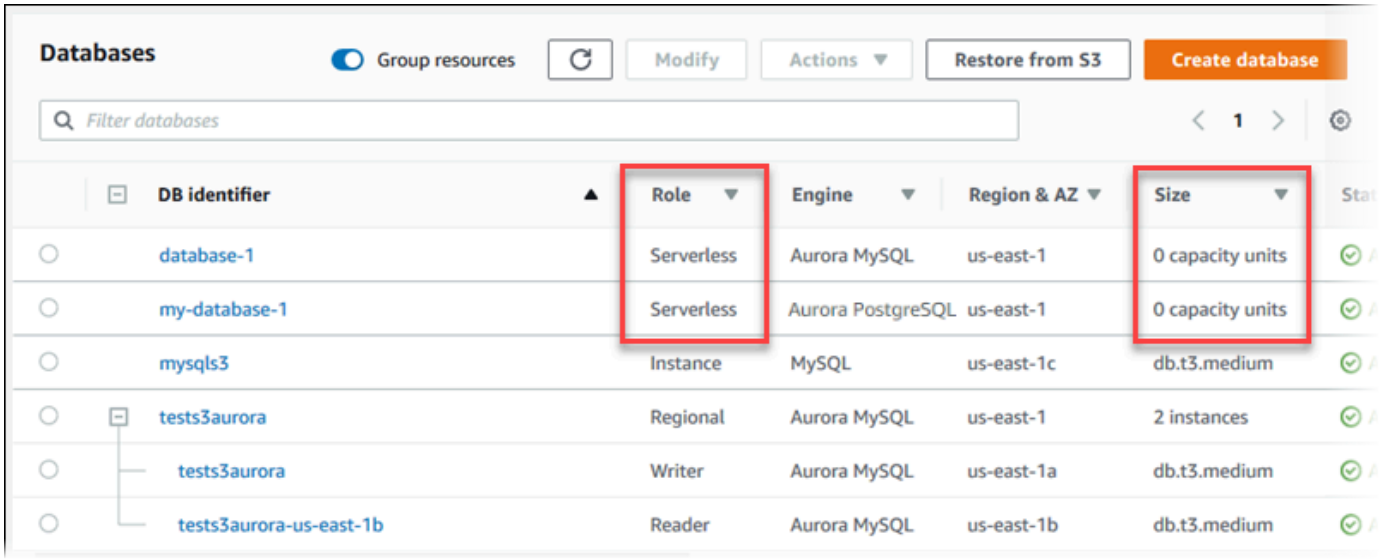
Melihat klaster DB Aurora Serverless v1

Setelah membuat satu atau beberapa klaster DB Aurora Serverless v1, Anda dapat melihat klaster DB mana yang berjenis Nirserver dan mana yang berjenis Instans. Anda juga dapat melihat jumlah terkini unit kapasitas Aurora (ACU) yang digunakan setiap klaster DB Aurora Serverless v1. Setiap ACU merupakan kombinasi dari kapasitas pemrosesan (CPU) dan memori (RAM).

Untuk melihat klaster DB Aurora Serverless v1 Anda

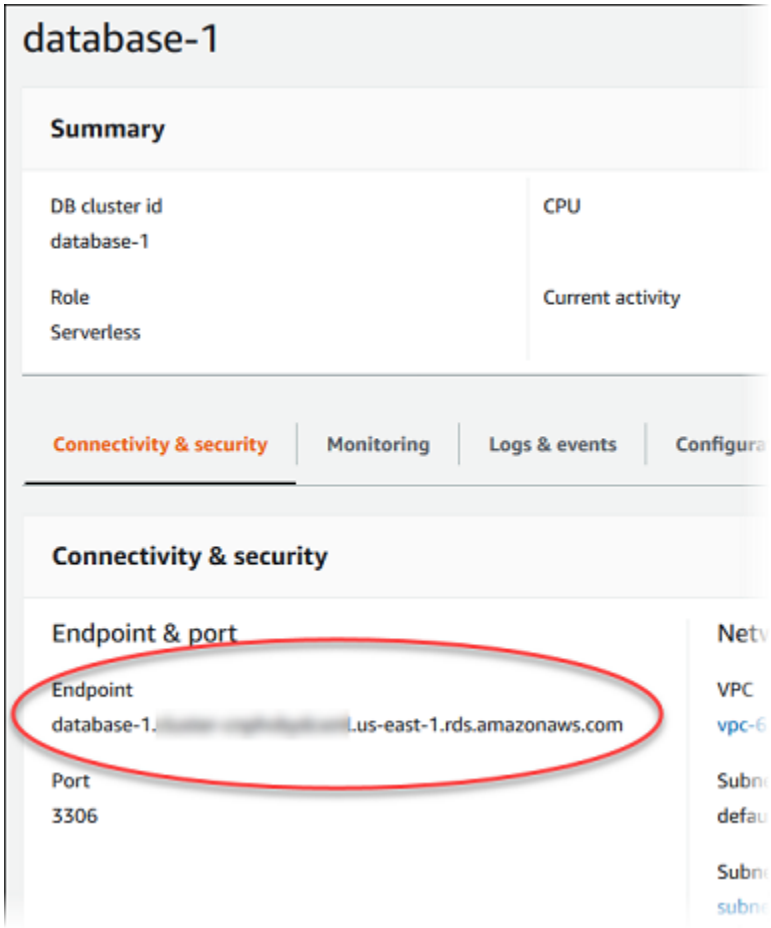
1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas AWS Management Console, pilih Wilayah AWS tempat klaster DB Aurora Serverless v1 dibuat.
3. Di panel navigasi, pilih Basis data.

   Untuk setiap klaster DB, jenis klaster DB ditampilkan di bagian Peran. Klaster DB Aurora Serverless v1 menampilkan Nirserver sebagai jenis. Anda dapat melihat kapasitas klaster DB Aurora Serverless v1 saat ini di bagian Ukuran.

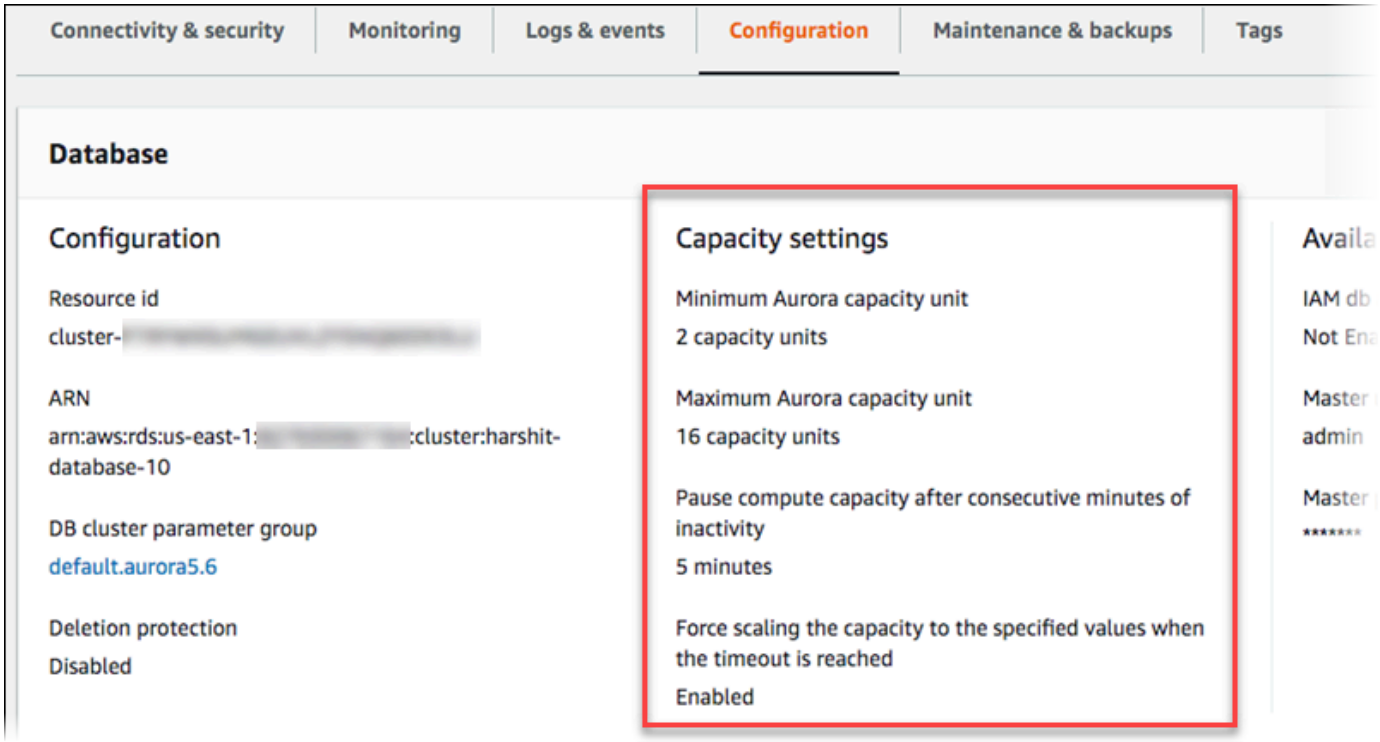


4. Pilih nama klaster DB Aurora Serverless v1 untuk menampilkan detailnya.

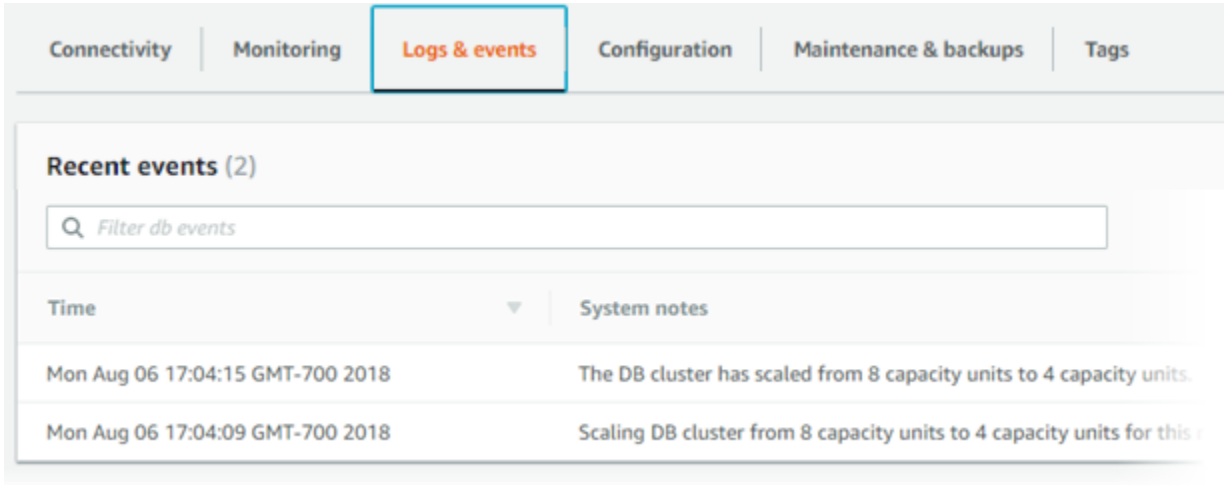
   Di tab Konektivitas & keamanan, perhatikan titik akhir basis data. Gunakan titik akhir ini untuk terhubung ke klaster DB Aurora Serverless v1 Anda.



Pilih tab Konfigurasi untuk melihat pengaturan kapasitas.



Peristiwa penskalaan dihasilkan ketika klaster DB dinaikkan skalanya, diturunkan skalanya, dijeda, atau dilanjutkan. Pilih tab Log & peristiwa untuk melihat peristiwa terbaru. Gambar berikut menunjukkan contoh peristiwa tersebut.



Memantau kapasitas dan peristiwa penskalaan untuk klaster DB Aurora Serverless v1 Anda

Anda dapat melihat klaster DB Aurora Serverless v1 Anda di CloudWatch untuk memantau kapasitas yang dialokasikan ke klaster DB dengan metrik `ServerlessDatabaseCapacity`.

Anda juga dapat memantau semua metrik Aurora CloudWatch standar, seperti `CPUUtilization`, `DatabaseConnections`, `Queries` dan seterusnya.

Anda dapat mengatur Aurora agar menerbitkan beberapa atau semua log basis data ke CloudWatch. Anda memilih log yang akan diterbitkan dengan mengaktifkan parameter konfigurasi seperti `general_log` dan `slow_query_log` dalam grup parameter klaster DB yang terkait dengan klaster Aurora Serverless v1. Tidak seperti klaster terprovisi, klaster Aurora Serverless v1 tidak mengharuskan Anda untuk menetapkan, dalam pengaturan klaster DB, jenis log mana yang akan diunggah ke CloudWatch. Klaster Aurora Serverless v1 secara otomatis mengunggah semua log yang tersedia. Saat Anda menonaktifkan parameter konfigurasi log, penerbitan log ke CloudWatch berhenti. Anda juga dapat menghapus log dalam CloudWatch jika tidak lagi diperlukan.

Untuk mulai menggunakan Amazon CloudWatch untuk klaster DB Aurora Serverless v1 Anda, lihat Melihat Aurora Serverless v1 log dengan Amazon CloudWatch. Untuk mempelajari selengkapnya tentang cara memantau klaster DB Aurora melalui CloudWatch, lihat Memantau peristiwa log di Amazon CloudWatch.

Untuk terhubung ke klaster DB Aurora Serverless v1, gunakan titik akhir basis data. Untuk informasi selengkapnya, lihat Menghubungkan ke klaster DB Amazon Aurora.

> **Note**
>
> Anda tidak dapat terhubung langsung ke instans DB tertentu di klaster DB Aurora Serverless v1 Anda.

Menghapus klaster DB Aurora Serverless v1

Saat Anda membuat klaster DB Aurora Serverless v1 menggunakan AWS Management Console, opsi Aktifkan perlindungan default diaktifkan secara default kecuali jika Anda membatalkan pemilihannya. Artinya, Anda tidak dapat segera menghapus klaster DB Aurora Serverless v1 dengan Perlindungan penghapusan yang diaktifkan. Untuk menghapus klaster DB Aurora Serverless v1 yang memiliki perlindungan penghapusan menggunakan AWS Management Console, Anda pertama-tama perlu memodifikasi klaster untuk menghapus perlindungan ini. Untuk informasi tentang penggunaan AWS CLI untuk tugas ini, lihat AWS CLI.

Untuk menonaktifkan perlindungan penghapusan menggunakan AWS Management Console

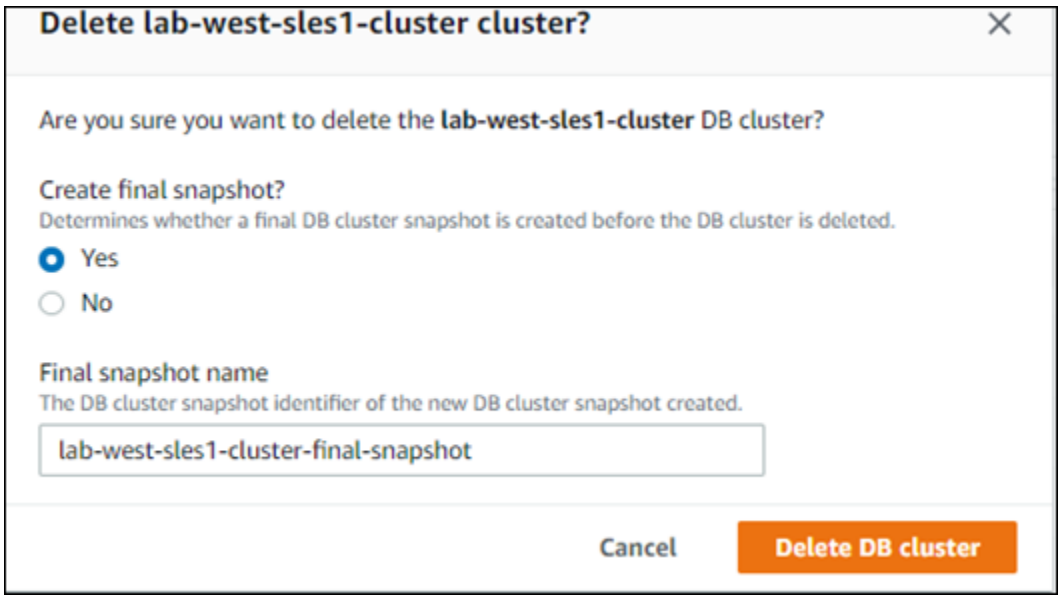
1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Klaster DB.
3. Pilih klaster DB Aurora Serverless v1 dari daftar.
4. Pilih Modifikasi untuk membuka konfigurasi klaster DB Anda. Halaman Modifikasi klaster DB membuka Pengaturan, Pengaturan kapasitas, dan detail konfigurasi lainnya untuk klaster DB Aurora Serverless v1 Anda. Perlindungan penghapusan ada di bagian Konfigurasi tambahan.
5. Kosongkan kotak centang Aktifkan perlindungan penghapusan di kartu properti Konfigurasi tambahan.
6. Pilih Lanjutkan. Ringkasan modifikasi akan muncul.
7. Pilih Ubah klaster untuk menerima ringkasan modifikasi tersebut. Anda juga dapat memilih Kembali untuk memodifikasi perubahan atau Batalkan untuk menghapus perubahan Anda.

Setelah perlindungan penghapusan tidak lagi aktif, Anda dapat menghapus klaster DB Aurora Serverless v1 menggunakan AWS Management Console.

Konsol

Untuk menghapus klaster DB Aurora Serverless v1

1. Masuk ke AWS Management Console lalu buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di bagian Sumber Daya, pilih Klaster DB.
3. Pilih klaster DB Aurora Serverless v1 yang ingin Anda hapus.
4. Untuk Tindakan, pilih Hapus. Anda akan diminta untuk mengonfirmasi bahwa Anda ingin menghapus klaster DB Aurora Serverless v1.
5. Kami menyarankan agar Anda tetap mempertahankan opsi yang telah dipilih:
   - Ya untuk Buat snapshot akhir?
   - Nama klaster DB Aurora Serverless v1 Anda ditambah `-final-snapshot` untuk Nama snapshot akhir. Namun, Anda dapat mengubah nama untuk snapshot akhir Anda di bidang ini.



Jika Anda memilih Tidak untuk Buat snapshot akhir? Anda tidak dapat memulihkan cluster DB Anda menggunakan snapshot atau point-in-time pemulihan.

6. Pilih Hapus klaster DB.

Aurora Serverless v1 akan menghapus klaster DB Anda. Jika Anda memilih untuk memiliki snapshot akhir, Anda akan melihat status klaster DB Aurora Serverless v1 berubah menjadi "Mencadangkan" sebelum klaster ini dihapus dan tidak lagi muncul dalam daftar.

AWS CLI

Sebelum memulai, konfigurasikan AWS CLI Anda dengan ID Kunci Akses AWS, Kunci Akses Rahasia AWS, dan Wilayah AWS tempat klaster DB Aurora Serverless v1 Anda berada. Untuk informasi selengkapnya, lihat Dasar-dasar konfigurasi dalam Panduan Pengguna AWS Command Line Interface.

Anda tidak dapat menghapus klaster DB Aurora Serverless v1 sebelum Anda menonaktifkan perlindungan penghapusan untuk klaster yang dikonfigurasi dengan opsi ini. Jika Anda mencoba menghapus klaster yang opsi perlindungannya diaktifkan, Anda akan melihat pesan kesalahan berikut ini.

```
An error occurred (InvalidParameterCombination) when calling the DeleteDBCluster
  operation: Cannot delete protected Cluster, please disable deletion protection and
 try again.
```

Anda dapat mengubah pengaturan perlindungan penghapusan klaster Aurora Serverless v1 DB Anda dengan menggunakan modify-db-clusterAWS CLIperintah seperti yang ditunjukkan pada berikut ini:

```
aws rds modify-db-cluster --db-cluster-identifier your-cluster-name --no-deletion-
protection
```

Perintah ini menampilkan properti yang direvisi untuk klaster DB yang ditentukan. Sekarang Anda dapat menghapus klaster DB Aurora Serverless v1 Anda.

Kami menyarankan agar Anda selalu membuat snapshot akhir setiap kali menghapus klaster DB Aurora Serverless v1. Contoh penggunaan berikut AWS CLI delete-db-clustermenunjukkan caranya. Anda memberi nama klaster DB Anda dan nama untuk snapshot.

Untuk Linux, macOS, atau Unix:

```
aws rds delete-db-cluster --db-cluster-identifier \
  your-cluster-name --no-skip-final-snapshot \
  --final-db-snapshot-identifier name-your-snapshot
```

Untuk Windows:

```
aws rds delete-db-cluster --db-cluster-identifier ^
  your-cluster-name --no-skip-final-snapshot ^
  --final-db-snapshot-identifier name-your-snapshot
```

Aurora Serverless v1 dan versi mesin basis data Aurora

Aurora Serverless v1 tersedia di Wilayah AWS tertentu serta untuk versi Aurora MySQL dan Aurora PostgreSQL tertentu saja. Untuk daftar terkini Wilayah AWS yang mendukung Aurora Serverless v1 serta versi Aurora MySQL dan Aurora PostgreSQL spesifik yang tersedia di setiap Wilayah, lihat Aurora Serverless v1.

Aurora Serverless v1 menggunakan mesin basis data Aurora terkaitnya untuk mengidentifikasi rilis tertentu yang didukung untuk setiap mesin basis data yang didukung, sebagai berikut:

- Aurora MySQL Serverless
- Aurora PostgreSQL Serverless

Saat rilis minor mesin basis data tersedia untuk Aurora Serverless v1, rilis tersebut diterapkan secara otomatis di berbagai Wilayah AWS tempat Aurora Serverless v1 tersedia. Dengan kata lain, Anda tidak perlu meng-upgrade klaster DB Aurora Serverless v1 untuk mendapatkan rilis minor baru untuk mesin DB klaster Anda jika tersedia untuk Aurora Serverless v1.

Aurora MySQL Serverless

Jika Anda ingin menggunakan Aurora MySQL Compatible Edition untuk klaster DB Aurora Serverless v1 Anda, Anda dapat memilih Aurora MySQL versi 2 yang kompatibel dengan MySQL 5.7. Untuk mempelajari penyempurnaan dan perbaikan bug untuk Aurora MySQL versi 2, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL versi 2 dalam Catatan Rilis untuk Aurora MySQL.

Aurora PostgreSQL Serverless

Jika Anda ingin menggunakan Aurora PostgreSQL untuk klaster DB Aurora Serverless v1 Anda, Anda dapat memilih di antara versi yang kompatibel dengan Aurora PostgreSQL 11 dan yang kompatibel dengan Aurora PostgreSQL 13. Rilis minor untuk Aurora PostgreSQL Compatible Edition hanya mencakup perubahan yang memiliki kompatibilitas mundur. Klaster DB Aurora Serverless v1 Anda di-upgrade secara transparan saat rilis minor Aurora PostgreSQL tersedia untuk Aurora Serverless v1 di Wilayah AWS Anda.

Misalnya, rilis Aurora PostgreSQL 11.16 versi minor diterapkan secara transparan ke semua klaster DB Aurora Serverless v1 yang menjalankan Aurora PostgreSQL versi sebelumnya. Untuk informasi selengkapnya tentang pembaruan Aurora PostgreSQL versi 11.16, lihat PostgreSQL 11.16 dalam Catatan Rilis untuk Aurora PostgreSQL.

Menggunakan RDS Data API

Dengan menggunakan RDS Data API (Data API), Anda dapat bekerja dengan antarmuka layanan web ke cluster Aurora DB Anda. Data API tidak memerlukan koneksi persisten ke cluster DB. Sebaliknya, API Data menyediakan titik akhir HTTP aman dan integrasi dengan AWS SDK. Anda dapat menggunakan titik akhir untuk menjalankan pernyataan SQL tanpa mengelola koneksi.

Semua panggilan ke Data API sinkron. Secara default, waktu panggilan akan habis jika tidak selesai diproses dalam 45 detik. Namun, Anda dapat terus menjalankan pernyataan SQL jika waktu panggilan habis dengan menggunakan parameter `continueAfterTimeout`. Lihat contohnya di Menjalankan transaksi SQL.

Pengguna tidak perlu meneruskan kredensil dengan panggilan ke Data API, karena Data API menggunakan kredenal database yang disimpan di dalamnya. AWS Secrets Manager Untuk menyimpan kredensil di Secrets Manager, pengguna harus diberikan izin yang sesuai untuk menggunakan Secrets Manager, dan juga Data API. Lihat informasi selengkapnya tentang mengizinkan pengguna di Mengotorisasi akses ke RDS Data API.

Anda juga dapat menggunakan Data API untuk mengintegrasikan Amazon Aurora dengan AWS aplikasi lain sepertiAWS Lambda,AWS AppSync, dan. AWS Cloud9 Data API menyediakan cara yang lebih aman untuk digunakanAWS Lambda. API ini memungkinkan Anda mengakses klaster basis data tanpa perlu mengonfigurasi fungsi Lambda untuk mengakses sumber daya di cloud privat virtual (VPC). Lihat informasi selengkapnya di AWS Lambda, AWS AppSync, dan AWS Cloud9.

Anda dapat mengaktifkan Data API saat membuat cluster Aurora DB. Anda juga dapat mengubah konfigurasinya di lain waktu. Untuk informasi selengkapnya, lihat Mengaktifkan API Data RDS.

Setelah mengaktifkan Data API, Anda juga dapat menggunakan editor kueri untuk menjalankan kueri ad hoc tanpa mengonfigurasi alat kueri untuk mengakses Aurora di VPC. Untuk informasi selengkapnya, lihat Menggunakan editor kueri.

Topik

- Ketersediaan Wilayah dan versi
- Keterbatasan dengan API Data RDS
- Perbandingan RDS Data API dengan Serverless v2 dan provisioned, dan Aurora Serverless v1
- Mengotorisasi akses ke RDS Data API
- Mengaktifkan API Data RDS

- Membuat endpoint Amazon VPC untuk RDS Data API () AWS PrivateLink
- Memanggil RDS Data API
- Menggunakan pustaka klien Java untuk RDS Data API
- Memproses hasil kueri dalam format JSON
- Memecahkan masalah RDS Data API
- Mencatat panggilan API Data RDS dengan AWS CloudTrail

Ketersediaan Wilayah dan versi

Untuk informasi tentang Wilayah dan versi engine yang tersedia untuk Data API, lihat bagian berikut.

Jenis cluster	Ketersediaan Wilayah dan versi
Aurora PostgreSQL disediakan dan Serverless v2	Data API dengan Aurora PostgreSQL Serverless v2 dan disediakan
Aurora PostgreSQL Tanpa Server v1	API Data dengan Aurora PostgreSQL Serverless v1
Aurora MySQL Tanpa Server v1	API Data dengan Aurora MySQL Serverless v1

> **Note**
>
> Saat ini, Data API tidak tersedia untuk klaster DB v2 yang disediakan Aurora MySQL atau Tanpa Server v2.

Jika Anda memerlukan modul kriptografi yang divalidasi oleh FIPS 140-2 saat mengakses Data API melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Lihat informasi selengkapnya tentang titik akhir FIPS yang tersedia di Standar Pemrosesan Informasi Federal (FIPS) 140-2.

Keterbatasan dengan API Data RDS

RDS Data API (Data API) memiliki batasan sebagai berikut:

- Anda hanya dapat menjalankan kueri Data API pada instance penulis di cluster DB. Namun, instance penulis dapat menerima kueri tulis dan baca.
- Dengan database global Aurora, Anda dapat mengaktifkan API Data pada cluster DB primer dan sekunder. Namun, sampai cluster sekunder dipromosikan menjadi yang utama, ia tidak memiliki instance penulis. Dengan demikian, kueri Data API yang Anda kirim ke sekunder gagal. Setelah sekunder yang dipromosikan memiliki instance penulis yang tersedia, kueri Data API pada instans DB tersebut akan berhasil.
- Performance Insights tidak mendukung pemantauan kueri database yang Anda buat menggunakan Data API.
- Data API tidak didukung pada kelas instans T DB.
- Untuk Aurora PostgreSQL Serverless v2 dan cluster DB yang disediakan, RDS Data API tidak mendukung tipe yang disebutkan. Untuk daftar tipe yang didukung, lihatthe section called "Perbandingan dengan Serverless v2 dan provisioned, dan Aurora Serverless v1".
- Untuk Aurora PostgreSQL versi 14 dan database yang lebih tinggi, Data API hanya mendukung scram-sha-256 untuk enkripsi kata sandi.

Perbandingan RDS Data API dengan Serverless v2 dan provisioned, dan Aurora Serverless v1

Tabel berikut menjelaskan perbedaan antara RDS Data API (Data API) dengan Aurora PostgreSQL Serverless v2 dan cluster DB yang disediakan, dan cluster DB. Aurora Serverless v1

Perbedaan	Aurora PostgreSQL Serverless v2 dan disediakan	Aurora Serverless v1
Jumlah maksimum permintaan per detik	Tidak terbatas.	1.000
Mengaktifkan atau menonaktifkan Data API pada database yang ada dengan menggunakan RDS API atau AWS CLI	• RDS API — Gunakan `EnableHttpEndpoint` dan `DisableHttpEndpoint` operasi. • AWS CLI— Gunakan `enable-http-endpoi`	• RDS API — Gunakan `ModifyDBCluster` operasi, dan tentukan `true` atau`false`, sebagaimana berlaku, untuk `EnableHttpEndpoint` parameter.

Perbedaan	Aurora PostgreSQL Serverless v2 dan disediakan	Aurora Serverless v1
	`nt` dan `disable-http-endpoint` operasi.	• AWS CLI— Gunakan `modify-db-cluster` operasi dengan `--no-enable-http-endpoint` opsi `--enable-http-endpoint` atau, sebagaimana berlaku.
CloudTrail acara	Peristiwa dari panggilan API Data adalah peristiwa data. Peristiwa ini secara otomatis dikecualikan dalam jejak secara default. Untuk informasi selengkapnya, lihat the section called “Menyertakan peristiwa API Data dalam suatu CloudTrail jejak”.	Peristiwa dari panggilan API Data adalah peristiwa manajemen. Peristiwa ini secara otomatis disertakan dalam jejak secara default. Untuk informasi selengkapnya, lihat the section called “Mengecualikan peristiwa Data API dari CloudTrail jejak (Aurora Serverless v1hanya)”.
Dukungan multistatement	Multistatement tidak didukung. Dalam hal ini, Data API melempar`ValidationException: Multistatements aren't supported .`	Untuk Aurora PostgreSQL, multistatement hanya mengembalikan respons kueri pertama. Untuk Aurora MySQL, multistatement tidak didukung.
BatchExecuteStatement	Objek bidang yang dihasilkan dalam hasil pembaruan kosong.	Objek bidang yang dihasilkan dalam hasil pembaruan mencakup nilai yang disisipkan.
Eksekusi QL	Tidak didukung	Telah usang

Perbedaan	Aurora PostgreSQL Serverless v2 dan disediakan	Aurora Serverless v1
ExecuteStatement	`ExecuteStatement` tidak mendukung pengambilan kolom array multidimensi. Dalam hal ini, Data API melempar`UnsupportedResultException` . Data API tidak mendukung beberapa tipe data, seperti tipe geometris dan moneter. Dalam hal ini, Data API melempar`UnsupportedResultException: The result contains the unsupported data type` *`data_type`* . Hanya jenis berikut yang didukung: • `BOOL` • `BYTEA` • `DATE` • `DECIMAL`, `NUMERIC` • `FLOAT8`, `DOUBLE PRECISION` • `INT`, `INT4`, `SERIAL` • `INT2`, `SMALLINT`, `SMALLSERIAL` • `INT8`, `BIGINT`, `BIGSERIAL` • `JSONB`, `JSON`	`ExecuteStatement` mendukung pengambilan kolom array multidimensi dan semua tipe data lanjutan.

Perbedaan	Aurora PostgreSQL Serverless v2 dan disediakan	Aurora Serverless v1
	• REAL, FLOAT • TEXT, CHAR(N), VARCHAR, NAME • TIME • TIMESTAMP • UUID • VECTOR Hanya tipe array berikut yang didukung: • BOOL[], BIT[] • DATE[] • DECIMAL[] , NUMERIC[] • FLOAT8[], DOUBLE PRECISION[] • INT[], INT4[] • INT2[] • INT8[], BIGINT[] • JSON[] • REAL[], FLOAT[] • TEXT[], CHAR(N)[] , VARCHAR[] , NAME[] • TIME[] • TIMESTAMP[] • UUID[]	

Mengotorisasi akses ke RDS Data API

Pengguna dapat menjalankan operasi RDS Data API (Data API) hanya jika mereka berwenang untuk melakukannya. Anda dapat memberikan izin kepada pengguna untuk menggunakan Data API dengan melampirkan kebijakan AWS Identity and Access Management (IAM) yang menentukan hak istimewa mereka. Anda juga dapat melampirkan kebijakan ini pada peran jika Anda menggunakan peran IAM. Kebijakan AWS terkelola`AmazonRDSDataFullAccess`, termasuk izin untuk API Data.

Kebijakan `AmazonRDSDataFullAccess` juga mencakup izin bagi pengguna untuk mendapatkan nilai sebuah rahasia dari AWS Secrets Manager. Pengguna perlu menggunakan Secrets Manager untuk menyimpan rahasia yang dapat mereka gunakan dalam panggilan mereka ke Data API. Menggunakan rahasia berarti bahwa pengguna tidak perlu menyertakan kredensi database untuk sumber daya yang mereka targetkan dalam panggilan mereka ke Data API. Data API secara transparan memanggil Secrets Manager, yang memungkinkan (atau menyangkal) permintaan pengguna untuk rahasia tersebut. Untuk informasi tentang menyiapkan rahasia yang akan digunakan dengan Data API, lihatMenyimpan kredensial database di AWS Secrets Manager.

`AmazonRDSDataFullAccess`Kebijakan ini menyediakan akses lengkap (melalui Data API) ke sumber daya. Anda dapat mempersempit cakupan dengan menentukan kebijakan Anda sendiri yang menentukan Amazon Resource Name (ARN) sumber daya.

Misalnya, kebijakan berikut menunjukkan contoh izin minimum yang diperlukan bagi pengguna untuk mengakses Data API untuk klaster DB yang diidentifikasi oleh ARN-nya. Kebijakan tersebut mencakup izin yang diperlukan untuk mengakses Secrets Manager dan mendapatkan otorisasi ke instans basis data untuk pengguna.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
```

```
            "Action": [
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:prod"
        }
    ]
}
```

Kami menyarankan agar Anda menggunakan ARN khusus untuk elemen "Sumber Daya" dalam pernyataan kebijakan Anda (seperti yang ditunjukkan dalam contoh), bukan karakter wildcard (*).

Bekerja dengan otorisasi berbasis tag

RDS Data API (Data API) dan Secrets Manager keduanya mendukung otorisasi berbasis tag. Tag adalah pasangan nilai kunci yang melabeli sumber daya, seperti klaster RDS, dengan nilai string tambahan, misalnya:

- `environment:production`
- `environment:development`

Anda dapat menerapkan tag ke sumber daya Anda untuk alokasi biaya, dukungan operasi, kontrol akses, dan banyak alasan lainnya. (Jika Anda belum memiliki tag pada sumber daya dan ingin menerapkannya, Anda dapat mempelajari lebih lanjut di Memberi tag pada sumber daya Amazon RDS.) Anda dapat menggunakan tag dalam pernyataan kebijakan Anda untuk membatasi akses ke klaster RDS yang dilabeli tag ini. Sebagai contoh, klaster basis data Aurora mungkin memiliki tag yang mengidentifikasi lingkungannya sebagai produksi atau pengembangan.

Contoh-contoh berikut menunjukkan bagaimana Anda dapat menggunakan tag dalam pernyataan kebijakan Anda. Pernyataan ini mengharuskan klaster dan rahasia yang diteruskan dalam permintaan API Data memiliki tag `environment:production`.

Begini cara kebijakan diterapkan: Saat pengguna melakukan panggilan menggunakan Data API, permintaan akan dikirim ke layanan. Data API pertama-tama memverifikasi bahwa ARN cluster yang diteruskan dalam permintaan ditandai dengan. `environment:production` API ini kemudian memanggil Secrets Manager untuk mengambil nilai rahasia pengguna dalam permintaan tersebut. Secrets Manager juga memverifikasi bahwa rahasia pengguna ditandai dengan

`environment:production`. Jika demikian, API Data kemudian menggunakan nilai yang diambil untuk kata sandi basis data pengguna. Terakhir, jika itu juga benar, permintaan API Data berhasil diinvokasi untuk pengguna.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SecretsManagerDbCredentialsAccess",
            "Effect": "Allow",
            "Action": [
                 "secretsmanager:GetSecretValue"
               ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*",
            "Condition": {
                    "StringEquals": {
                        "aws:ResourceTag/environment": [
                                        "production"
                                       ]
                     }
             }
        },
        {
            "Sid": "RDSDataServiceAccess",
            "Effect": "Allow",
            "Action": [
                  "rds-data:*"
               ],
            "Resource": "arn:aws:rds:us-east-2:111122223333:cluster:*",
            "Condition": {
                    "StringEquals": {
                        "aws:ResourceTag/environment": [
                                        "production"
                                       ]
                     }
             }
         }
     ]
}
```

Contoh menunjukkan tindakan terpisah untuk `rds-data` dan `secretsmanager` untuk Data API dan Secrets Manager. Namun, Anda dapat menggabungkan tindakan dan menentukan kondisi

tag dengan berbagai cara untuk mendukung kasus penggunaan spesifik Anda. Lihat informasi selengkapnya di Menggunakan Kebijakan Berbasis Identitas (Kebijakan IAM) untuk Amazon DynamoDB.

Di elemen "Kondisi" kebijakan, Anda dapat memilih kunci tag dari beberapa pilihan berikut:

- `aws:TagKeys`
- `aws:ResourceTag/${TagKey}`

Untuk mempelajari lebih lanjut tentang tag sumber daya dan cara menggunakan `aws:TagKeys`, lihat Mengontrol akses ke sumber daya AWS menggunakan tag sumber daya.

> **Note**
>
> Baik Data API dan AWS Secrets Manager otorisasi pengguna. Jika Anda tidak memiliki izin untuk semua tindakan yang ditentukan dalam kebijakan, Anda akan mendapatkan kesalahan `AccessDeniedException`.

Menyimpan kredensial database di AWS Secrets Manager

Saat Anda memanggil RDS Data API (Data API), Anda meneruskan kredensil untuk klaster Aurora DB dengan menggunakan rahasia di Secrets Manager. Untuk memberikan kredensial dengan cara ini, Anda menentukan nama rahasia atau Amazon Resource Name (ARN) rahasia tersebut.

Menyimpan kredensial klaster basis data dalam rahasia

1. Gunakan Secrets Manager untuk membuat rahasia yang berisi kredensial untuk klaster basis data Aurora.

   Lihat petunjuknya di Membuat rahasia basis data dalam Panduan Pengguna AWS Secrets Manager.

2. Gunakan konsol Secrets Manager untuk melihat detail rahasia yang Anda buat, atau jalankan perintah `aws secretsmanager describe-secret` AWS CLI.

   Catat nama dan ARN rahasia. Anda dapat menggunakannya dalam panggilan ke Data API.

Lihat informasi selengkapnya tentang menggunakan Secrets Manager di Panduan Pengguna AWS Secrets Manager.

Untuk memahami bagaimana Amazon Aurora mengelola Identity and Access Management, lihat Bagaimana Amazon Aurora bekerja dengan IAM.

Lihat informasi selengkapnya tentang pembuatan kebijakan IAM di Membuat Kebijakan IAM dalam Panduan Pengguna IAM. Lihat informasi tentang cara menambahkan kebijakan IAM ke pengguna di Menambahkan dan Menghapus Izin Identitas IAM dalam Panduan Pengguna IAM.

Mengaktifkan API Data RDS

Untuk menggunakan RDS Data API (Data API), aktifkan untuk cluster Aurora DB Anda. Anda dapat mengaktifkan Data API saat membuat atau memodifikasi cluster DB.

> **Note**
>
> Untuk Aurora PostgreSQL, Data API didukung dengan,, dan database yang disediakan. Aurora Serverless v2 Aurora Serverless v1 Untuk Aurora MySQL, Data API hanya didukung dengan database. Aurora Serverless v1

Topik

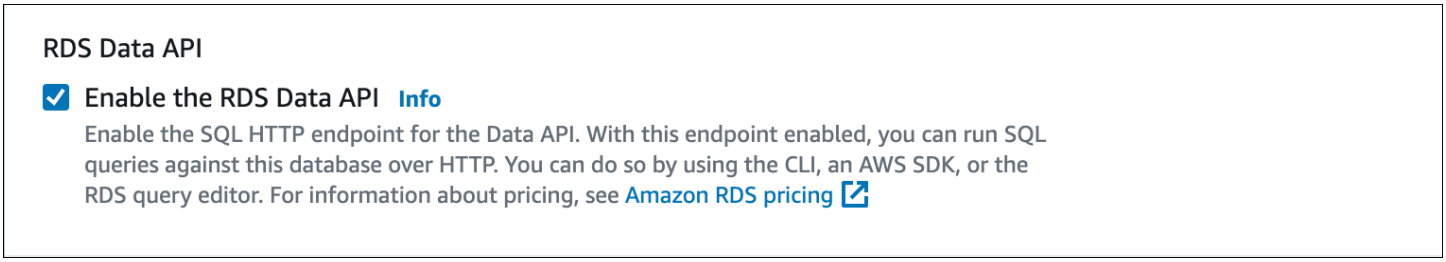
- Mengaktifkan RDS Data API saat Anda membuat database
- Mengaktifkan API Data RDS pada database yang ada

Mengaktifkan RDS Data API saat Anda membuat database

Saat Anda membuat database yang mendukung RDS Data API (Data API), Anda dapat mengaktifkan fitur ini. Prosedur berikut menjelaskan cara melakukannya ketika Anda menggunakanAWS Management Console,AWS CLI, atau RDS API.

Konsol

Untuk mengaktifkan Data API saat Anda membuat cluster DB, pilih kotak centang Aktifkan API Data RDS di bagian Konektivitas pada halaman Buat database, seperti pada tangkapan layar berikut.



Untuk petunjuk tentang cara membuat database, lihat berikut ini:

- Untuk Aurora PostgreSQL Serverless v2 dan database yang disediakan - Membuat klaster DB Amazon Aurora
- Untuk Aurora Serverless v1 - Membuat klaster DB Aurora Serverless v1

AWS CLI

Untuk mengaktifkan Data API saat Anda membuat cluster Aurora DB, jalankan create-db-clusterAWS CLIperintah dengan opsi. `--enable-http-endpoint`

Contoh berikut membuat cluster Aurora PostgreSQL DB dengan Data API diaktifkan.

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
    --db-cluster-identifier my_pg_cluster \
    --engine aurora-postgresql \
    --enable-http-endpoint
```

Untuk Windows:

```
aws rds create-db-cluster ^
    --db-cluster-identifier my_pg_cluster ^
    --engine aurora-postgresql ^
    --enable-http-endpoint
```

API RDS

Untuk mengaktifkan Data API saat Anda membuat cluster Aurora DB, gunakan operasi CreateDBCluster dengan nilai parameter yang disetel ke. `EnableHttpEndpoint` `true`

Mengaktifkan API Data RDS pada database yang ada

Anda dapat memodifikasi cluster DB yang mendukung RDS Data API (Data API) untuk mengaktifkan atau menonaktifkan fitur ini.

Topik

- Mengaktifkan atau menonaktifkan Data API (Aurora PostgreSQL Serverless v2 dan disediakan)
- Mengaktifkan atau menonaktifkan Data API (hanya) Aurora Serverless v1

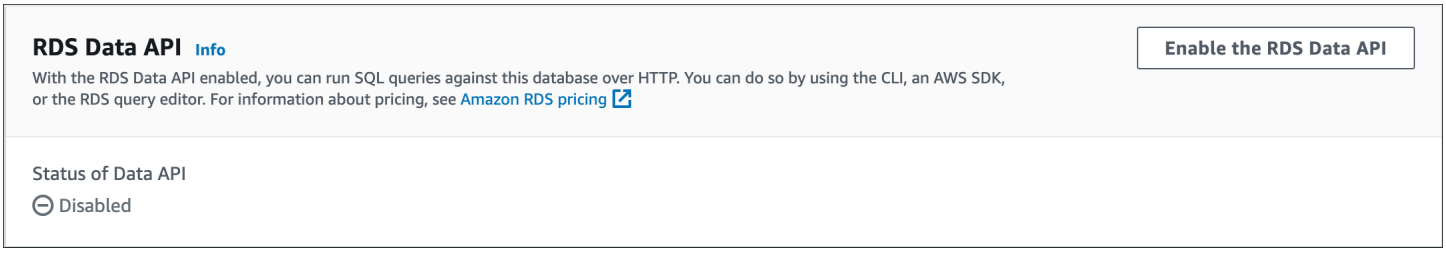
Mengaktifkan atau menonaktifkan Data API (Aurora PostgreSQL Serverless v2 dan disediakan)

Gunakan prosedur berikut untuk mengaktifkan atau menonaktifkan Data API di Aurora PostgreSQL Serverless v2 dan database yang disediakan. Untuk mengaktifkan atau menonaktifkan Data API pada Aurora Serverless v1 database, gunakan prosedur dithe section called “Mengaktifkan atau menonaktifkan Data API (hanya) Aurora Serverless v1”.

Konsol

Anda dapat mengaktifkan atau menonaktifkan Data API dengan menggunakan konsol RDS untuk cluster DB yang mendukung fitur ini. Untuk melakukannya, buka halaman detail cluster dari database tempat Anda ingin mengaktifkan atau menonaktifkan Data API, dan pada tab Konektivitas & keamanan, buka bagian API Data RDS. Bagian ini menampilkan status Data API, dan memungkinkan Anda untuk mengaktifkan atau menonaktifkannya.

Screenshot berikut menunjukkan bahwa RDS Data API tidak diaktifkan.



AWS CLI

Untuk mengaktifkan atau menonaktifkan Data API pada database yang ada, jalankan disable-http-endpointAWS CLIperintah enable-http-endpointor, dan tentukan ARN cluster DB Anda.

Contoh berikut memungkinkan Data API.

Untuk Linux, macOS, atau Unix:

```
aws rds enable-http-endpoint \
    --resource-arn cluster_arn
```

Untuk Windows:

```
aws rds enable-http-endpoint ^
    --resource-arn cluster_arn
```

API RDS

Untuk mengaktifkan atau menonaktifkan Data API pada database yang ada, gunakan EnableHttpEndpointdan DisableHttpEndpointoperasi.

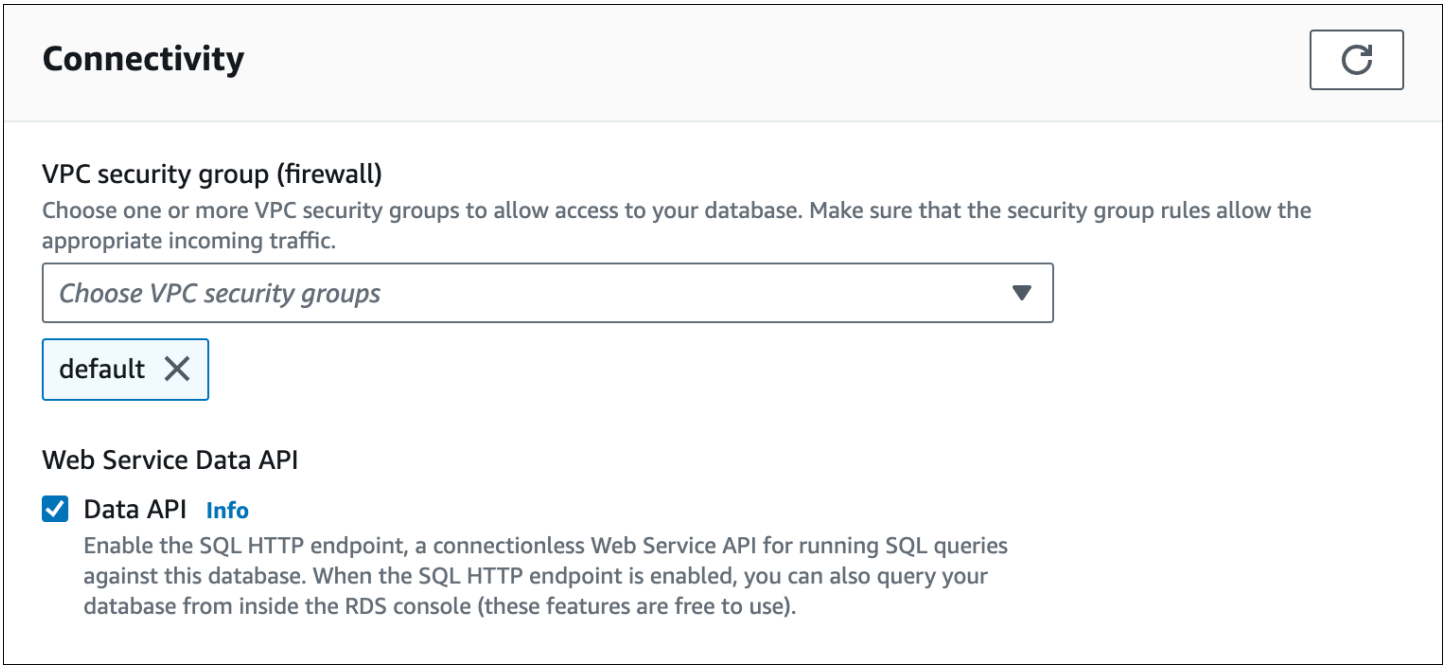
Mengaktifkan atau menonaktifkan Data API (hanya) Aurora Serverless v1

Gunakan prosedur berikut untuk mengaktifkan atau menonaktifkan Data API pada Aurora Serverless v1 database yang ada. Untuk mengaktifkan atau menonaktifkan Data API pada Aurora PostgreSQL Serverless v2 dan database yang disediakan, gunakan prosedur di. the section called “Mengaktifkan atau menonaktifkan Data API”

Konsol

Saat memodifikasi cluster Aurora Serverless v1 DB, Anda mengaktifkan Data API di bagian Konektivitas konsol RDS.

Tangkapan layar berikut menunjukkan API Data yang diaktifkan saat memodifikasi cluster Aurora DB.



Untuk petunjuk tentang cara memodifikasi cluster Aurora Serverless v1 DB, lihatMemodifikasi klaster DB Aurora Serverless v1.

AWS CLI

Untuk mengaktifkan atau menonaktifkan Data API, jalankan modify-db-clusterAWS CLIperintah, dengan `--enable-http-endpoint` atau`--no-enable-http-endpoint`, sebagaimana berlaku.

Contoh berikut memungkinkan Data API aktif`sample-cluster`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
    --db-cluster-identifier sample-cluster \
    --enable-http-endpoint
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier sample-cluster ^
    --enable-http-endpoint
```

API RDS

Untuk mengaktifkan Data API, gunakan operasi ModifyDBCluster, dan tetapkan nilai ke atau, sebagaimana berlaku. `EnableHttpEndpoint` `true` `false`

Membuat endpoint Amazon VPC untuk RDS Data API () AWS PrivateLink

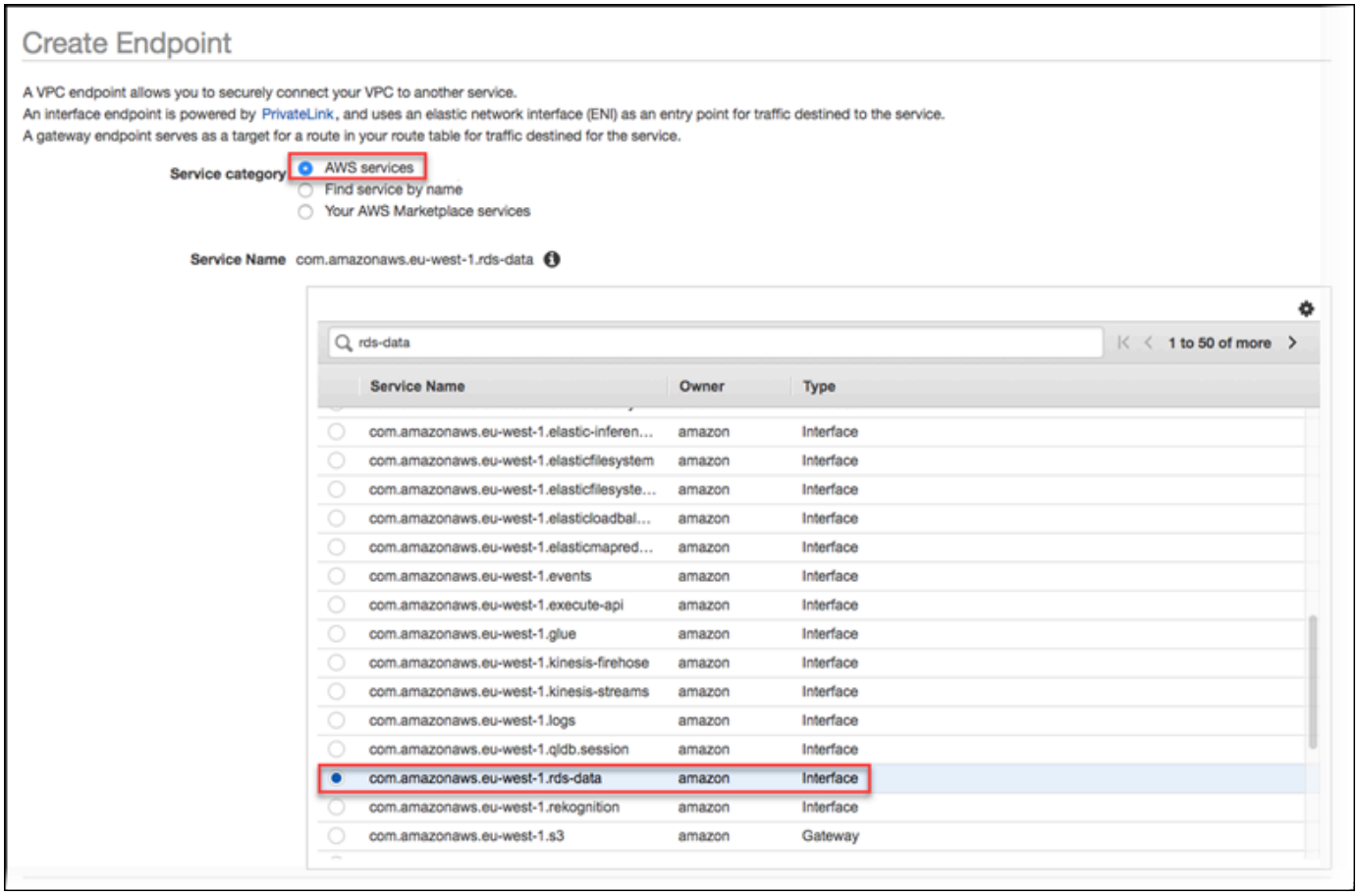
Amazon VPC memungkinkan Anda untuk meluncurkan sumber daya AWS, seperti klaster dan aplikasi basis data Aurora, ke dalam cloud privat virtual (VPC). AWS PrivateLink menyediakan konektivitas privat antara VPC dan layanan AWS dengan keamanan tinggi di jaringan Amazon. Dengan menggunakan AWS PrivateLink, Anda dapat membuat titik akhir Amazon VPC, yang memungkinkan Anda terhubung ke layanan di berbagai akun dan VPC berdasarkan Amazon VPC. Lihat informasi selengkapnya tentang AWS PrivateLink di Layanan Titik Akhir VPC (AWS PrivateLink) dalam Panduan Pengguna Amazon Virtual Private Cloud.

Anda dapat memanggil RDS Data API (Data API) dengan titik akhir Amazon VPC. Menggunakan endpoint Amazon VPC menjaga lalu lintas antar aplikasi di Amazon VPC dan Data API di AWS jaringan, tanpa menggunakan alamat IP publik. Titik akhir Amazon VPC dapat membantu Anda memenuhi persyaratan kepatuhan dan peraturan yang berkaitan dengan membatasi konektivitas internet publik. Misalnya, jika Anda menggunakan titik akhir VPC Amazon, Anda dapat menjaga lalu lintas antara aplikasi yang berjalan pada instans Amazon EC2 dan Data API di VPC yang berisi aplikasi tersebut.

Setelah membuat titik akhir Amazon VPC, Anda dapat mulai menggunakannya tanpa membuat kode atau perubahan konfigurasi apa pun di aplikasi Anda.

Untuk membuat titik akhir VPC Amazon untuk Data API

1. Masuk ke AWS Management Console dan buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
2. Pilih Titik Akhir, lalu pilih Buat Titik Akhir.
3. Di halaman Buat Titik Akhir, untuk Kategori layanan, pilih Layanan AWS. Untuk Nama Layanan, pilih rds-data.



4. Untuk VPC, pilih VPC tempat membuat titik akhir.

   Pilih VPC yang berisi aplikasi yang membuat panggilan API Data.

5. Untuk Subnet, pilih subnet untuk setiap Zona Ketersediaan (AZ) yang digunakan oleh layanan AWS yang menjalankan aplikasi Anda.

   Untuk membuat titik akhir Amazon VPC, tentukan rentang alamat IP pribadi yang dapat mengakses titik akhir. Untuk melakukannya, pilih subnet untuk setiap Zona Ketersediaan. Hal tersebut membatasi titik akhir VPC ke rentang alamat IP pribadi khusus untuk setiap Zona Ketersediaan dan membuat titik akhir Amazon VPC di setiap Zona Ketersediaan.

6. Untuk Aktifkan nama DNS, pilih Aktifkan untuk titik akhir ini.

   DNS Pribadi menyelesaikan nama host DNS API Data standar (`https://rds-data.`*`region`*`.amazonaws.com`) ke alamat IP pribadi yang dikaitkan dengan nama host DNS khusus untuk titik akhir Amazon VPC Anda. Akibatnya, Anda dapat mengakses titik akhir VPC Data API menggunakan AWS CLI atau AWS SDK tanpa membuat perubahan kode atau konfigurasi apa pun untuk memperbarui URL titik akhir Data API.

7. Untuk Grup keamanan, pilih satu grup keamanan untuk dikaitkan dengan titik akhir Amazon VPC.

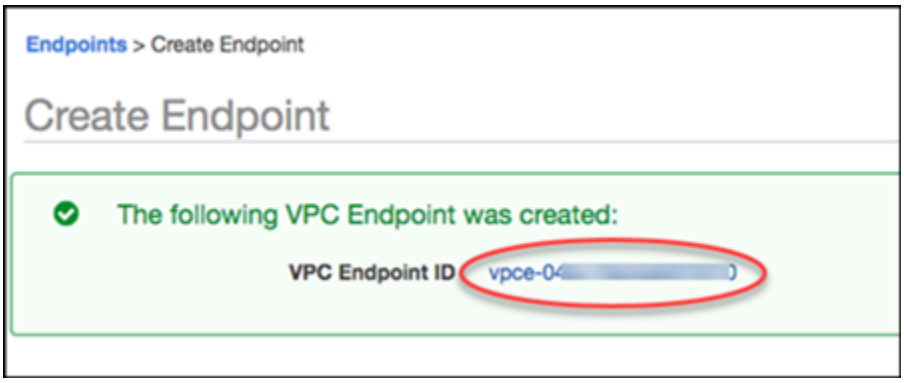
   Pilih grup keamanan yang mengizinkan akses ke layanan AWS yang menjalankan aplikasi Anda. Misalnya, jika instans Amazon EC2 menjalankan aplikasi Anda, pilih grup keamanan yang mengizinkan akses ke instans Amazon EC2. Grup keamanan memungkinkan Anda mengontrol lalu lintas ke titik akhir Amazon VPC dari sumber daya di VPC Anda.

8. Untuk Kebijakan, pilih Akses Penuh untuk mengizinkan siapa pun di dalam Amazon VPC mengakses API Data melalui titik akhir ini. Atau, pilih Khusus untuk menentukan kebijakan yang membatasi akses.

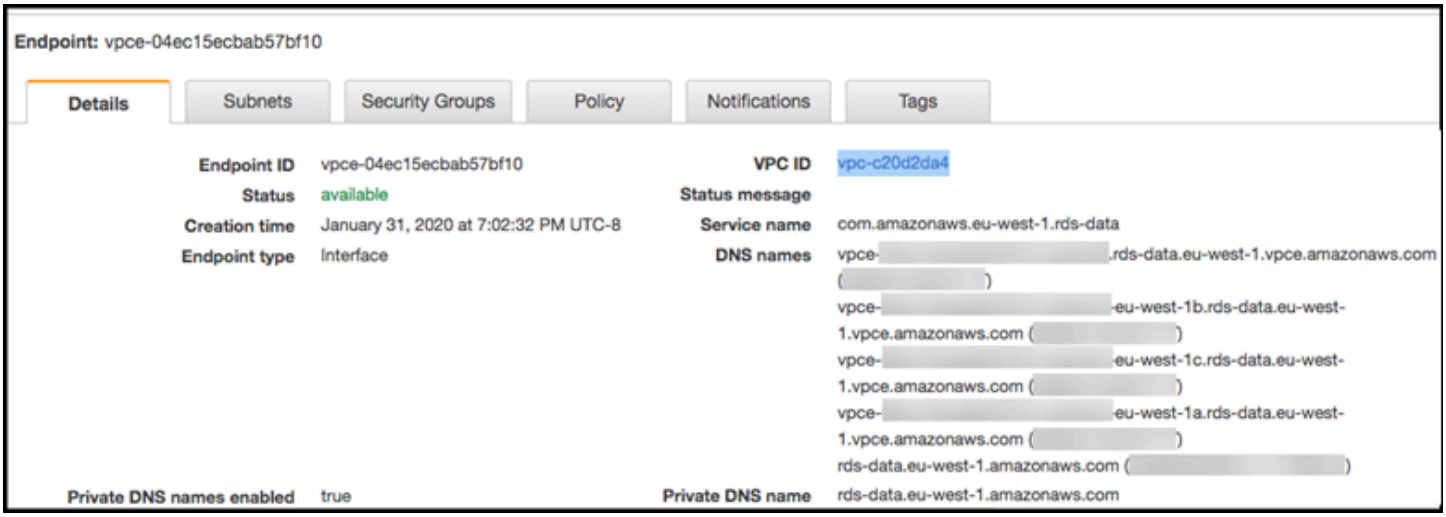
   Jika Anda memilih Khusus, masukkan kebijakan di alat pembuatan kebijakan.

9. Pilih Create endpoint.

Setelah titik akhir dibuat, pilih tautan di AWS Management Console untuk melihat detail titik akhir.



Tab Detail titik akhir menunjukkan nama host DNS yang dibuat saat membuat titik akhir Amazon VPC.



Anda dapat menggunakan titik akhir standar (`rds-data.`*`region`*`.amazonaws.com`) atau salah satu titik akhir khusus VPC untuk memanggil API Data dalam Amazon VPC. Titik akhir API Data standar secara otomatis merutekan ke titik akhir Amazon VPC. Perutean ini terjadi karena nama host DNS Pribadi diaktifkan saat titik akhir Amazon VPC dibuat.

Saat Anda menggunakan titik akhir VPC Amazon dalam panggilan Data API, semua lalu lintas antara aplikasi dan Data API tetap berada di VPC Amazon yang berisi mereka. Anda dapat menggunakan titik akhir Amazon VPC untuk semua jenis panggilan API Data. Untuk informasi tentang memanggil Data API, lihatMemanggil RDS Data API.

Memanggil RDS Data API

Dengan RDS Data API (Data API) diaktifkan pada cluster Aurora DB Anda, Anda dapat menjalankan pernyataan SQL pada cluster Aurora DB dengan menggunakan Data API atau file. AWS CLI Data API mendukung bahasa pemrograman yang didukung oleh AWS SDK. Lihat informasi selengkapnya di Alat untuk membangun di AWS.

> **Note**
>
> Saat ini, Data API tidak mendukung array Universal Unique Identifiers (UUID).

Data API menyediakan operasi berikut untuk melakukan pernyataan SQL.

Operasi API Data	Perintah AWS CLI	Deskripsi
ExecuteStatement	aws rds-data execute-statement	Menjalankan pernyataan SQL pada basis data.
BatchExecuteStatement	aws rds-data batch-execute-statement	Menjalankan pernyataan SQL batch pada susunan data untuk pembaruan massal dan operasi penyisipan. Anda dapat menjalankan pernyataan bahasa manipulasi data (DML) dengan susunan set parameter. Pernyataan SQL batch dapat memberikan peningkatan kinerja yang signifikan atas pernyataan penyisipan dan pembaruan individu.

Anda dapat menggunakan operasi mana pun untuk menjalankan pernyataan SQL individual atau untuk menjalankan transaksi. Untuk transaksi, Data API menyediakan operasi berikut.

Operasi API Data	Perintah AWS CLI	Deskripsi
BeginTransaction	aws rds-data begin-transaction	Memulai transaksi SQL.
CommitTransaction	aws rds-data commit-transaction	Mengakhiri transaksi SQL dan menerapkan perubahan.
RollbackTransaction	aws rds-data rollback-transaction	Melakukan pembatalan transaksi.

Operasi untuk melakukan pernyataan SQL dan mendukung transaksi memiliki parameter API Data umum dan opsi AWS CLI berikut. Beberapa operasi mendukung parameter atau opsi lain.

Parameter operasi API Data	Opsi perintah AWS CLI	Diperlukan	Deskripsi
`resourceArn`	`--resource-arn`	Ya	Nama Sumber Daya Amazon (ARN) dari cluster Aurora DB.
`secretArn`	`--secret-arn`	Ya	Nama atau ARN rahasia yang memungkinkan akses ke klaster basis data.

Anda dapat menggunakan parameter dalam panggilan API Data ke `ExecuteStatement` dan `BatchExecuteStatement`, serta ketika Anda menjalankan Perintah AWS CLI `execute-statement` dan `batch-execute-statement`. Untuk menggunakan parameter, tentukan pasangan nama-nilai di tipe data `SqlParameter`. Tentukan nilai dengan tipe data `Field`. Tabel berikut memetakan tipe data Java Database Connectivity (JDBC) ke tipe data yang Anda tentukan dalam panggilan API Data.

Tipe data JDBC	Tipe data API Data
`INTEGER, TINYINT, SMALLINT, BIGINT`	`LONG` (atau `STRING`)
`FLOAT, REAL, DOUBLE`	`DOUBLE`
`DECIMAL`	`STRING`
`BOOLEAN, BIT`	`BOOLEAN`
`BLOB, BINARY, LONGVARBINARY, VARBINARY`	`BLOB`
`CLOB`	`STRING`
Tipe lainnya (termasuk tipe terkait tanggal dan waktu)	`STRING`

> **Note**
>
> Anda dapat menentukan tipe data `LONG` atau `STRING` dalam panggilan API Data Anda untuk nilai `LONG` yang dihasilkan oleh basis data. Kami menyarankan Anda melakukannya untuk menghindari kehilangan presisi untuk jumlah yang sangat besar, yang dapat terjadi ketika Anda bekerja dengan JavaScript.

Tipe tertentu, seperti `DECIMAL` dan`TIME`, memerlukan petunjuk agar Data API meneruskan `String` nilai ke database sebagai tipe yang benar. Untuk menggunakan petunjuk, sertakan nilai untuk `typeHint` di tipe data `SqlParameter`. Berikut adalah nilai-nilai yang mungkin untuk `typeHint`:

- `DATE` – Nilai parameter `String` yang sesuai dikirim sebagai objek tipe `DATE` ke basis data. Format yang diterima adalah `YYYY-MM-DD`.
- `DECIMAL` – Nilai parameter `String` yang sesuai dikirim sebagai objek tipe `DECIMAL` ke basis data.
- `JSON` – Nilai parameter `String` yang sesuai dikirim sebagai objek tipe `JSON` ke basis data.
- `TIME` – Nilai parameter `String` yang sesuai dikirim sebagai objek tipe `TIME` ke basis data. Format yang diterima adalah `HH:MM:SS[.FFF]`.
- `TIMESTAMP` – Nilai parameter `String` yang sesuai dikirim sebagai objek tipe `TIMESTAMP` ke basis data. Format yang diterima adalah `YYYY-MM-DD HH:MM:SS[.FFF]`.
- `UUID` – Nilai parameter `String` yang sesuai dikirim sebagai objek tipe `UUID` ke basis data.

> **Note**
>
> Untuk Amazon Aurora PostgreSQL, Data API selalu menampilkan tipe data Aurora PostgreSQL di zona waktu UTC. `TIMESTAMPTZ`

Memanggil RDS Data API dengan AWS CLI

Anda dapat memanggil RDS Data API (Data API) menggunakan file. AWS CLI

Contoh berikut menggunakan AWS CLI for Data API. Lihat informasi selengkapnya di Referensi AWS CLI untuk API Data.

Dalam setiap contoh, ganti Amazon Resource Name (ARN) untuk cluster DB dengan ARN untuk cluster Aurora DB Anda. Selain itu, ganti ARN rahasia dengan ARN rahasia di Secrets Manager yang mengizinkan akses ke klaster basis data.

> **Note**
>
> AWS CLI dapat memformat respons di JSON.

Topik

- Memulai transaksi SQL
- Menjalankan pernyataan SQL
- Menjalankan pernyataan SQL batch pada susunan data
- Menerapkan transaksi SQL
- Membatalkan transaksi SQL

Memulai transaksi SQL

Anda dapat memulai transaksi SQL menggunakan perintah CLI `aws rds-data begin-transaction`. Panggilan ini menghasilkan pengidentifikasi transaksi.

> **Important**
>
> Waktu transaksi habis jika tidak ada panggilan yang menggunakan ID transaksinya dalam tiga menit. Jika waktu transaksi habis sebelum diterapkan, transaksi akan dibatalkan secara otomatis.
> Pernyataan bahasa definisi data (DDL) di dalam transaksi menyebabkan penerapan implisit. Kami menyarankan agar Anda menjalankan setiap pernyataan DDL dalam perintah `execute-statement` terpisah dengan opsi `--continue-after-timeout`.

Selain opsi umum, tentukan opsi `--database` yang menyediakan nama basis data.

Misalnya, perintah CLI berikut memulai transaksi SQL.

Untuk Linux, macOS, atau Unix:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret"
```

Untuk Windows:

```
aws rds-data begin-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret"
```

Berikut adalah contoh respons tersebut.

```
{
    "transactionId": "ABC1234567890xyz"
}
```

Menjalankan pernyataan SQL

Anda dapat menjalankan pernyataan SQL menggunakan perintah CLI `aws rds-data execute-statement`.

Anda dapat menjalankan pernyataan SQL dalam transaksi dengan menentukan pengidentifikasi transaksi dengan opsi `--transaction-id`. Anda dapat memulai transaksi menggunakan perintah CLI `aws rds-data begin-transaction`. Anda dapat mengakhiri dan menerapkan transaksi menggunakan perintah CLI `aws rds-data commit-transaction`.

> **Important**
>
> Jika Anda tidak menentukan opsi `--transaction-id`, perubahan yang dihasilkan dari panggilan akan diterapkan secara otomatis.

Selain opsi umum, tentukan opsi-opsi berikut:

- `--sql` (wajib) – Pernyataan SQL untuk dijalankan pada klaster basis data.
- `--transaction-id` (opsional) – Pengidentifikasi transaksi yang dimulai menggunakan perintah CLI `begin-transaction`. Tentukan ID transaksi yang ingin Anda sertakan pernyataan SQL-nya.

- `--parameters` (opsional) – Parameter untuk pernyataan SQL.
- `--include-result-metadata | --no-include-result-metadata` (opsional) – Nilai yang menunjukkan apakah metadata disertakan dalam hasil. Standarnya adalah `--no-include-result-metadata`.
- `--database` (opsional) – Nama basis data.

  Opsi `--database` mungkin tidak berfungsi ketika Anda menjalankan pernyataan SQL setelah menjalankan `--sql "use database_name;"` di permintaan sebelumnya. Kami merekomendasikan agar Anda menggunakan `--database` opsi, bukan menjalankan pernyataan `--sql "use database_name;"`.
- `--continue-after-timeout | --no-continue-after-timeout` (opsional) – Nilai yang menunjukkan apakah pernyataan dilanjutkan setelah waktu panggilan habis. Standarnya adalah `--no-continue-after-timeout`.

  Untuk pernyataan bahasa definisi data (DDL), kami merekomendasikan agar Anda terus menjalankan pernyataan setelah waktu panggilan habis untuk menghindari kesalahan dan kemungkinan struktur data rusak.
- `--format-records-as "JSON"|"NONE"` – Nilai opsional yang menentukan apakah hasil yang ditetapkan akan diformat sebagai string JSON. Default-nya adalah `"NONE"`. Lihat informasi penggunaan tentang pemrosesan set hasil JSON di Memproses hasil kueri dalam format JSON.

Klaster basis data menghasilkan respons untuk panggilan.

> **Note**
>
> Batas ukuran respons adalah 1 MiB. Jika panggilan menghasilkan data respons dengan ukuran lebih dari 1 MiB, panggilan tersebut akan diakhiri.
> UntukAurora Serverless v1, jumlah maksimum permintaan per detik adalah 1.000. Untuk semua database lain yang didukung, tidak ada batasan.

Misalnya, perintah CLI berikut menjalankan satu pernyataan SQL dan menghilangkan metadata dalam hasil (default).

Untuk Linux, macOS, atau Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "select * from mytable"
```

Untuk Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "select * from mytable"
```

Berikut adalah contoh respons tersebut.

```
{
    "numberOfRecordsUpdated": 0,
    "records": [
        [
            {
                "longValue": 1
            },
            {
                "stringValue": "ValueOne"
            }
        ],
        [
            {
                "longValue": 2
            },
            {
                "stringValue": "ValueTwo"
            }
        ],
        [
            {
                "longValue": 3
            },
            {
                "stringValue": "ValueThree"
            }
```

```
        ]
    ]
}
```

Perintah CLI berikut menjalankan satu pernyataan SQL dalam transaksi dengan menentukan opsi `--transaction-id`.

Untuk Linux, macOS, atau Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

Untuk Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "update mytable set quantity=5 where id=201" --transaction-id "ABC1234567890xyz"
```

Berikut adalah contoh respons tersebut.

```
{
    "numberOfRecordsUpdated": 1
}
```

Perintah CLI berikut menjalankan satu pernyataan SQL dengan parameter.

Untuk Linux, macOS, atau Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

Untuk Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" --parameters "[{\"name\": \"id\",
 \"value\": {\"longValue\": 1}},{\"name\": \"val\", \"value\": {\"stringValue\":
 \"value1\"}}]"
```

Berikut adalah contoh respons tersebut.

```
{
    "numberOfRecordsUpdated": 1
}
```

Perintah CLI berikut menjalankan pernyataan SQL bahasa definisi data (DDL). Pernyataan DDL mengganti nama kolom `job` menjadi kolom `role`.

> **Important**
>
> Untuk pernyataan DDL, kami merekomendasikan agar Anda terus menjalankan pernyataan tersebut setelah waktu panggilan habis. Ketika pernyataan DDL dihentikan sebelum selesai dijalankan, hal ini dapat mengakibatkan kesalahan dan kemungkinan struktur data rusak. Untuk terus menjalankan pernyataan setelah waktu panggilan habis, tentukan opsi `--continue-after-timeout`.

Untuk Linux, macOS, atau Unix:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

Untuk Windows:

```
aws rds-data execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
```

```
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "alter table mytable change column job role varchar(100)" --continue-after-
timeout
```

Berikut adalah contoh respons tersebut.

```
{
    "generatedFields": [],
    "numberOfRecordsUpdated": 0
}
```

> **Note**
>
> Data `generatedFields` tidak didukung oleh Aurora PostgreSQL. Untuk mendapatkan nilai-nilai bidang yang dihasilkan, gunakan klausa `RETURNING`. Lihat informasi selengkapnya di Mengembalikan data dari baris yang diubah dalam dokumentasi PostgreSQL.

Menjalankan pernyataan SQL batch pada susunan data

Anda dapat menjalankan pernyataan SQL batch pada susunan data dengan menggunakan perintah CLI `aws rds-data batch-execute-statement`. Anda dapat menggunakan perintah ini untuk melakukan impor massal atau operasi pembaruan.

Anda dapat menjalankan pernyataan SQL dalam transaksi dengan menentukan pengidentifikasi transaksi dengan opsi `--transaction-id`. Anda dapat memulai transaksi menggunakan perintah CLI `aws rds-data begin-transaction`. Anda dapat mengakhiri dan menerapkan transaksi menggunakan perintah CLI `aws rds-data commit-transaction`.

> **Important**
>
> Jika Anda tidak menentukan opsi `--transaction-id`, perubahan yang dihasilkan dari panggilan akan diterapkan secara otomatis.

Selain opsi umum, tentukan opsi-opsi berikut:

- `--sql` (wajib) – Pernyataan SQL untuk dijalankan pada klaster basis data.

> **Tip**
>
> Untuk pernyataan yang kompatibel dengan MySQL, jangan sertakan titik koma di akhir parameter `--sql`. Titik koma di belakang dapat menyebabkan kesalahan sintaksis.

- `--transaction-id` (opsional) – Pengidentifikasi transaksi yang dimulai menggunakan perintah CLI `begin-transaction`. Tentukan ID transaksi yang ingin Anda sertakan pernyataan SQL-nya.
- `--parameter-set` (opsional) – Set parameter untuk operasi batch.
- `--database` (opsional) – Nama basis data.

Klaster basis data menghasilkan respons untuk panggilan.

> **Note**
>
> Tidak ada batas atas tetap pada jumlah set parameter. Namun, ukuran maksimum permintaan HTTP yang dikirimkan melalui Data API adalah 4 MiB. Jika permintaan melebihi batas ini, Data API mengembalikan kesalahan dan tidak memproses permintaan. Batas 4 MiB ini mencakup ukuran header HTTP dan notasi JSON dalam permintaan. Dengan demikian, jumlah set parameter yang dapat Anda sertakan tergantung pada kombinasi faktor, seperti ukuran pernyataan SQL dan ukuran setiap set parameter.
> Batas ukuran respons adalah 1 MiB. Jika panggilan menghasilkan data respons dengan ukuran lebih dari 1 MiB, panggilan tersebut akan diakhiri.
> UntukAurora Serverless v1, jumlah maksimum permintaan per detik adalah 1.000. Untuk semua database lain yang didukung, tidak ada batasan.

Misalnya, perintah CLI berikut menjalankan pernyataan SQL batch pada susunan data dengan satu set parameter.

Untuk Linux, macOS, atau Unix:

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" \
--sql "insert into mytable values (:id, :val)" \
```

```
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

Untuk Windows:

```
aws rds-data batch-execute-statement --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--database "mydb" --secret-arn "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret" ^
--sql "insert into mytable values (:id, :val)" ^
--parameter-sets "[[{\"name\": \"id\", \"value\": {\"longValue\": 1}},{\"name\":
 \"val\", \"value\": {\"stringValue\": \"ValueOne\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 2}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueTwo\"}}],
[{\"name\": \"id\", \"value\": {\"longValue\": 3}},{\"name\": \"val\", \"value\":
 {\"stringValue\": \"ValueThree\"}}]]"
```

> **Note**
>
> Jangan sertakan jeda baris di opsi `--parameter-sets`.

Menerapkan transaksi SQL

Dengan menggunakan perintah CLI `aws rds-data commit-transaction`, Anda dapat mengakhiri transaksi SQL yang Anda mulai dengan `aws rds-data begin-transaction` dan menerapkan perubahan.

Selain opsi umum, tentukan opsi-opsi berikut:

- `--transaction-id` (wajib) – Pengidentifikasi transaksi yang dimulai menggunakan perintah CLI `begin-transaction`. Tentukan ID transaksi yang ingin Anda akhiri dan terapkan.

Misalnya, perintah CLI berikut mengakhiri transaksi SQL dan menerapkan perubahan.

Untuk Linux, macOS, atau Unix:

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

Untuk Windows:

```
aws rds-data commit-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

Berikut adalah contoh respons tersebut.

```
{
    "transactionStatus": "Transaction Committed"
}
```

Membatalkan transaksi SQL

Dengan menggunakan perintah CLI `aws rds-data rollback-transaction`, Anda dapat membatalkan transaksi SQL yang Anda mulai dengan `aws rds-data begin-transaction`. Membatalkan transaksi akan membatalkan perubahannya.

> **Important**
>
> Jika ID transaksi telah kedaluwarsa, transaksi dibatalkan secara otomatis. Dalam kasus ini, perintah `aws rds-data rollback-transaction` yang menentukan ID transaksi yang kedaluwarsa akan menghasilkan kesalahan.

Selain opsi umum, tentukan opsi-opsi berikut:

- `--transaction-id` (wajib) – Pengidentifikasi transaksi yang dimulai menggunakan perintah CLI `begin-transaction`. Tentukan ID transaksi yang ingin Anda batalkan.

Misalnya, perintah AWS CLI berikut akan membatalkan transaksi SQL.

Untuk Linux, macOS, atau Unix:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" \
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" \
--transaction-id "ABC1234567890xyz"
```

Untuk Windows:

```
aws rds-data rollback-transaction --resource-arn "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster" ^
--secret-arn "arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret" ^
--transaction-id "ABC1234567890xyz"
```

Berikut adalah contoh respons tersebut.

```
{
    "transactionStatus": "Rollback Complete"
    }
```

Memanggil RDS Data API dari aplikasi Python

Anda dapat memanggil RDS Data API (Data API) dari aplikasi Python.

Contoh berikut menggunakan AWS SDK for Python (Boto). Lihat informasi selengkapnya tentang Boto di Dokumentasi AWS SDK for Python (Boto 3).

Dalam setiap contoh, ganti Amazon Resource Name (ARN) cluster DB dengan ARN untuk cluster Aurora DB Anda. Selain itu, ganti ARN rahasia dengan ARN rahasia di Secrets Manager yang mengizinkan akses ke klaster basis data.

Topik

- Menjalankan kueri SQL
- Menjalankan pernyataan SQL DML
- Menjalankan transaksi SQL

Menjalankan kueri SQL

Anda dapat menjalankan pernyataan `SELECT` dan mengambil hasilnya dengan aplikasi Python.

Contoh berikut menjalankan kueri SQL.

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

response1 = rdsData.execute_statement(
            resourceArn = cluster_arn,
            secretArn = secret_arn,
            database = 'mydb',
            sql = 'select * from employees limit 3')

print (response1['records'])
[
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'ROSALEZ'
        },
        {
            'stringValue': 'ALEJANDRO'
        },
        {
            'stringValue': '2016-02-15 04:34:33.0'
        }
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'DOE'
        },
        {
            'stringValue': 'JANE'
        },
        {
            'stringValue': '2014-05-09 04:34:33.0'
        }
```

```
    ],
    [
        {
            'longValue': 1
        },
        {
            'stringValue': 'STILES'
        },
        {
            'stringValue': 'JOHN'
        },
        {
            'stringValue': '2017-09-20 04:34:33.0'
        }
    ]
]
```

Menjalankan pernyataan SQL DML

Anda dapat menjalankan pernyataan bahasa manipulasi data (DML) untuk memasukkan, memperbarui, atau menghapus data dalam basis data Anda. Anda juga dapat menggunakan parameter dalam pernyataan DML.

> **Important**
>
> Jika panggilan bukan bagian dari transaksi karena tidak menyertakan parameter `transactionID`, perubahan yang dihasilkan dari panggilan tersebut diterapkan secara otomatis.

Contoh berikut menjalankan pernyataan penyisipan SQL dan menggunakan parameter.

```
import boto3

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

rdsData = boto3.client('rds-data')


param1 = {'name':'firstname', 'value':{'stringValue': 'JACKSON'}}
```

```
param2 = {'name':'lastname', 'value':{'stringValue': 'MATEO'}}
paramSet = [param1, param2]

response2 = rdsData.execute_statement(resourceArn=cluster_arn,
                                      secretArn=secret_arn,
                                      database='mydb',
                                      sql='insert into employees(first_name, last_name)
 VALUES(:firstname, :lastname)',
                                      parameters = paramSet)

print (response2["numberOfRecordsUpdated"])
```

Menjalankan transaksi SQL

Anda dapat memulai transaksi SQL, menjalankan satu atau beberapa pernyataan SQL, lalu menerapkan perubahan dengan aplikasi Python.

> **Important**
>
> Waktu transaksi habis jika tidak ada panggilan yang menggunakan ID transaksinya dalam tiga menit. Jika waktu transaksi habis sebelum diterapkan, transaksi akan dibatalkan secara otomatis.
>
> Jika Anda tidak menentukan ID transaksi, perubahan yang dihasilkan dari panggilan akan diterapkan secara otomatis.

Contoh berikut menjalankan transaksi SQL yang menyisipkan baris dalam tabel.

```
import boto3

rdsData = boto3.client('rds-data')

cluster_arn = 'arn:aws:rds:us-east-1:123456789012:cluster:mydbcluster'
secret_arn = 'arn:aws:secretsmanager:us-east-1:123456789012:secret:mysecret'

tr = rdsData.begin_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     database = 'mydb')

response3 = rdsData.execute_statement(
```

```
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     database = 'mydb',
     sql = 'insert into employees(first_name, last_name) values('XIULAN', 'WANG')',
     transactionId = tr['transactionId'])

cr = rdsData.commit_transaction(
     resourceArn = cluster_arn,
     secretArn = secret_arn,
     transactionId = tr['transactionId'])

cr['transactionStatus']
'Transaction Committed'

response3['numberOfRecordsUpdated']
1
```

> **Note**
>
> Jika Anda menjalankan pernyataan bahasa definisi data (DDL), kami merekomendasikan agar Anda terus menjalankan pernyataan setelah waktu panggilan habis. Ketika pernyataan DDL dihentikan sebelum selesai dijalankan, hal ini dapat mengakibatkan kesalahan dan kemungkinan struktur data rusak. Untuk terus menjalankan pernyataan setelah waktu panggilan habis, atur parameter `continueAfterTimeout` ke `true`.

Memanggil RDS Data API dari aplikasi Java

Anda dapat memanggil RDS Data API (Data API) dari aplikasi Java.

Contoh berikut menggunakan AWS SDK for Java. Lihat informasi selengkapnya di Panduan Developer AWS SDK for Java.

Dalam setiap contoh, ganti Amazon Resource Name (ARN) cluster DB dengan ARN untuk cluster Aurora DB Anda. Selain itu, ganti ARN rahasia dengan ARN rahasia di Secrets Manager yang mengizinkan akses ke klaster basis data.

Topik

- Menjalankan kueri SQL
- Menjalankan transaksi SQL

- Menjalankan operasi SQL batch

Menjalankan kueri SQL

Anda dapat menjalankan pernyataan SELECT dan mengambil hasilnya dengan aplikasi Java.

Contoh berikut menjalankan kueri SQL.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementResult;
import com.amazonaws.services.rdsdata.model.Field;

import java.util.List;

public class FetchResultsExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    ExecuteStatementRequest request = new ExecuteStatementRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb")
            .withSql("select * from mytable");

    ExecuteStatementResult result = rdsData.executeStatement(request);

    for (List<Field> fields: result.getRecords()) {
      String stringValue = fields.get(0).getStringValue();
      long numberValue = fields.get(1).getLongValue();

      System.out.println(String.format("Fetched row: string = %s, number = %d",
 stringValue, numberValue));
    }
  }
```

```
}
```

Menjalankan transaksi SQL

Anda dapat memulai transaksi SQL, menjalankan satu atau beberapa pernyataan SQL, lalu menerapkan perubahan dengan aplikasi Java.

> **Important**
>
> Waktu transaksi habis jika tidak ada panggilan yang menggunakan ID transaksinya dalam tiga menit. Jika waktu transaksi habis sebelum diterapkan, transaksi akan dibatalkan secara otomatis.
>
> Jika Anda tidak menentukan ID transaksi, perubahan yang dihasilkan dari panggilan akan diterapkan secara otomatis.

Contoh berikut menjalankan transaksi SQL.

```
package com.amazonaws.rdsdata.examples;

import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BeginTransactionRequest;
import com.amazonaws.services.rdsdata.model.BeginTransactionResult;
import com.amazonaws.services.rdsdata.model.CommitTransactionRequest;
import com.amazonaws.services.rdsdata.model.ExecuteStatementRequest;

public class TransactionExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
    AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BeginTransactionRequest beginTransactionRequest = new BeginTransactionRequest()
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withDatabase("mydb");
    BeginTransactionResult beginTransactionResult =
 rdsData.beginTransaction(beginTransactionRequest);
```

```
    String transactionId = beginTransactionResult.getTransactionId();

    ExecuteStatementRequest executeStatementRequest = new ExecuteStatementRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table VALUES ('hello world!')");
    rdsData.executeStatement(executeStatementRequest);

    CommitTransactionRequest commitTransactionRequest = new CommitTransactionRequest()
            .withTransactionId(transactionId)
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN);
    rdsData.commitTransaction(commitTransactionRequest);
  }
}
```

> **Note**
>
> Jika Anda menjalankan pernyataan bahasa definisi data (DDL), kami merekomendasikan agar Anda terus menjalankan pernyataan setelah waktu panggilan habis. Ketika pernyataan DDL dihentikan sebelum selesai dijalankan, hal ini dapat mengakibatkan kesalahan dan kemungkinan struktur data rusak. Untuk terus menjalankan pernyataan setelah waktu panggilan habis, atur parameter `continueAfterTimeout` ke `true`.

Menjalankan operasi SQL batch

Anda dapat menjalankan operasi penyisipan dan pembaruan massal pada susunan data dengan aplikasi Java. Anda dapat menjalankan pernyataan DML dengan susunan set parameter.

> **Important**
>
> Jika Anda tidak menentukan ID transaksi, perubahan yang dihasilkan dari panggilan akan diterapkan secara otomatis.

Contoh berikut menjalankan operasi penyisipan batch.

```
package com.amazonaws.rdsdata.examples;
```

```
import com.amazonaws.services.rdsdata.AWSRDSData;
import com.amazonaws.services.rdsdata.AWSRDSDataClient;
import com.amazonaws.services.rdsdata.model.BatchExecuteStatementRequest;
import com.amazonaws.services.rdsdata.model.Field;
import com.amazonaws.services.rdsdata.model.SqlParameter;

import java.util.Arrays;

public class BatchExecuteExample {
  public static final String RESOURCE_ARN = "arn:aws:rds:us-
east-1:123456789012:cluster:mydbcluster";
  public static final String SECRET_ARN = "arn:aws:secretsmanager:us-
east-1:123456789012:secret:mysecret";

  public static void main(String[] args) {
      AWSRDSData rdsData = AWSRDSDataClient.builder().build();

    BatchExecuteStatementRequest request = new BatchExecuteStatementRequest()
            .withDatabase("test")
            .withResourceArn(RESOURCE_ARN)
            .withSecretArn(SECRET_ARN)
            .withSql("INSERT INTO test_table2 VALUES (:string, :number)")
            .withParameterSets(Arrays.asList(
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("Hello")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(1L))
                    ),
                    Arrays.asList(
                            new SqlParameter().withName("string").withValue(new
 Field().withStringValue("World")),
                            new SqlParameter().withName("number").withValue(new
 Field().withLongValue(2L))
                    )
            ));

    rdsData.batchExecuteStatement(request);
  }
}
```

Menggunakan pustaka klien Java untuk RDS Data API

Anda dapat mengunduh dan menggunakan pustaka klien Java untuk RDS Data API (Data API). Pustaka klien Java ini menyediakan cara alternatif untuk menggunakan Data API. Dengan menggunakan pustaka ini, Anda dapat memetakan kelas sisi klien Anda ke permintaan dan respons Data API. Dukungan pemetaan ini dapat memudahkan integrasi dengan beberapa jenis Java tertentu, seperti `Date`, `Time`, dan `BigDecimal`.

Mengunduh pustaka klien Java untuk API Data

Pustaka klien Java Data API adalah open source GitHub di lokasi berikut:

https://github.com/awslabs/ rds-data-api-client -perpustakaan-java

Anda dapat membuat pustaka secara manual dari file sumber, tetapi praktik terbaiknya adalah memanfaatkan pustaka tersebut menggunakan manajemen dependensi Apache Maven. Tambahkan dependensi berikut ke file Maven POM Anda:

Untuk versi 2.x, yang kompatibel dengan AWS SDK 2.x, gunakan pilihan berikut:

```
<dependency>
   <groupId>software.amazon.rdsdata</groupId>
   <artifactId>rds-data-api-client-library-java</artifactId>
   <version>2.0.0</version>
</dependency>
```

Untuk versi 1.x, yang kompatibel dengan AWS SDK 1.x, gunakan pilihan berikut:

```
<dependency>
    <groupId>software.amazon.rdsdata</groupId>
    <artifactId>rds-data-api-client-library-java</artifactId>
    <version>1.0.8</version>
</dependency>
```

Contoh pustaka klien Java

Di bawah ini Anda dapat menemukan beberapa contoh umum penggunaan pustaka klien Java API Data. Contoh berikut mengasumsikan bahwa Anda memiliki tabel `accounts` dengan dua kolom: `accountId` dan `name`. Anda juga memiliki data transfer object (DTO) berikut.

```
public class Account {
    int accountId;
    String name;
    // getters and setters omitted
}
```

Pustaka klien memungkinkan Anda meneruskan DTO sebagai parameter input. Contoh berikut menunjukkan bagaimana pelanggan DTO dipetakan ke set parameter input.

```
var account1 = new Account(1, "John");
var account2 = new Account(2, "Mary");
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
         .withParamSets(account1, account2)
         .execute();
```

Dalam beberapa kasus, bekerja dengan nilai sederhana sebagai parameter input akan terasa lebih mudah. Anda dapat melakukannya dengan sintaksis berikut.

```
client.forSql("INSERT INTO accounts(accountId, name) VALUES(:accountId, :name)")
         .withParameter("accountId", 3)
         .withParameter("name", "Zhang")
         .execute();
```

Berikut adalah contoh lain yang bekerja dengan nilai sederhana sebagai parameter input.

```

client.forSql("INSERT INTO accounts(accountId, name) VALUES(?, ?)", 4, "Carlos")
         .execute();

```

Pustaka klien menyediakan pemetaan otomatis ke DTO saat hasilnya dihasilkan. Contoh berikut menunjukkan bagaimana hasil dipetakan ke DTO Anda.

```
List<Account> result = client.forSql("SELECT * FROM accounts")
          .execute()
          .mapToList(Account.class);

Account result = client.forSql("SELECT * FROM accounts WHERE account_id = 1")
          .execute()
          .mapToSingle(Account.class);
```

Dalam banyak kasus, kumpulan hasil basis data hanya berisi satu nilai. Untuk menyederhanakan pengambilan hasil tersebut, pustaka klien menawarkan API berikut:

```
int numberOfAccounts = client.forSql("SELECT COUNT(*) FROM accounts")
          .execute()
          .singleValue(Integer.class);
```

> **Note**
>
> Fungsi `mapToList` mengonversi set hasil SQL ke dalam daftar objek yang ditentukan pengguna. Kami tidak mendukung penggunaan pernyataan `.withFormatRecordsAs(RecordsFormatType.JSON)` dalam panggilan `ExecuteStatement` untuk pustaka klien Java karena fungsinya sama. Untuk informasi selengkapnya, lihat Memproses hasil kueri dalam format JSON.

Memproses hasil kueri dalam format JSON

Ketika Anda memanggil operasi `ExecuteStatement`, Anda dapat memilih untuk menampilkan hasil kueri sebagai string dalam format JSON. Dengan demikian, Anda dapat menggunakan kemampuan parsing JSON bahasa pemrograman Anda untuk menafsirkan dan memformat ulang set hasil. Dengan melakukan ini, Anda dapat membantu menghindari penulisan kode tambahan untuk mengulang set hasil dan menafsirkan setiap nilai kolom.

Untuk meminta set hasil dalam format JSON, teruskan parameter `formatRecordsAs` opsional dengan nilai `JSON`. Set hasil yang diformat JSON ditampilkan di bidang `formattedRecords` struktur `ExecuteStatementResponse`.

Tindakan `BatchExecuteStatement` tidak menampilkan set hasil. Dengan demikian, opsi JSON tidak berlaku untuk tindakan itu.

Untuk menyesuaikan kunci dalam struktur hash JSON, tentukan alias kolom dalam set hasil. Anda dapat melakukannya dengan menggunakan klausa `AS` dalam daftar kolom kueri SQL Anda.

Anda dapat menggunakan kemampuan JSON untuk membuat set hasil lebih mudah dibaca dan memetakan isinya ke kerangka kerja khusus bahasa. Karena volume set hasil berenkode ASCII lebih besar dari representasi default, Anda dapat memilih representasi default untuk kueri yang menampilkan sejumlah besar baris atau nilai kolom besar yang menghabiskan lebih banyak memori daripada yang tersedia untuk aplikasi Anda.

Topik

- Mengambil hasil kueri dalam format JSON
- Pemetaan Tipe Data
- Memecahkan masalah
- Contoh-contoh

Mengambil hasil kueri dalam format JSON

Untuk menerima hasil yang ditetapkan sebagai string JSON, sertakan `.withFormatRecordsAs(RecordsFormatType.JSON)` dalam `ExecuteStatement` panggilan. Nilai pengembalian kembali sebagai string JSON di bidang `formattedRecords`. Dalam kasus ini, `columnMetadata` bernilai `null`. Label kolom adalah kunci dari objek yang mewakili setiap baris. Nama kolom ini diulang untuk setiap baris dalam set hasil. Nilai kolom adalah string berkutip, nilai numerik, atau nilai khusus yang merepresentasikan `true`, `false`, atau `null`. Metadata kolom seperti batasan panjang dan tipe yang tepat untuk angka dan string tidak dipertahankan dalam respons JSON.

Jika Anda menghilangkan panggilan `.withFormatRecordsAs()` atau menentukan parameter `NONE`, set hasil ditampilkan dalam format biner menggunakan bidang `Records` dan `columnMetadata`.

Pemetaan Tipe Data

Nilai SQL dalam set hasil dipetakan ke set tipe JSON yang lebih kecil. Nilai direpresentasikan dalam JSON sebagai string, angka, dan beberapa konstanta khusus seperti `true`, `false`, dan `null`. Anda dapat mengonversi nilai-nilai ini menjadi berbagai variabel dalam aplikasi Anda, dengan menggunakan pengetikan kuat atau lemah yang sesuai untuk bahasa pemrograman Anda.

Tipe data JDBC	Tipe data JSON
`INTEGER`, `TINYINT`, `SMALLINT`, `BIGINT`	Angka secara default. String jika opsi `LongReturnType` diatur ke `STRING`.
`FLOAT`, `REAL`, `DOUBLE`	Angka

Tipe data JDBC	Tipe data JSON
DECIMAL	String secara default. Angka jika opsi DecimalReturnType diatur ke DOUBLE_OR_LONG .
STRING	String
BOOLEAN, BIT	Boolean
BLOB, BINARY, VARBINARY , LONGVARBINARY	String dalam enkode base64.
CLOB	String
ARRAY	Susunan
NULL	null
Tipe lainnya (termasuk tipe terkait tanggal dan waktu)	String

Memecahkan masalah

Respons JSON dibatasi hingga 10 megabyte. Jika responsnya lebih besar dari batas ini, program Anda menerima kesalahan BadRequestException. Dalam kasus ini, Anda dapat mengatasi kesalahan menggunakan salah satu teknik berikut:

- Kurangi jumlah baris di set hasil. Untuk melakukannya, tambahkan klausa LIMIT. Anda dapat membagi set hasil besar menjadi beberapa set yang lebih kecil dengan mengirimkan beberapa kueri dengan klausa LIMIT dan OFFSET.

  Jika set hasil menyertakan baris yang difilter oleh logika aplikasi, Anda dapat menghapus baris tersebut dari set hasil dengan menambahkan kondisi lainnya dalam klausa WHERE.
- Kurangi jumlah kolom di set hasil. Untuk melakukannya, hapus item dari daftar pilih kueri.
- Persingkat label kolom dengan menggunakan alias kolom dalam kueri. Setiap nama kolom diulang dalam string JSON untuk setiap baris dalam set hasil. Oleh karena itu, hasil kueri dengan nama

kolom panjang dan banyak baris dapat melebihi batas ukuran. Khususnya, gunakan alias kolom untuk ekspresi rumit agar seluruh ekspresi tidak diulang dalam string JSON.

- Meskipun Anda dapat menggunakan alias kolom dengan SQL untuk menghasilkan set hasil yang memiliki beberapa kolom dengan nama yang sama, nama kunci duplikat tidak diperbolehkan di JSON. RDS API Data menampilkan kesalahan jika Anda meminta set hasil dalam format JSON dan beberapa kolom memiliki nama yang sama. Jadi, pastikan semua label kolom memiliki nama yang unik.

Contoh-contoh

Contoh Java berikut menunjukkan cara memanggil `ExecuteStatement` dengan respons sebagai string berformat JSON, lalu menafsirkan set hasil. *`Gantikan nilai yang sesuai untuk parameter DatabaseName, secretStoreArn, dan ClusterArn.`*

Contoh Java berikut menunjukkan kueri yang menampilkan nilai numerik desimal di set hasil. Panggilan `assertThat` menguji apakah bidang respons memiliki properti yang diharapkan berdasarkan aturan untuk set hasil JSON.

Contoh ini bekerja dengan skema dan data sampel berikut:

```
create table test_simplified_json (a float);
insert into test_simplified_json values(10.0);
```

```
public void JSON_result_set_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request);
}
```

Nilai bidang `formattedRecords` dari program sebelumnya adalah:

```
[{"a":10.0}]
```

Bidang `Records` dan `ColumnMetadata` dalam respons keduanya bernilai null karena adanya set hasil JSON.

Contoh Java berikut menunjukkan kueri yang menampilkan nilai numerik integer di set hasil. Contoh panggilan `getFormattedRecords` untuk menampilkan hanya string berformat JSON dan mengabaikan bidang respons lain yang kosong atau bernilai null. Contoh ini melakukan deserialisasi hasil ke dalam struktur yang merepresentasikan daftar catatan. Setiap catatan memiliki bidang yang namanya sesuai dengan alias kolom dari set kumpulan. Teknik ini menyederhanakan kode yang mem-parsing set hasil. Aplikasi Anda tidak harus mengulang baris dan kolom dari set hasil dan mengonversi setiap nilai ke tipe yang sesuai.

Contoh ini bekerja dengan skema dan data sampel berikut:

```
create table test_simplified_json (a int);
insert into test_simplified_json values(17);
```

```
public void JSON_deserialization_demo() {
    var sql = "select * from test_simplified_json";
    var request = new ExecuteStatementRequest()
      .withDatabase(databaseName)
      .withSecretArn(secretStoreArn)
      .withResourceArn(clusterArn)
      .withSql(sql)
      .withFormatRecordsAs(RecordsFormatType.JSON);
    var result = rdsdataClient.executeStatement(request)
      .getFormattedRecords();

/* Turn the result set into a Java object, a list of records.
   Each record has a field 'a' corresponding to the column
   labelled 'a' in the result set. */
    private static class Record { public int a; }
    var recordsList = new ObjectMapper().readValue(
        response, new TypeReference<List<Record>>() {
        });
}
```

Nilai bidang `formattedRecords` dari program sebelumnya adalah:

```
[{"a":17}]
```

Untuk mengambil kolom a baris hasil 0, aplikasi akan merujuk ke `recordsList.get(0).a`.

Sebaliknya, contoh Java berikut menunjukkan tipe kode yang diperlukan untuk membangun struktur data yang mempertahankan set hasil ketika Anda tidak menggunakan format JSON. Dalam kasus ini, setiap baris dari set hasil berisi bidang dengan informasi tentang satu pengguna. Membangun struktur data untuk merepresentasikan set hasil membutuhkan perulangan melalui baris. Untuk setiap baris, kode mengambil nilai setiap bidang, melakukan konversi tipe yang sesuai, dan menetapkan hasilnya ke bidang yang sesuai dalam objek yang mewakili baris. Kemudian, kode menambahkan objek yang merepresentasikan setiap pengguna ke struktur data yang mewakili seluruh set hasil. Jika kueri diubah untuk menyusun ulang, menambah, atau menghapus bidang dalam set hasil, kode aplikasi juga harus diubah.

```
/* Verbose result-parsing code that doesn't use the JSON result set format */
for (var row: response.getRecords()) {
    var user = User.builder()
      .userId(row.get(0).getLongValue())
      .firstName(row.get(1).getStringValue())
      .lastName(row.get(2).getStringValue())
      .dob(Instant.parse(row.get(3).getStringValue()))
      .build();
    result.add(user);
  }
```

Nilai sampel berikut menunjukkan nilai bidang `formattedRecords` untuk set hasil dengan jumlah kolom, alias kolom, dan tipe data kolom yang berbeda.

Jika set hasil mencakup beberapa baris, setiap barisnya direpresentasikan sebagai objek yang merupakan elemen susunan. Setiap kolom dalam set hasil menjadi kunci dalam objek. Nama kolom ini diulang untuk setiap baris dalam set hasil. Oleh karena itu, untuk set hasil yang terdiri dari banyak baris dan kolom, Anda mungkin perlu menentukan alias kolom pendek agar tidak melebihi batas panjang untuk seluruh respons.

Contoh ini bekerja dengan skema dan data sampel berikut:

```
create table sample_names (id int, name varchar(128));
insert into sample_names values (0, "Jane"), (1, "Mohan"), (2, "Maria"), (3, "Bruce"),
 (4, "Jasmine");
```

```
[{"id":0,"name":"Jane"},{"id":1,"name":"Mohan"},
{"id":2,"name":"Maria"},{"id":3,"name":"Bruce"},{"id":4,"name":"Jasmine"}]
```

Jika kolom dalam set hasil didefinisikan sebagai ekspresi, teks ekspresi menjadi kunci JSON. Oleh karena itu, praktik mudah umumnya adalah menentukan alias kolom deskriptif untuk setiap ekspresi dalam daftar pilih kueri. Misalnya, kueri berikut mencakup ekspresi seperti panggilan fungsi dan operasi aritmatika dalam daftar pilihannya.

```
select count(*), max(id), 4+7 from sample_names;
```

Ekspresi tersebut diteruskan ke set hasil JSON sebagai kunci.

```
[{"count(*)":5,"max(id)":4,"4+7":11}]
```

Menambahkan kolom AS dengan label deskriptif membuat kunci lebih sederhana untuk ditafsirkan dalam set hasil JSON.

```
select count(*) as rows, max(id) as largest_id, 4+7 as addition_result from
 sample_names;
```

Dengan kueri SQL yang direvisi, label kolom yang ditentukan oleh klausa AS digunakan sebagai nama kunci.

```
[{"rows":5,"largest_id":4,"addition_result":11}]
```

Nilai untuk setiap pasangan kunci-nilai dalam string JSON dapat berupa string berkutip. String dapat berisi karakter unicode. Jika string berisi urutan escape maupun karakter '' atau \, karakter tersebut didahului oleh karakter escape backslash. Contoh string JSON berikut menunjukkan kemungkinan ini. Misalnya, hasil `string_with_escape_sequences` berisi karakter khusus backspace, newline, carriage return, tab, form feed, dan \.

```
[{"quoted_string":"hello"}]
[{"unicode_string":"####"}]
[{"string_with_escape_sequences":"\b \n \r \t \f \\ '"}]
```

Nilai untuk setiap pasangan kunci-nilai dalam string JSON juga dapat merepresentasikan angka. Angka tersebut dapat berupa bilangan bulat, nilai floating-point, nilai negatif, atau nilai yang direpresentasikan sebagai notasi eksponensial. Contoh string JSON berikut menunjukkan kemungkinan ini.

```
[{"integer_value":17}]
[{"float_value":10.0}]
```

```
[{"negative_value":-9223372036854775808,"positive_value":9223372036854775807}]
[{"very_small_floating_point_value":4.9E-324,"very_large_floating_point_value":1.79769313486231
```

Nilai Boolean dan null direpresentasikan dengan kata kunci khusus tanpa kutip `true`, `false`, dan `null`. Contoh string JSON berikut menunjukkan kemungkinan ini.

```
[{"boolean_value_1":true,"boolean_value_2":false}]
[{"unknown_value":null}]
```

Jika Anda memilih nilai tipe BLOB, hasilnya direpresentasikan dalam string JSON sebagai nilai berenkode base64. Untuk mengonversi kembali nilai ke representasi aslinya, Anda dapat menggunakan fungsi dekode yang sesuai dalam bahasa aplikasi Anda. Misalnya, Anda memanggil fungsi `Base64.getDecoder().decode()` di Java. Output sampel berikut menunjukkan hasil pemilihan nilai BLOB `hello world` dan menampilkan set hasil sebagai string JSON.

```
[{"blob_column":"aGVsbG8gd29ybGQ="}]
```

Contoh Python berikut menunjukkan cara mengakses nilai-nilai dari hasil panggilan ke fungsi `execute_statement` Python. Set hasil adalah nilai string di bidang `response['formattedRecords']`. Kode ini mengubah string JSON menjadi struktur data dengan memanggil fungsi `json.loads`. Kemudian, setiap baris dari set hasil adalah elemen daftar dalam struktur data, dan Anda dapat merujuk ke setiap bidang set hasil berdasarkan nama dalam setiap baris.

```
import json

result = json.loads(response['formattedRecords'])
print (result[0]["id"])
```

JavaScript Contoh berikut menunjukkan cara mengakses nilai-nilai dari hasil panggilan ke JavaScript `executeStatement` fungsi. Set hasil adalah nilai string di bidang `response.formattedRecords`. Kode ini mengubah string JSON menjadi struktur data dengan memanggil fungsi `JSON.parse`. Kemudian, setiap baris dari set hasil adalah elemen susunan dalam struktur data, dan Anda dapat merujuk ke setiap bidang set hasil berdasarkan nama dalam setiap baris.

```
<script>
    const result = JSON.parse(response.formattedRecords);
    document.getElementById("display").innerHTML = result[0].id;
</script>
```

Memecahkan masalah RDS Data API

Gunakan bagian berikut, berjudul dengan pesan kesalahan umum, untuk membantu memecahkan masalah yang Anda miliki dengan RDS Data API (Data API).

Topik

- Transaksi <transaction_ID> tidak ditemukan
- Paket untuk kueri terlalu besar
- Respons basis data melebihi batas ukuran
- HttpEndpoint tidak diaktifkan untuk klaster <cluster_ID>

Transaksi <transaction_ID> tidak ditemukan

Dalam kasus ini, ID transaksi yang ditentukan dalam panggilan API Data tidak ditemukan. Penyebab masalah ini ditambahkan ke pesan kesalahan, dan merupakan salah satu dari yang berikut:

- Transaksi mungkin kedaluwarsa.

  Pastikan setiap panggilan transaksional berjalan dalam tiga menit sejak panggilan sebelumnya berjalan.

  Mungkin juga ID transaksi yang ditentukan tidak dibuat oleh BeginTransactionpanggilan. Pastikan panggilan Anda memiliki ID transaksi yang valid.
- Satu panggilan sebelumnya mengakibatkan penghentian transaksi Anda.

  Transaksi telah diakhiri oleh panggilan `CommitTransaction` atau `RollbackTransaction` Anda.
- Transaksi telah dibatalkan karena kesalahan dari panggilan sebelumnya.

  Periksa apakah panggilan Anda sebelumnya telah memberikan pengecualian apa pun.

Lihat informasi tentang menjalankan transaksi di Memanggil RDS Data API.

Paket untuk kueri terlalu besar

Dalam kasus ini, set hasil yang ditampilkan untuk satu baris terlalu besar. Batas ukuran API Data adalah 64 KB per baris dalam set hasil yang ditampilkan oleh basis data.

Untuk mengatasi masalah ini, pastikan setiap baris dalam set hasil berukuran 64 KB atau kurang.

Respons basis data melebihi batas ukuran

Dalam kasus ini, ukuran set hasil yang ditampilkan oleh basis data terlalu besar. Batas API Data adalah 1 MB per baris dalam set hasil yang ditampilkan oleh basis data.

Untuk mengatasi masalah ini, pastikan panggilan ke Data API mengembalikan 1 MiB data atau kurang. Jika Anda perlu menampilkan lebih dari 1 MiB, Anda dapat menggunakan beberapa panggilan `ExecuteStatement` dengan klausa `LIMIT` di kueri Anda.

Lihat informasi selengkapnya tentang klausa `LIMIT` di Sintaksis SELECT dalam dokumentasi MySQL.

HttpEndpoint tidak diaktifkan untuk klaster <cluster_ID>

Periksa penyebab potensial berikut untuk masalah ini:

- Cluster Aurora DB tidak mendukung API Data. Misalnya, untuk Aurora MySQL, Anda hanya dapat menggunakan Data API dengan. Aurora Serverless v1 Untuk informasi tentang jenis cluster DB yang didukung RDS Data API, lihat. the section called “Ketersediaan Wilayah dan versi”
- Data API tidak diaktifkan untuk cluster Aurora DB. Untuk menggunakan Data API dengan cluster Aurora DB, Data API harus diaktifkan untuk cluster DB. Untuk informasi tentang mengaktifkan Data API, lihatMengaktifkan API Data RDS.
- Cluster DB diganti namanya setelah Data API diaktifkan untuknya. Dalam hal ini, matikan Data API untuk cluster itu dan kemudian aktifkan lagi.
- ARN yang Anda tentukan tidak sama persis dengan ARN klaster. Periksa apakah ARN yang ditampilkan dari sumber lain atau dibangun oleh logika program cocok dengan ARN klaster secara tepat. Misalnya, pastikan ARN yang Anda gunakan memiliki kapitalisasi huruf yang benar untuk semua karakter alfabet.

Mencatat panggilan API Data RDS dengan AWS CloudTrail

RDS Data API (Data API) terintegrasi dengan AWS CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau AWS layanan di Data API. CloudTrail menangkap semua panggilan API untuk Data API sebagai peristiwa, termasuk panggilan dari konsol Amazon RDS dan dari panggilan kode ke operasi API Data. Jika Anda membuat jejak, Anda dapat

mengaktifkan pengiriman CloudTrail peristiwa secara berkelanjutan ke bucket Amazon S3, termasuk peristiwa untuk Data API. Dengan menggunakan data yang dikumpulkan oleh CloudTrail, Anda dapat menentukan banyak informasi. Informasi ini mencakup permintaan yang dibuat untuk API Data, alamat IP asal permintaan, siapa yang membuat permintaan, kapan permintaan tersebut dibuat, dan detail tambahan.

Untuk mempelajari selengkapnya CloudTrail, lihat Panduan AWS CloudTrail Pengguna.

Bekerja dengan informasi API Data di CloudTrail

CloudTrail diaktifkan di AWS akun Anda saat Anda membuat akun. Ketika aktivitas yang didukung (peristiwa manajemen) terjadi di Data API, aktivitas tersebut direkam dalam suatu CloudTrail peristiwa bersama dengan peristiwa AWS layanan lainnya dalam riwayat Acara. Anda dapat melihat, mencari, dan mengunduh acara manajemen terbaru di AWS akun Anda. Untuk informasi selengkapnya, lihat Bekerja dengan riwayat CloudTrail Acara di Panduan AWS CloudTrail Pengguna.

Untuk catatan peristiwa yang sedang berlangsung di AWS akun Anda, termasuk peristiwa untuk Data API, buat jejak. Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket Amazon S3. Secara default, saat Anda membuat jejak di konsol, jejak tersebut berlaku untuk semua AWS Wilayah. Jejak mencatat peristiwa dari semua AWS Wilayah di AWS partisi dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Selain itu, Anda dapat mengonfigurasi AWS layanan lain untuk menganalisis lebih lanjut dan menindaklanjuti data peristiwa yang dikumpulkan dalam CloudTrail log. Untuk informasi selengkapnya, lihat topik berikut di Panduan Pengguna AWS CloudTrail :

- Ikhtisar untuk membuat jejak
- CloudTrail layanan dan integrasi yang didukung
- Mengonfigurasi notifikasi Amazon SNS untuk CloudTrail
- Menerima file CloudTrail log dari beberapa Wilayah dan Menerima file CloudTrail log dari beberapa akun

Semua operasi API Data dicatat oleh CloudTrail dan didokumentasikan dalam referensi API layanan data Amazon RDS. Misalnya, panggilan ke`BatchExecuteStatement`, `BeginTransaction``CommitTransaction`, dan `ExecuteStatement` operasi menghasilkan entri dalam file CloudTrail log.

Setiap peristiwa atau entri log berisi informasi tentang siapa yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan hal berikut:

- Baik permintaan tersebut dibuat dengan kredensial pengguna root atau pengguna.
- Apakah permintaan dibuat dengan kredensial keamanan sementara untuk suatu peran atau pengguna gabungan.
- Apakah permintaan itu dibuat oleh AWS layanan lain.

Untuk informasi selengkapnya, lihat Elemen userIdentity CloudTrail .

Menyertakan dan mengecualikan peristiwa API Data dari jejak AWS CloudTrail

Sebagian besar pengguna API Data mengandalkan peristiwa dalam AWS CloudTrail jejak untuk menyediakan catatan operasi API Data. Data peristiwa tidak mengungkapkan nama database, nama skema, atau pernyataan SQL dalam permintaan ke Data API. Namun, mengetahui pengguna mana yang membuat jenis panggilan terhadap cluster DB tertentu pada waktu tertentu dapat membantu mendeteksi pola akses anomali.

Menyertakan peristiwa API Data dalam suatu AWS CloudTrail jejak

Untuk Aurora PostgreSQL Serverless v2 dan database yang disediakan, operasi API Data berikut dicatat sebagai peristiwa data. AWS CloudTrail Peristiwa data adalah operasi API data-plane volume tinggi yang CloudTrail tidak masuk secara default. Biaya tambahan berlaku untuk peristiwa data. Untuk informasi tentang CloudTrail harga, lihat AWS CloudTrail Harga.

- BatchExecuteStatement
- BeginTransaction
- CommitTransaction
- ExecuteStatement
- RollbackTransaction

Anda dapat menggunakan CloudTrail konsolAWS CLI, atau operasi CloudTrail API untuk mencatat operasi API Data ini. Di CloudTrail konsol, pilih RDS Data API - DB Cluster untuk tipe peristiwa Data. Untuk informasi selengkapnya, lihat Mencatat peristiwa data dengan AWS Management Console di Panduan AWS CloudTrail Pengguna.

Dengan menggunakan AWS CLI, jalankan `aws cloudtrail put-event-selectors` perintah untuk mencatat operasi Data API ini untuk jejak Anda. Untuk mencatat semua peristiwa Data API

pada kluster DB, `AWS::RDS::DBCluster` tentukan jenis sumber daya. Contoh berikut mencatat semua peristiwa Data API pada cluster DB. Untuk informasi selengkapnya, lihat Mencatat peristiwa data dengan AWS Command Line Interface di Panduan AWS CloudTrail Pengguna.

```
aws cloudtrail put-event-selectors --trail-name trail_name --advanced-event-selectors \
'{
   "Name": "RDS Data API Selector",
   "FieldSelectors": [
      {
         "Field": "eventCategory",
         "Equals": [
            "Data"
         ]
      },
      {
         "Field": "resources.type",
         "Equals": [
            "AWS::RDS::DBCluster"
         ]
      }
   ]
}'
```

Anda dapat mengonfigurasi pemilih acara lanjutan untuk memfilter tambahan pada`readOnly`, `eventName`, dan `resources.`ARN bidang. Untuk informasi lebih lanjut tentang bidang ini, lihat AdvancedFieldSelector.

Mengecualikan peristiwa Data API dari AWS CloudTrail jejak (Aurora Serverless v1hanya)

UntukAurora Serverless v1, peristiwa Data API adalah peristiwa manajemen. Secara default, semua peristiwa Data API disertakan dalam AWS CloudTrail jejak. Namun, karena Data API dapat menghasilkan sejumlah besar peristiwa, Anda mungkin ingin mengecualikan peristiwa ini dari CloudTrail jejak Anda. Setelan peristiwa Exclude Amazon RDS Data API mengecualikan semua peristiwa Data API dari jejak. Anda tidak dapat mengecualikan peristiwa API Data tertentu.

Untuk mengecualikan peristiwa API Data dari jejak, lakukan hal berikut:

- Di CloudTrail konsol, pilih setelan Kecualikan peristiwa Amazon RDS Data API saat Anda membuat jejak atau memperbarui jejak.

- Di CloudTrail API, gunakan PutEventSelectorsoperasi. Jika Anda menggunakan pemilih peristiwa lanjutan, Anda dapat mengecualikan peristiwa API Data dengan menyetel `eventSource` bidang yang tidak sama dengan`rdsdata.amazonaws.com`. Jika Anda menggunakan pemilih acara dasar, Anda dapat mengecualikan peristiwa Data API dengan menyetel nilai `ExcludeManagementEventSources` atribut `kerdsdata.amazonaws.com`. Untuk informasi selengkapnya, lihat Mencatat peristiwa dengan AWS Command Line Interface di Panduan AWS CloudTrail Pengguna.

> **Warning**
>
> Mengecualikan peristiwa Data API dari CloudTrail log dapat mengaburkan tindakan API Data. Berhati-hatilah saat memberikan `cloudtrail:PutEventSelectors` kepada pengguna utama yang diperlukan untuk melakukan operasi ini.

Anda dapat mematikan pengecualian ini kapan saja dengan mengubah pengaturan konsol atau pemilih peristiwa untuk jejak. Jejak kemudian akan mulai mencatat peristiwa API Data. Namun, jejak tidak dapat memulihkan peristiwa API Data yang terjadi sedangkan pengecualian tersebut sedang berlaku.

Saat Anda mengecualikan peristiwa Data API dengan menggunakan konsol atau API, operasi CloudTrail `PutEventSelectors` API yang dihasilkan juga dicatat di CloudTrail log Anda. Jika peristiwa Data API tidak muncul di CloudTrail log Anda, cari `PutEventSelectors` peristiwa dengan `ExcludeManagementEventSources` atribut yang disetel `kerdsdata.amazonaws.com`.

Untuk informasi lebih lanjut, lihat Mencatat peristiwa manajemen untuk jejak di Panduan Pengguna AWS CloudTrail .

Memahami entri file log API Data

Trail adalah konfigurasi yang memungkinkan pengiriman peristiwa sebagai file log ke bucket Amazon S3 yang Anda tentukan. CloudTrail file log berisi satu atau lebih entri log. Peristiwa mewakili permintaan tunggal dari sumber manapun dan mencakup informasi tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya. CloudTrail file log bukanlah jejak tumpukan yang diurutkan dari panggilan API publik, jadi file tersebut tidak muncul dalam urutan tertentu.

Aurora PostgreSQL Serverless v2 dan disediakan

Contoh berikut menunjukkan entri CloudTrail log yang menunjukkan `ExecuteStatement` operasi untuk Aurora PostgreSQL Serverless v2 dan database yang disediakan. Untuk database ini, semua peristiwa Data API adalah peristiwa data di mana sumber peristiwa adalah rdsdataapi.amazonaws.com dan jenis acara adalah Rds Data Service.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdataapi.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
        "includeResultMetadata": false,
        "parameters": [],
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
        "sql": "**********"
    },
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "Rds Data Service",
    "recipientAccountId": "123456789012"
}
```


Aurora Serverless v1

Contoh berikut menunjukkan bagaimana entri CloudTrail log contoh sebelumnya muncul untuk. Aurora Serverless v1 UntukAurora Serverless v1, semua acara adalah acara manajemen di mana sumber acara adalah rdsdata.amazonaws.com dan jenis acaranya adalah. AwsApiCall


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/johndoe",
        "accountId": "123456789012",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "johndoe"
    },
    "eventTime": "2019-12-18T00:49:34Z",
    "eventSource": "rdsdata.amazonaws.com",
    "eventName": "ExecuteStatement",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "aws-cli/1.16.102 Python/3.7.2 Windows/10 botocore/1.12.92",
    "requestParameters": {
        "continueAfterTimeout": false,
        "database": "**********",
        "includeResultMetadata": false,
        "parameters": [],
        "resourceArn": "arn:aws:rds:us-east-1:123456789012:cluster:my-database-1",
        "schema": "**********",
        "secretArn": "arn:aws:secretsmanager:us-
east-1:123456789012:secret:dataapisecret-ABC123",
        "sql": "**********"
    },
    "responseElements": null,
    "requestID": "6ba9a36e-b3aa-4ca8-9a2e-15a9eada988e",
    "eventID": "a2c7a357-ee8e-4755-a0d0-aed11ed4253a",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

Menggunakan editor kueri

Dengan editor kueri, Anda dapat menjalankan kueri SQL di konsol RDS. Anda dapat menjalankan manipulasi data dan pernyataan SQL definisi data pada cluster DB. SQL yang dapat Anda jalankan tunduk pada batasan Data API. Untuk informasi selengkapnya, lihat the section called “Batasan”.

Editor kueri memerlukan cluster Aurora DB dengan RDS Data API (Data API) diaktifkan. Untuk informasi tentang cluster DB yang mendukung Data API dan cara mengaktifkannya, lihatMenggunakan RDS Data API.

Ketersediaan editor kueri

Editor kueri tersedia untuk cluster Aurora DB menggunakan versi mesin Aurora MySQL dan Aurora PostgreSQL yang mendukung API Data, dan di mana Data API tersedia. Wilayah AWS Untuk informasi selengkapnya, lihat API Data RDS.

Mengotorisasi akses ke editor kueri

Pengguna harus diotorisasi untuk menjalankan kueri di editor kueri. Anda dapat mengotorisasi pengguna untuk menjalankan kueri di editor kueri dengan menambahkan `AmazonRDSDataFullAccess` kebijakan, kebijakan yang telah ditentukan AWS Identity and Access Management (IAM), ke pengguna tersebut.

> **Note**
>
> Pastikan untuk menggunakan nama pengguna dan kata sandi yang sama saat membuat pengguna seperti yang Anda lakukan untuk pengguna basis data, seperti nama pengguna dan kata sandi utama. Untuk informasi selengkapnya, lihat Membuat Pengguna IAM di Akun AWS Anda dalam Panduan Pengguna AWS Identity and Access Management .

Anda juga dapat membuat kebijakan IAM yang memberikan akses ke editor kueri. Setelah Anda membuat kebijakan, tambahkan ke setiap pengguna yang membutuhkan akses ke editor kueri.

Kebijakan berikut memberikan izin minimum yang diperlukan bagi pengguna untuk mengakses editor kueri.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "QueryEditor0",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutResourcePolicy",
                "secretsmanager:PutSecretValue",
                "secretsmanager:DeleteSecret",
                "secretsmanager:DescribeSecret",
                "secretsmanager:TagResource"
            ],
            "Resource": "arn:aws:secretsmanager:*:*:secret:rds-db-credentials/*"
        },
        {
            "Sid": "QueryEditor1",
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword",
                "tag:GetResources",
                "secretsmanager:CreateSecret",
                "secretsmanager:ListSecrets",
                "dbqms:CreateFavoriteQuery",
                "dbqms:DescribeFavoriteQueries",
                "dbqms:UpdateFavoriteQuery",
                "dbqms:DeleteFavoriteQueries",
                "dbqms:GetQueryString",
                "dbqms:CreateQueryHistory",
                "dbqms:UpdateQueryHistory",
                "dbqms:DeleteQueryHistory",
                "dbqms:DescribeQueryHistory",
                "rds-data:BatchExecuteStatement",
                "rds-data:BeginTransaction",
                "rds-data:CommitTransaction",
                "rds-data:ExecuteStatement",
                "rds-data:RollbackTransaction"
            ],
            "Resource": "*"
        }
    ]
}
```

Untuk informasi tentang pembuatan kebijakan IAM, lihat Membuat kebijakan IAM di Panduan Pengguna AWS Identity and Access Management .

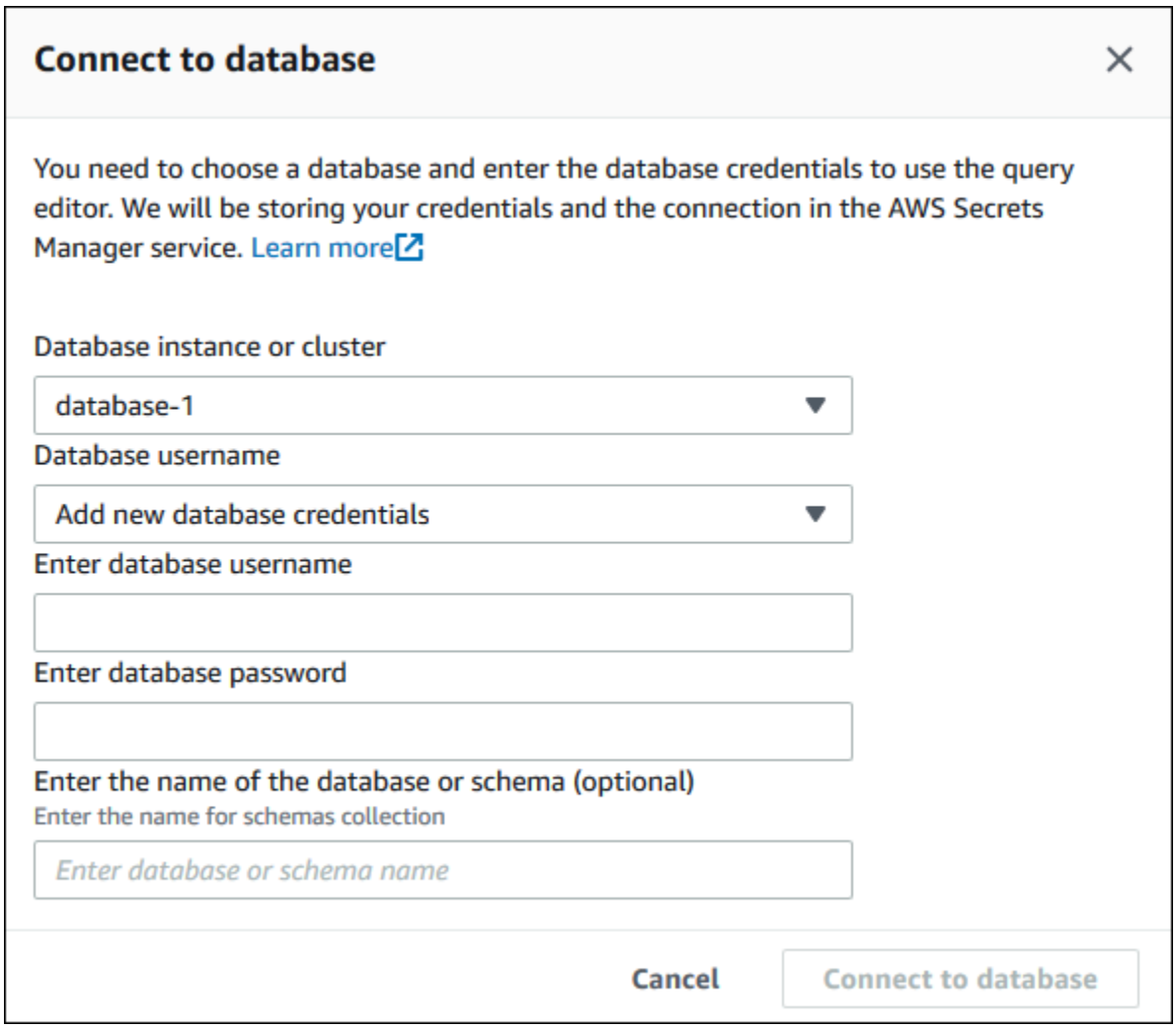
Untuk informasi tentang cara menambahkan kebijakan IAM ke pengguna, lihat Menambahkan dan menghapus izin identitas IAM di Panduan Pengguna AWS Identity and Access Management .

Menjalankan kueri di editor kueri

Anda dapat menjalankan pernyataan SQL pada cluster Aurora DB di editor kueri. SQL yang dapat Anda jalankan tunduk pada batasan Data API. Untuk informasi selengkapnya, lihat the section called “Batasan”.

Untuk menjalankan kueri di editor kueri

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di sudut kanan atas AWS Management Console, pilih Wilayah AWS di mana Anda membuat cluster Aurora DB yang ingin Anda kueri.
3. Di panel navigasi, pilih Basis Data.
4. Pilih cluster Aurora DB tempat Anda ingin menjalankan kueri SQL.
5. Untuk Tindakan, pilih Kueri. Jika Anda belum tersambung ke basis data sebelumnya, halaman Hubungkan ke basis data akan terbuka.



6. Masukkan informasi berikut:

   a. Untuk instance atau cluster Database, pilih cluster Aurora DB tempat Anda ingin menjalankan kueri SQL.

   b. Untuk Nama pengguna basis data, pilih nama pengguna dari pengguna basis data yang akan dihubungkan, atau pilih Tambahkan kredensial basis data baru. Jika Anda memilih Tambahkan kredensial basis data baru, masukkan nama pengguna untuk kredensial basis data baru di Masukkan nama pengguna basis data.

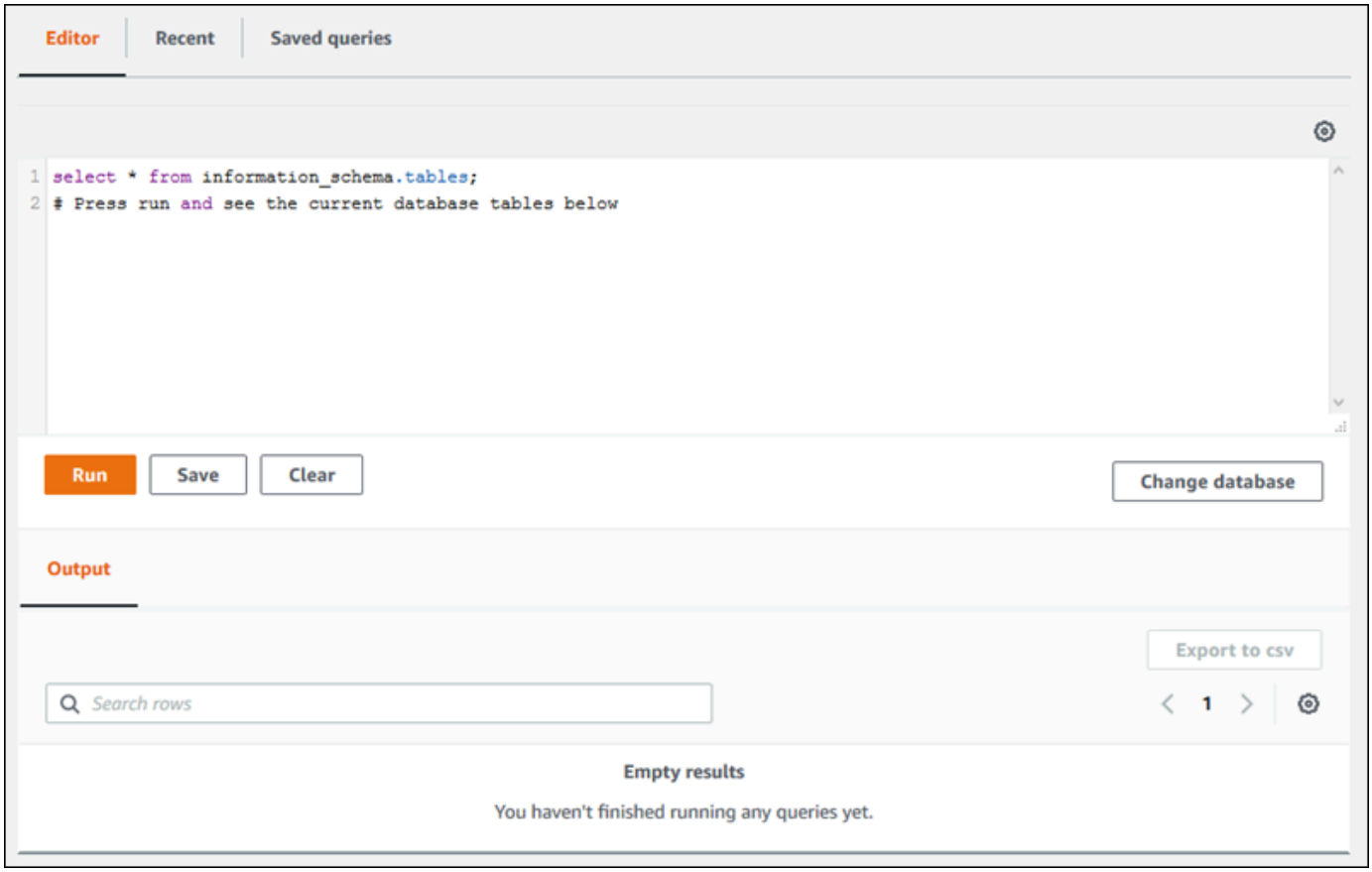
   c. Untuk Masukkan kata sandi basis data, masukkan kata sandi untuk pengguna basis data yang Anda pilih.

   d. Di kotak terakhir, masukkan nama basis data atau skema yang ingin digunakan untuk klaster DB Aurora.

   e. Pilih Hubungkan ke basis data.

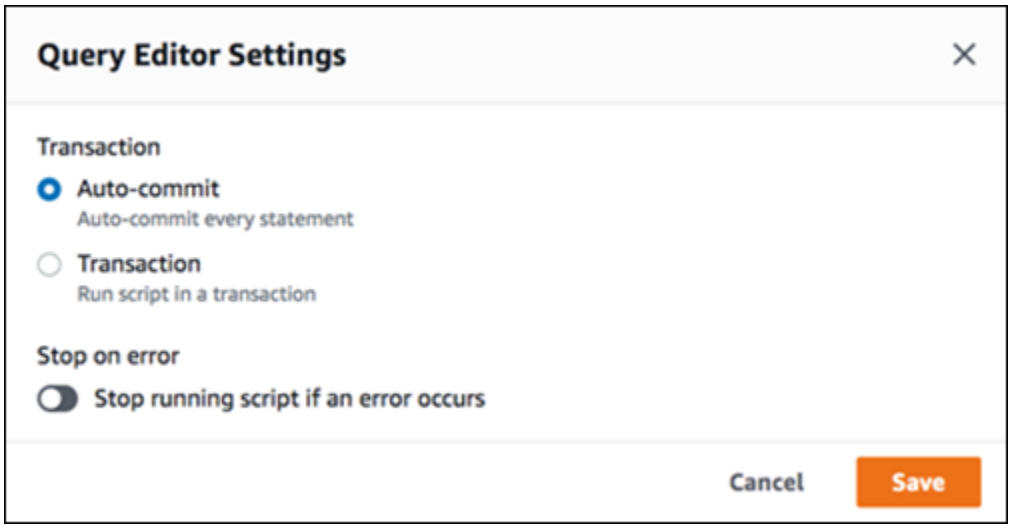
> **Note**
>
> Jika koneksi berhasil, informasi koneksi dan autentikasi Anda akan disimpan di AWS Secrets Manager. Anda tidak perlu memasukkan informasi koneksi lagi.

7. Di editor kueri, masukkan kueri SQL yang ingin dijalankan di basis data.



Setiap pernyataan SQL dapat dilakukan secara otomatis, atau Anda dapat menjalankan pernyataan SQL dalam skrip sebagai bagian dari transaksi. Untuk mengontrol perilaku ini, pilih ikon roda gigi di atas jendela kueri.

Jendela Pengaturan Editor Kueri muncul.



Jika Anda memilih Lakukan otomatis, setiap pernyataan SQL akan dilakukan secara otomatis. Jika Anda memilih Transaksi, Anda dapat menjalankan grup pernyataan dalam skrip. Pernyataan secara otomatis dilakukan di akhir skrip kecuali secara eksplisit dilakukan atau dibatalkan sebelum itu. Selain itu, Anda dapat memilih untuk menghentikan skrip yang sedang berjalan jika terjadi kesalahan dengan mengaktifkan Hentikan saat terjadi kesalahan.

> **Note**
>
> Dalam grup pernyataan, pernyataan bahasa definisi data (DDL) dapat menyebabkan pernyataan bahasa manipulasi data (DML) sebelumnya dilakukan. Anda juga dapat memasukkan pernyataan `COMMIT` dan `ROLLBACK` dalam grup pernyataan dalam skrip.

Setelah Anda menentukan pilihan di jendela Pengaturan Editor Kueri, pilih Simpan.

8. Pilih Jalankan atau tekan Ctrl+Enter, dan editor kueri akan menampilkan hasil kueri Anda.

   Setelah menjalankan kueri, simpan ke Kueri tersimpan dengan memilih Simpan.

   Ekspor hasil kueri ke format spreadsheet dengan memilih Ekspor ke csv.

Anda dapat menemukan, mengedit, dan menjalankan kembali kueri sebelumnya. Untuk melakukannya, pilih tab Baru atau tab Kueri tersimpan, pilih teks kueri, lalu pilih Jalankan.

Untuk mengubah basis data, pilih Ubah basis data.

Referensi API Layanan Metadata Kueri Basis Data (DBQMS)

Layanan Metadata Kueri Basis Data (`dbqms`) adalah layanan khusus internal. Layanan ini menyediakan kueri terbaru dan tersimpan untuk editor kueri pada AWS Management Console untuk beberapa layanan AWS, termasuk Amazon RDS.

Tindakan DBQMS berikut didukung:

Topik

- CreateFavoriteQuery
- CreateQueryHistory
- CreateTab
- DeleteFavoriteQueries
- DeleteQueryHistory
- DeleteTab
- DescribeFavoriteQueries

- DescribeQueryHistory
- DescribeTabs
- GetQueryString
- UpdateFavoriteQuery
- UpdateQueryHistory
- UpdateTab

CreateFavoriteQuery

Menyimpan kueri favorit baru. Setiap pengguna dapat membuat hingga 1.000 kueri yang disimpan. Batas ini dapat berubah di masa mendatang.

CreateQueryHistory

Menyimpan entri riwayat kueri baru.

CreateTab

Menyimpan tab kueri baru. Setiap pengguna dapat membuat hingga 10 tab kueri.

DeleteFavoriteQueries

Menghapus satu atau beberapa kueri tersimpan.

DeleteQueryHistory

Menghapus entri riwayat kueri.

DeleteTab

Menghapus entri tab kueri.

DescribeFavoriteQueries

Mencantumkan kueri disimpan yang dibuat oleh pengguna dalam akun tertentu.

DescribeQueryHistory

Mencantumkan entri riwayat kueri.

DescribeTabs

Mencantumkan tab kueri yang dibuat oleh pengguna dalam akun tertentu.

GetQueryString

Mengambil teks kueri penuh dari ID kueri.

UpdateFavoriteQuery

Memperbarui string, deskripsi, nama, atau tanggal kedaluwarsa kueri.

UpdateQueryHistory

Memperbarui status riwayat kueri.

UpdateTab

Memperbarui nama tab kueri dan string kueri.

Menggunakan machine learning Amazon Aurora

Dengan menggunakan pembelajaran mesin Amazon Aurora, Anda dapat mengintegrasikan cluster Aurora DB Anda dengan salah satu layanan pembelajaran AWS mesin berikut, tergantung pada kebutuhan Anda. Mereka masing-masing mendukung kasus penggunaan pembelajaran mesin tertentu.

Amazon Bedrock

Amazon Bedrock adalah layanan yang dikelola sepenuhnya yang membuat model fondasi terkemuka dari perusahaan AI tersedia melalui API, bersama dengan alat pengembang untuk membantu membangun dan menskalakan aplikasi AI generatif. Dengan Amazon Bedrock, Anda membayar untuk menjalankan inferensi pada salah satu model yayasan pihak ketiga. Harga didasarkan pada volume token input dan token output, dan apakah Anda telah membeli throughput yang disediakan untuk model tersebut. Untuk informasi lebih lanjut, lihat Apa itu Amazon Bedrock? di Panduan Pengguna Amazon Bedrock.

Amazon Comprehend

Amazon Comprehend adalah layanan pemrosesan bahasa alami (NLP) terkelola yang digunakan untuk mengekstraksi wawasan dari dokumen. Dengan Amazon Comprehend, Anda dapat menyimpulkan sentimen berdasarkan konten dokumen, dengan menganalisis entitas, frasa kunci, bahasa, dan fitur lainnya. Untuk mempelajari selengkapnya, lihat Apa itu Amazon Comprehend? di Panduan Pengembang Amazon Comprehend.

SageMaker

Amazon SageMaker adalah layanan pembelajaran mesin yang dikelola sepenuhnya. Ilmuwan dan pengembang data menggunakan Amazon SageMaker untuk membangun, melatih, dan menguji model pembelajaran mesin untuk berbagai tugas inferensi, seperti deteksi penipuan dan rekomendasi produk. Ketika model pembelajaran mesin siap digunakan dalam produksi, model ini dapat diterapkan ke lingkungan yang SageMaker dihosting Amazon. Untuk informasi selengkapnya, lihat Apa itu Amazon SageMaker? di Panduan SageMaker Pengembang Amazon.

Menggunakan Amazon Comprehend dengan cluster Aurora DB Anda memiliki pengaturan awal yang lebih sedikit daripada menggunakan. SageMaker Jika Anda baru mengenal pembelajaran AWS mesin, kami sarankan Anda memulai dengan menjelajahi Amazon Comprehend.

Topik

- Menggunakan machine learning Amazon Aurora dengan Aurora MySQL
- Menggunakan machine learning Amazon Aurora dengan Aurora PostgreSQL

Menggunakan machine learning Amazon Aurora dengan Aurora MySQL

Dengan menggunakan pembelajaran mesin Amazon Aurora dengan cluster DB MySQL Aurora Anda, Anda dapat menggunakan Amazon Bedrock, Amazon Comprehend, atau Amazon, tergantung pada kebutuhan Anda. SageMaker Mereka masing-masing mendukung kasus penggunaan pembelajaran mesin yang berbeda.

Daftar Isi

- Persyaratan untuk menggunakan machine learning Aurora dengan Aurora MySQL
- Wilayah dan ketersediaan versi
- Fitur yang didukung dan batasan machine learning Aurora dengan Aurora MySQL
- Menyiapkan klaster DB Aurora MySQL untuk menggunakan machine learning Aurora
  - Menyiapkan cluster DB MySQL Aurora Anda untuk menggunakan Amazon Bedrock
  - Menyiapkan klaster DB Aurora MySQL untuk menggunakan Amazon Comprehend
  - Menyiapkan cluster DB MySQL Aurora Anda untuk digunakan SageMaker
    - Menyiapkan cluster DB MySQL Aurora Anda untuk menggunakan Amazon S3 untuk (Opsional) SageMaker
  - Memberikan akses pengguna basis data ke machine learning Aurora
    - Memberikan akses ke fungsi Amazon Bedrock
    - Memberikan akses ke fungsi Amazon Comprehend
    - Memberikan akses ke fungsi SageMaker
- Menggunakan Amazon Bedrock dengan cluster DB MySQL Aurora Anda
- Menggunakan Amazon Comprehend dengan klaster DB Aurora MySQL
- Menggunakan SageMaker dengan cluster DB MySQL Aurora Anda
  - Persyaratan set karakter untuk SageMaker fungsi yang mengembalikan string
  - Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model (Lanjutan)
- Pertimbangan performa untuk machine learning Aurora dengan Aurora MySQL
  - Model dan prompt

  - Cache kueri
  - Optimalisasi batch untuk panggilan fungsi machine learning Aurora
- Memantau machine learning Aurora

Persyaratan untuk menggunakan machine learning Aurora dengan Aurora MySQL

AWS Layanan pembelajaran mesin adalah layanan terkelola yang diatur dan dijalankan di lingkungan produksi mereka sendiri. Pembelajaran mesin Aurora mendukung integrasi dengan Amazon Bedrock, Amazon Comprehend, dan. SageMaker Sebelum mencoba menyiapkan klaster DB Aurora MySQL untuk menggunakan machine learning Aurora, pastikan Anda memahami persyaratan dan prasyarat berikut.

- Layanan pembelajaran mesin harus berjalan sama Wilayah AWS dengan cluster DB MySQL Aurora Anda. Anda tidak dapat menggunakan layanan pembelajaran mesin dari cluster DB MySQL Aurora di Wilayah yang berbeda.
- Jika cluster DB MySQL Aurora Anda berada di cloud publik virtual (VPC) yang berbeda dari Amazon Bedrock, Amazon Comprehend, SageMaker atau layanan Anda, grup Keamanan VPC perlu mengizinkan koneksi keluar ke layanan pembelajaran mesin Aurora target. Lihat informasi yang lebih lengkap di Mengendalikan lalu lintas ke sumber daya AWS dengan menggunakan grup keamanan dalam Panduan Pengguna Amazon VPC.
- Anda dapat meningkatkan klaster Aurora yang menjalankan Aurora MySQL versi yang lebih rendah ke versi yang lebih tinggi yang didukung jika Anda ingin menggunakan machine learning Aurora dengan klaster tersebut. Untuk informasi selengkapnya, lihat Pembaruan mesin basis data untuk Amazon Aurora MySQL.
- Cluster Aurora MySQL DB Anda harus menggunakan grup parameter cluster DB kustom. Di akhir proses penyiapan untuk setiap layanan machine learning Aurora yang ingin Anda gunakan, tambahkan Amazon Resource Name (ARN) dari peran IAM terkait yang dibuat untuk layanan tersebut. Sebaiknya buat grup parameter klaster DB kustom untuk Aurora MySQL Anda terlebih dahulu dan konfigurasikan klaster DB Aurora MySQL Anda untuk menggunakannya sehingga siap untuk Anda modifikasi di akhir proses penyiapan.
- Untuk SageMaker:
  - Komponen pembelajaran mesin yang ingin Anda gunakan untuk kesimpulan harus diatur dan siap digunakan. Selama proses konfigurasi untuk cluster DB MySQL Aurora Anda, pastikan untuk memiliki ARN dari endpoint yang tersedia. SageMaker Para ilmuwan data di tim Anda

mungkin paling mampu menangani bekerja dengan SageMaker untuk mempersiapkan model dan menangani tugas-tugas lain semacam itu. Untuk memulai dengan Amazon SageMaker, lihat Memulai dengan Amazon SageMaker. Untuk informasi selengkapnya tentang inferensi dan titik akhir, lihat Inferensi real-time.

- Untuk menggunakan SageMaker data pelatihan Anda sendiri, Anda harus menyiapkan bucket Amazon S3 sebagai bagian dari konfigurasi Aurora MySQL Anda untuk pembelajaran mesin Aurora. Untuk melakukannya, Anda mengikuti proses umum yang sama seperti untuk menyiapkan SageMaker integrasi. Untuk ringkasan proses penyiapan opsional ini, lihat Menyiapkan cluster DB MySQL Aurora Anda untuk menggunakan Amazon S3 untuk (Opsional) SageMaker .

- Untuk database global Aurora, Anda menyiapkan layanan pembelajaran mesin Aurora yang ingin Anda gunakan dalam semua yang Wilayah AWS membentuk basis data global Aurora Anda. Misalnya, jika Anda ingin menggunakan pembelajaran mesin Aurora untuk database global Aurora Anda, Anda melakukan hal berikut SageMaker untuk setiap cluster DB MySQL Aurora di setiap: Wilayah AWS
  - Siapkan SageMaker layanan Amazon dengan model SageMaker pelatihan dan titik akhir yang sama. Ini juga harus menggunakan nama yang sama.
  - Buat peran IAM seperti yang dijelaskan dalam Menyiapkan klaster DB Aurora MySQL untuk menggunakan machine learning Aurora.
  - Tambahkan ARN peran IAM ke grup parameter klaster DB kustom untuk setiap klaster DB Aurora MySQL di setiap Wilayah AWS.

  Tugas-tugas ini mengharuskan pembelajaran mesin Aurora tersedia untuk versi Aurora MySQL Anda di semua yang Wilayah AWS membentuk basis data global Aurora Anda.

Wilayah dan ketersediaan versi

Ketersediaan fitur dan dukungan bervariasi di seluruh versi khusus dari setiap mesin basis data Aurora, dan di seluruh Wilayah AWS.

- Untuk informasi tentang versi dan ketersediaan Wilayah untuk Amazon Comprehend dan SageMaker Amazon dengan Aurora MySQL, lihat. Machine learning Aurora dengan Aurora MySQL
- Amazon Bedrock hanya didukung pada Aurora MySQL versi 3.06 dan lebih tinggi.

  Untuk informasi tentang ketersediaan Wilayah untuk Amazon Bedrock, lihat Model yang didukung di Amazon Bedrock di Panduan Pengguna Amazon Bedrock.

Fitur yang didukung dan batasan machine learning Aurora dengan Aurora MySQL

Saat menggunakan Aurora MySQL dengan pembelajaran mesin Aurora, batasan berikut berlaku:

- Ekstensi pembelajaran mesin Aurora tidak mendukung antarmuka vektor.
- Integrasi pembelajaran mesin Aurora tidak didukung saat digunakan dalam pemicu.
- Fungsi pembelajaran mesin Aurora tidak kompatibel dengan replikasi biner logging (binlog).
  - Pengaturan `--binlog-format=STATEMENT` memunculkan pengecualian untuk panggilan ke fungsi machine learning Aurora.
  - Fungsi machine learning Aurora Function bersifat nondeterministik, dan fungsi tersimpan nondeterministik tidak kompatibel dengan format binlog.

  Untuk informasi selengkapnya, lihat Format Pencatatan Biner dalam dokumentasi MySQL.
- Fungsi tersimpan yang memanggil tabel dengan kolom generated-always tidak didukung. Ini berlaku untuk semua fungsi tersimpan Aurora MySQL. Untuk mempelajari lebih lanjut tentang jenis kolom ini, lihat CREATE TABLE and Generated Columns dalam dokumentasi MySQL.
- Fungsi Amazon Bedrock tidak mendukung`RETURNS JSON`. Anda dapat menggunakan `CONVERT` atau `CAST` mengonversi dari `TEXT` ke `JSON` jika diperlukan.
- Amazon Bedrock tidak mendukung permintaan batch.
- Aurora MySQL mendukung SageMaker titik akhir apa pun yang membaca dan menulis format nilai dipisahkan koma (CSV), melalui format file. `ContentType text/csv` Format ini diterima oleh SageMaker algoritma bawaan berikut:
  - Linear Learner
  - Random Cut Forest
  - XGBoost

  Untuk mempelajari lebih lanjut tentang algoritme ini, lihat Memilih Algoritma di Panduan SageMaker Pengembang Amazon.

Menyiapkan klaster DB Aurora MySQL untuk menggunakan machine learning Aurora

Dalam topik berikut, Anda dapat menemukan prosedur pengaturan terpisah untuk masing-masing layanan machine learning Aurora ini.

Topik

- Menyiapkan cluster DB MySQL Aurora Anda untuk menggunakan Amazon Bedrock
- Menyiapkan klaster DB Aurora MySQL untuk menggunakan Amazon Comprehend
- Menyiapkan cluster DB MySQL Aurora Anda untuk digunakan SageMaker
  - Menyiapkan cluster DB MySQL Aurora Anda untuk menggunakan Amazon S3 untuk (Opsional) SageMaker
- Memberikan akses pengguna basis data ke machine learning Aurora
  - Memberikan akses ke fungsi Amazon Bedrock
  - Memberikan akses ke fungsi Amazon Comprehend
  - Memberikan akses ke fungsi SageMaker

Menyiapkan cluster DB MySQL Aurora Anda untuk menggunakan Amazon Bedrock

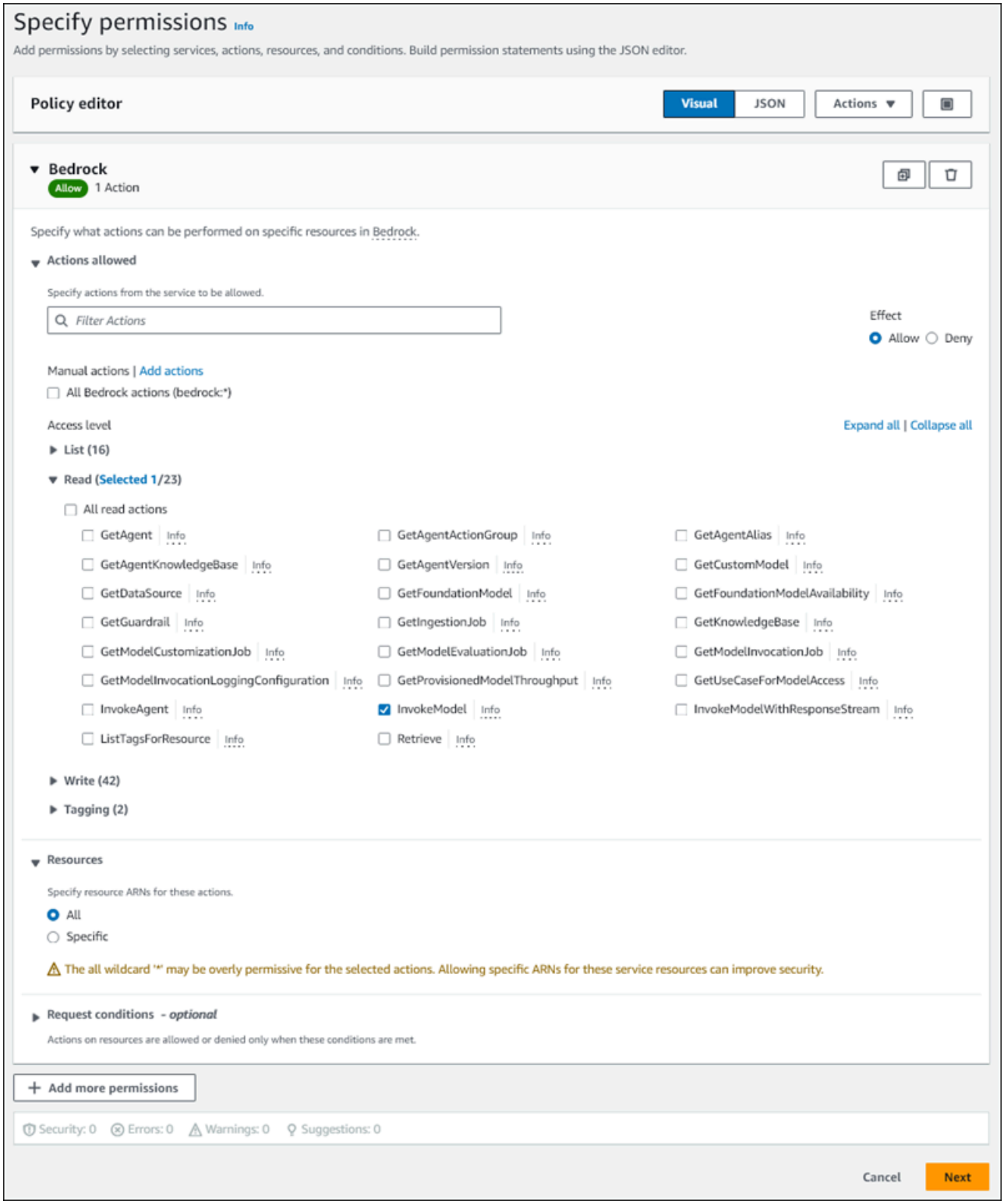
Pembelajaran mesin Aurora bergantung pada peran dan kebijakan AWS Identity and Access Management (IAM) untuk memungkinkan klaster DB MySQL Aurora Anda mengakses dan menggunakan layanan Amazon Bedrock. Prosedur berikut membuat kebijakan dan peran izin IAM sehingga kluster DB Anda dapat berintegrasi dengan Amazon Bedrock.

Untuk membuat kebijakan IAM

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Pilih Kebijakan di panel navigasi.
3. Pilih Buat kebijakan.
4. Pada halaman Tentukan izin, untuk Pilih layanan, pilih Batuan Dasar.

   Tampilan izin Amazon Bedrock.
5. Perluas Baca, lalu pilih InvokeModel.
6. Untuk Sumber Daya, pilih Semua.

   Halaman Tentukan izin harus menyerupai gambar berikut.



7. Pilih Berikutnya.
8. Pada halaman Tinjau dan buat, masukkan nama untuk kebijakan Anda, misalnya**`BedrockInvokeModel`**.

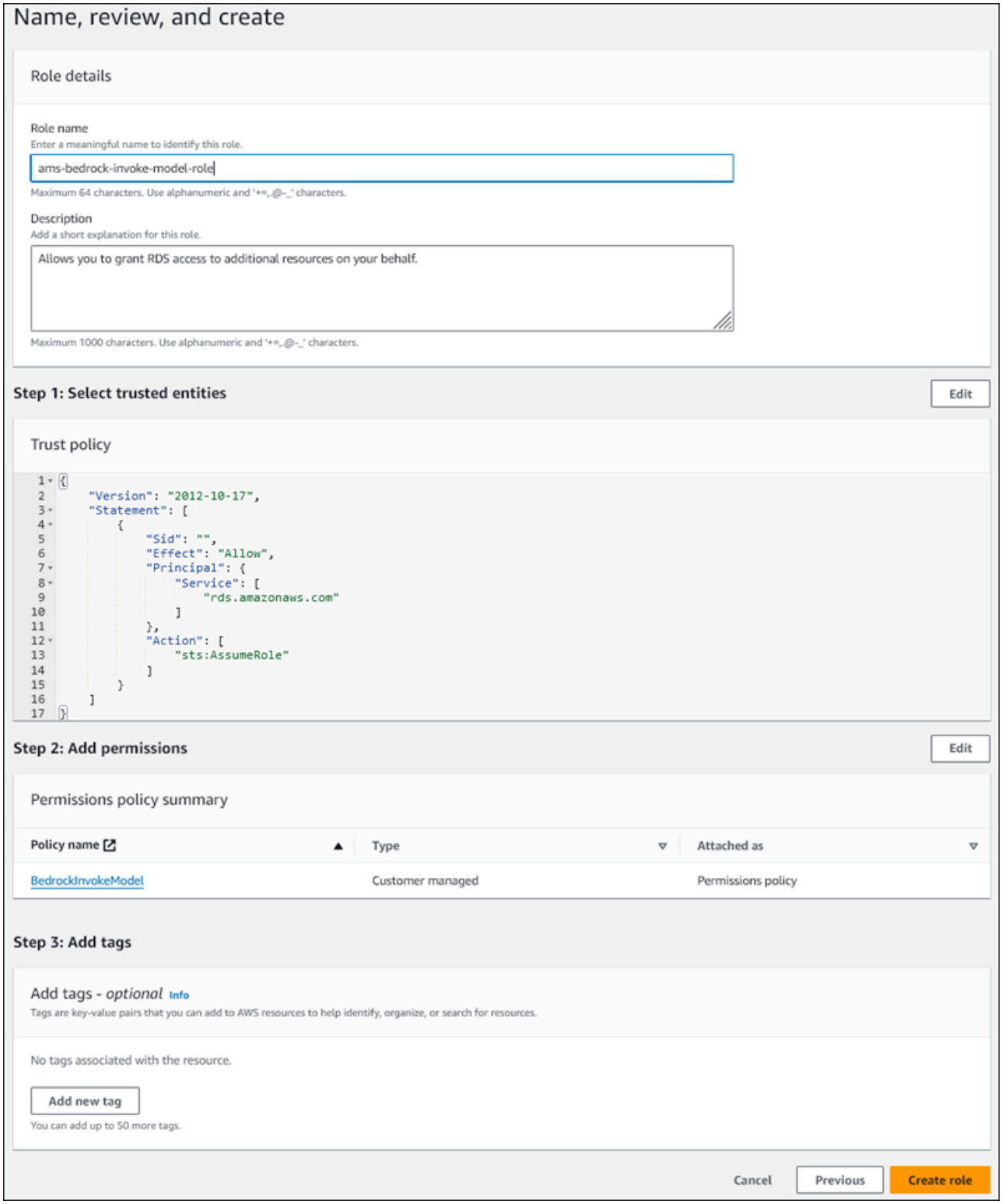
9. Tinjau kebijakan Anda, lalu pilih Buat kebijakan.

Selanjutnya Anda membuat peran IAM yang menggunakan kebijakan izin Amazon Bedrock.

Untuk membuat peran IAM

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Pilih Peran di panel navigasi.
3. Pilih Buat peran.
4. Pada halaman Pilih entitas tepercaya, untuk kasus Penggunaan, pilih RDS.
5. Pilih RDS - Tambahkan Peran ke Database, lalu pilih Berikutnya.
6. Pada halaman Tambahkan izin, untuk kebijakan Izin, pilih kebijakan IAM yang Anda buat, lalu pilih Berikutnya.
7. Pada halaman Nama, tinjau, dan buat, masukkan nama untuk peran Anda, misalnya**`ams-bedrock-invoke-model-role`**.

   Peran harus menyerupai gambar berikut.



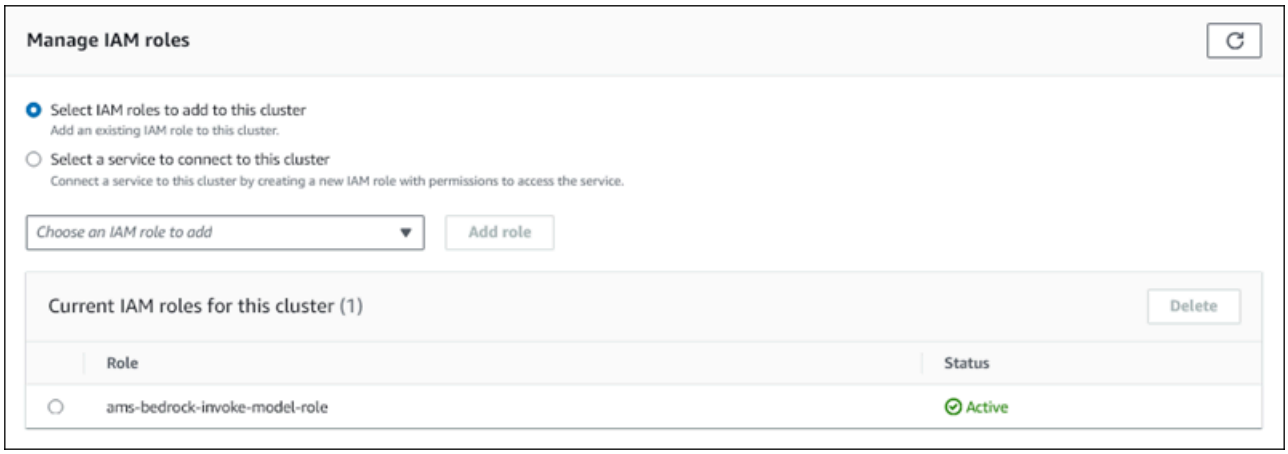
8. Tinjau peran Anda, lalu pilih Buat peran.

Selanjutnya Anda mengaitkan peran Amazon Bedrock IAM dengan cluster DB Anda.

Untuk mengaitkan peran IAM dengan cluster DB Anda

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data dari panel navigasi.
3. Pilih cluster DB MySQL Aurora yang ingin Anda sambungkan ke layanan Amazon Bedrock.
4. Pilih tab Konektivitas & keamanan.
5. Untuk bagian Kelola peran IAM, pilih Pilih IAM untuk ditambahkan ke klaster ini.
6. Pilih IAM yang Anda buat, lalu pilih Tambah peran.

   Peran IAM dikaitkan dengan cluster DB Anda, pertama dengan status Tertunda, lalu Aktif. Setelah proses selesai, Anda dapat menemukan peran tersebut di daftar Peran IAM saat ini untuk klaster ini.



Anda harus menambahkan ARN peran IAM ini ke parameter grup `aws_default_bedrock_role` parameter cluster DB kustom yang terkait dengan cluster DB MySQL Aurora Anda. Jika klaster DB Aurora MySQL Anda tidak menggunakan grup parameter klaster DB kustom, Anda perlu membuatnya untuk digunakan dengan klaster DB Aurora MySQL Anda untuk menyelesaikan integrasi. Untuk informasi selengkapnya, lihat Bekerja dengan grup parameter klaster DB.

Untuk mengkonfigurasi parameter cluster DB

1. Di Konsol Amazon RDS, buka tab Konfigurasi dari klaster DB Aurora MySQL Anda.
2. Temukan grup parameter cluster DB yang dikonfigurasi untuk cluster Anda. Pilih tautan untuk membuka grup parameter cluster DB kustom Anda, lalu pilih Edit.

3. Temukan parameter `aws_default_bedrock_role` dalam grup parameter klaster DB kustom Anda.
4. Di bidang Nilai, masukkan ARN dari peran IAM.
5. Pilih Simpan perubahan untuk menyimpan pengaturan.
6. Boot ulang instans utama klaster DB Aurora MySQL Anda sehingga pengaturan parameter ini berlaku.

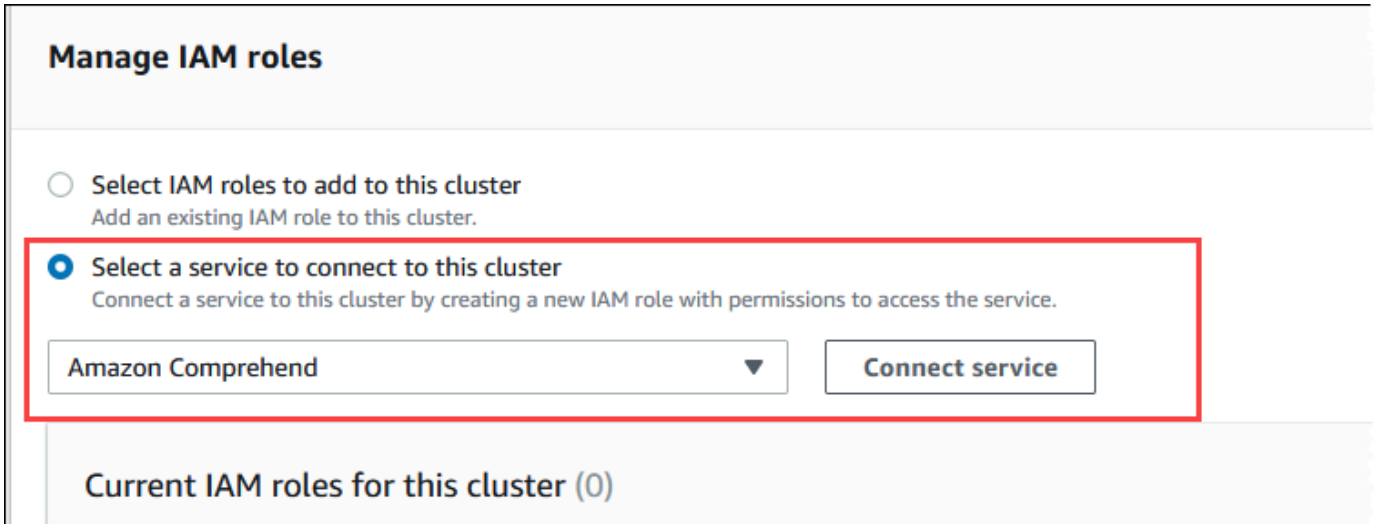
Integrasi IAM untuk Amazon Bedrock selesai. Lanjutkan menyiapkan cluster DB MySQL Aurora Anda untuk bekerja dengan Amazon Bedrock oleh. Memberikan akses pengguna basis data ke machine learning Aurora

Menyiapkan klaster DB Aurora MySQL untuk menggunakan Amazon Comprehend

Pembelajaran mesin Aurora bergantung pada AWS Identity and Access Management peran dan kebijakan untuk memungkinkan klaster DB MySQL Aurora Anda mengakses dan menggunakan layanan Amazon Comprehend. Prosedur berikut secara otomatis membuat kebijakan dan peran IAM untuk klaster Anda sehingga dapat menggunakan Amazon Comprehend.

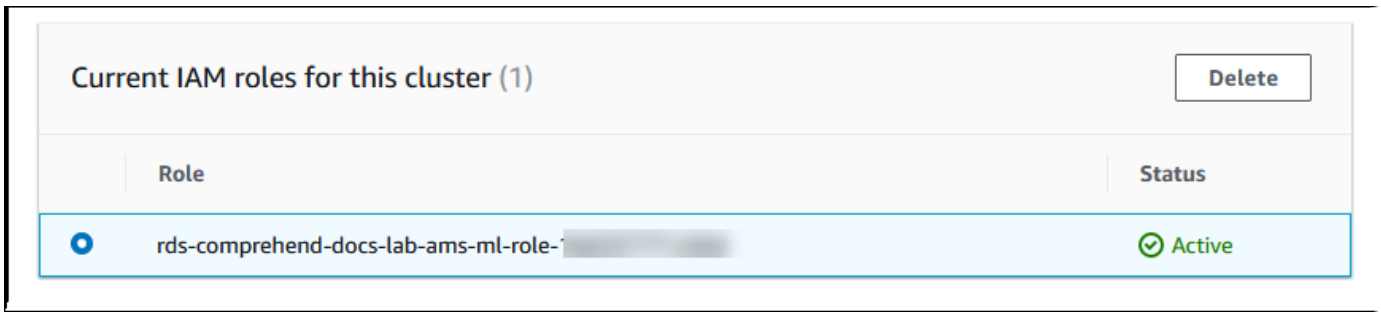
Untuk menyiapkan klaster DB Aurora MySQL untuk menggunakan Amazon Comprehend

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Basis data dari panel navigasi.
3. Pilih cluster DB MySQL Aurora yang ingin Anda sambungkan ke layanan Amazon Comprehend.
4. Pilih tab Konektivitas & keamanan.
5. Untuk bagian Kelola peran IAM, pilih Pilih layanan untuk terhubung ke klaster ini.
6. Pilih Amazon Comprehend dari menu, lalu pilih Connect service.



7. Dialog Hubungkan klaster ke Amazon Comprehend tidak memerlukan informasi tambahan apa pun. Namun, Anda mungkin melihat pesan yang memberi tahu Anda bahwa integrasi antara Aurora dan Amazon Comprehend saat ini sedang dalam pratinjau. Pastikan untuk membaca pesan tersebut sebelum melanjutkan. Anda dapat memilih Batal jika Anda memilih untuk tidak melanjutkan.
8. Pilih Hubungkan layanan untuk menyelesaikan proses integrasi.

   Aurora menciptakan peran IAM. Ini juga membuat kebijakan yang memungkinkan klaster DB MySQL Aurora menggunakan layanan Amazon Comprehend dan melampirkan kebijakan ke peran tersebut. Setelah proses selesai, Anda dapat menemukan peran tersebut di daftar Peran IAM saat ini untuk klaster ini seperti yang ditunjukkan pada gambar berikut.

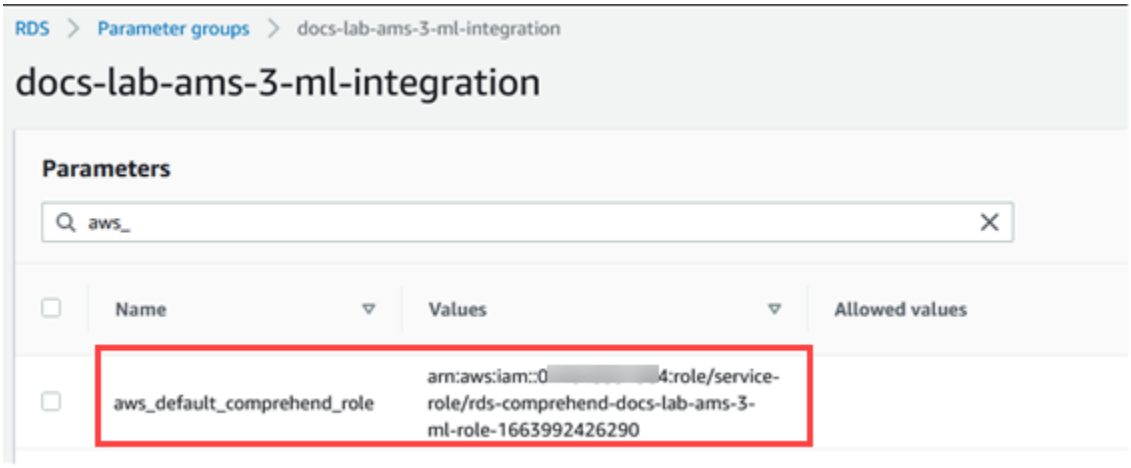


   Anda perlu menambahkan ARN peran IAM ini ke parameter grup `aws_default_comprehend_role` parameter cluster DB kustom yang terkait dengan cluster DB MySQL Aurora Anda. Jika klaster DB Aurora MySQL Anda tidak menggunakan grup parameter klaster DB kustom, Anda perlu membuatnya untuk digunakan dengan klaster DB Aurora MySQL Anda untuk menyelesaikan integrasi. Untuk informasi selengkapnya, lihat Bekerja dengan grup parameter klaster DB.

Setelah membuat grup parameter klaster DB kustom Anda dan mengaitkannya dengan klaster DB Aurora MySQL Anda, Anda dapat melanjutkan mengikuti langkah-langkah ini.

Jika klaster Anda menggunakan grup parameter klaster DB kustom, lakukan hal berikut.

a. Di Konsol Amazon RDS, buka tab Konfigurasi dari klaster DB Aurora MySQL Anda.
b. Temukan grup parameter cluster DB yang dikonfigurasi untuk cluster Anda. Pilih tautan untuk membuka grup parameter cluster DB kustom Anda, lalu pilih Edit.
c. Temukan parameter `aws_default_comprehend_role` dalam grup parameter klaster DB kustom Anda.
d. Di bidang Nilai, masukkan ARN dari peran IAM.
e. Pilih Simpan perubahan untuk menyimpan pengaturan. Pada gambar berikut, Anda dapat melihat contohnya.



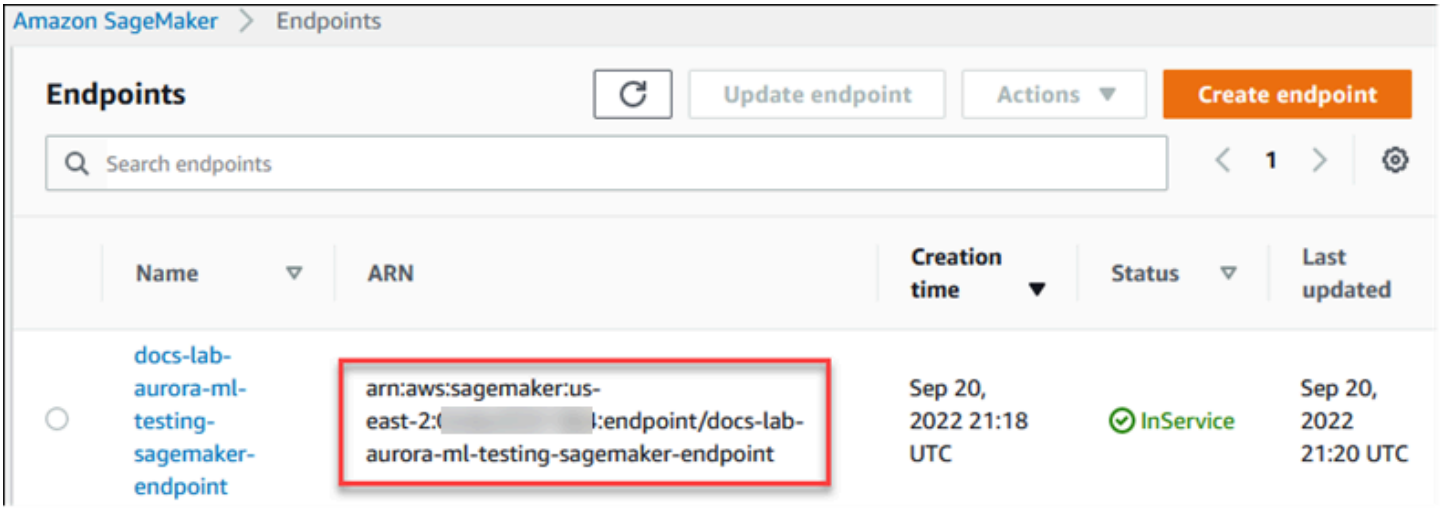
Boot ulang instans utama klaster DB Aurora MySQL Anda sehingga pengaturan parameter ini berlaku.

Integrasi IAM untuk Amazon Comprehend selesai. Lanjutkan penyiapan klaster DB Aurora MySQL Anda untuk bekerja dengan Amazon Comprehend dengan memberikan akses ke pengguna basis data yang sesuai.

Menyiapkan cluster DB MySQL Aurora Anda untuk digunakan SageMaker

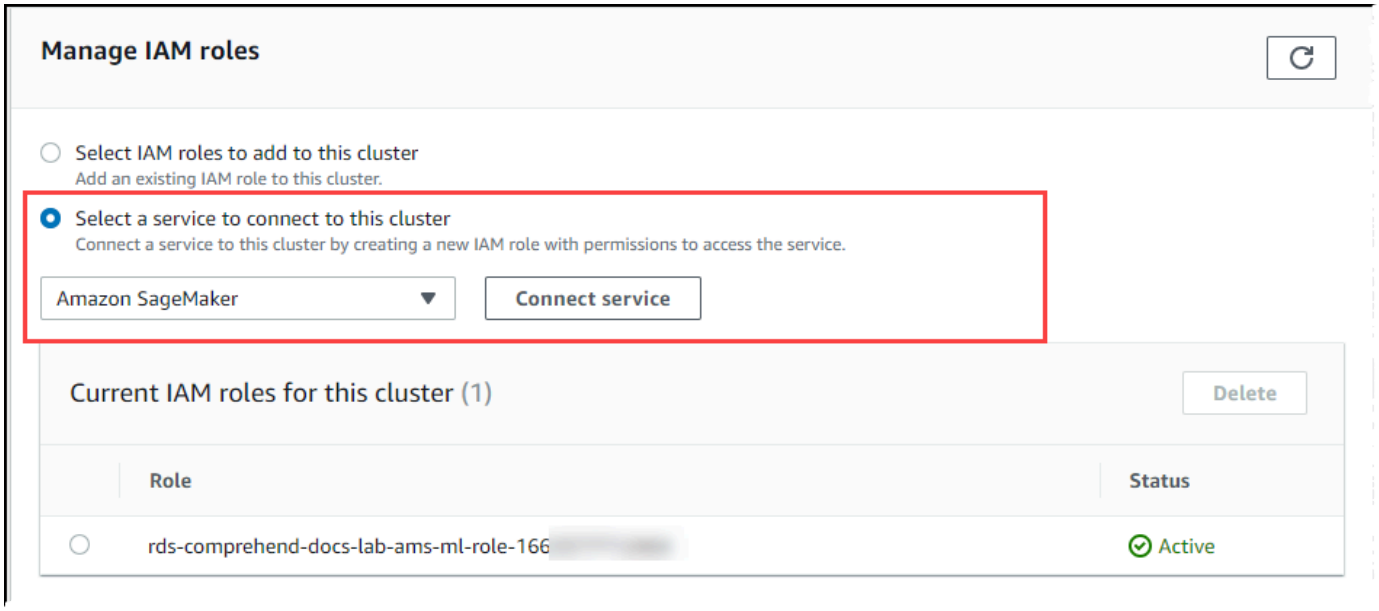
Prosedur berikut secara otomatis membuat peran dan kebijakan IAM untuk cluster Aurora MySQL DB Anda sehingga dapat digunakan. SageMaker Sebelum mencoba mengikuti prosedur ini, pastikan Anda memiliki SageMaker titik akhir yang tersedia sehingga Anda dapat memasukkannya saat

diperlukan. Biasanya, ilmuwan data di tim Anda akan melakukan pekerjaan untuk menghasilkan titik akhir yang dapat Anda gunakan dari klaster DB Aurora MySQL Anda. Anda dapat menemukan titik akhir seperti itu di SageMaker konsol. Di panel navigasi, buka menu Inferensi dan pilih Titik akhir. Pada gambar berikut, Anda dapat melihat contohnya.

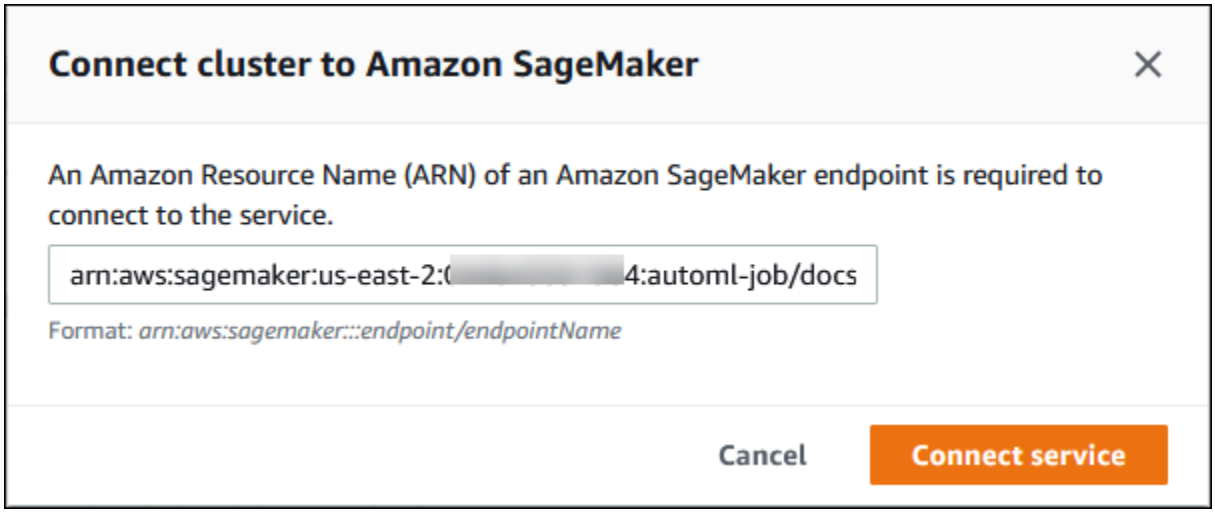


Untuk mengatur cluster DB MySQL Aurora Anda untuk digunakan SageMaker

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Database dari menu navigasi Amazon RDS dan kemudian pilih cluster Aurora MySQL DB yang ingin Anda sambungkan ke layanan. SageMaker
3. Pilih tab Konektivitas & keamanan.
4. Gulir ke bagian Kelola peran IAM lalu Pilih layanan untuk dihubungkan ke klaster ini. Pilih SageMakerdari pemilih.



5. Pilih Hubungkan layanan.
6. Dalam Connect cluster to SageMaker dialog, masukkan ARN dari endpoint. SageMaker



7. Aurora menciptakan peran IAM. Ini juga membuat kebijakan yang memungkinkan cluster DB MySQL Aurora untuk menggunakan SageMaker layanan dan melampirkan kebijakan ke peran tersebut. Setelah proses selesai, Anda dapat menemukan peran tersebut di daftar Peran IAM saat ini untuk klaster ini.
8. Buka konsol IAM di https://console.aws.amazon.com/iam/.
9. Pilih Peran dari bagian Manajemen akses pada menu navigasi AWS Identity and Access Management .
10. Temukan peran yang dicari di daftar peran. Namanya menggunakan pola berikut.

```
rds-sagemaker-your-cluster-name-role-auto-generated-digits
```

11. Buka halaman Ringkasan peran dan temukan ARN. Catat ARN atau salin menggunakan widget salin.
12. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
13. Pilih klaster DB Aurora MySQL Anda, lalu pilih tab Konfigurasi.
14. Temukan grup parameter klaster DB, dan pilih tautan untuk membuka grup parameter klaster DB kustom Anda. Temukan parameter `aws_default_sagemaker_role` dan masukkan ARN peran IAM di bidang Nilai dan Simpan pengaturan.
15. Boot ulang instans utama klaster DB Aurora MySQL Anda sehingga pengaturan parameter ini berlaku.

Pengaturan IAM sudah selesai. Lanjutkan menyiapkan cluster DB MySQL Aurora Anda untuk bekerja dengan SageMaker dengan memberikan akses ke pengguna database yang sesuai.

Jika Anda ingin menggunakan SageMaker model Anda untuk pelatihan daripada menggunakan SageMaker komponen pra-bangun, Anda juga perlu menambahkan bucket Amazon S3 ke cluster DB MySQL Aurora Anda, seperti yang diuraikan dalam berikut ini. Menyiapkan cluster DB MySQL Aurora Anda untuk menggunakan Amazon S3 untuk (Opsional) SageMaker

Menyiapkan cluster DB MySQL Aurora Anda untuk menggunakan Amazon S3 untuk (Opsional) SageMaker

Untuk menggunakan SageMaker model Anda sendiri daripada menggunakan komponen pra-bangun yang disediakan oleh SageMaker, Anda perlu menyiapkan bucket Amazon S3 untuk digunakan oleh cluster DB MySQL Aurora. Untuk informasi selengkapnya tentang membuat bucket Amazon S3, lihat Membuat bucket di Panduan Pengguna Amazon Simple Storage Service.

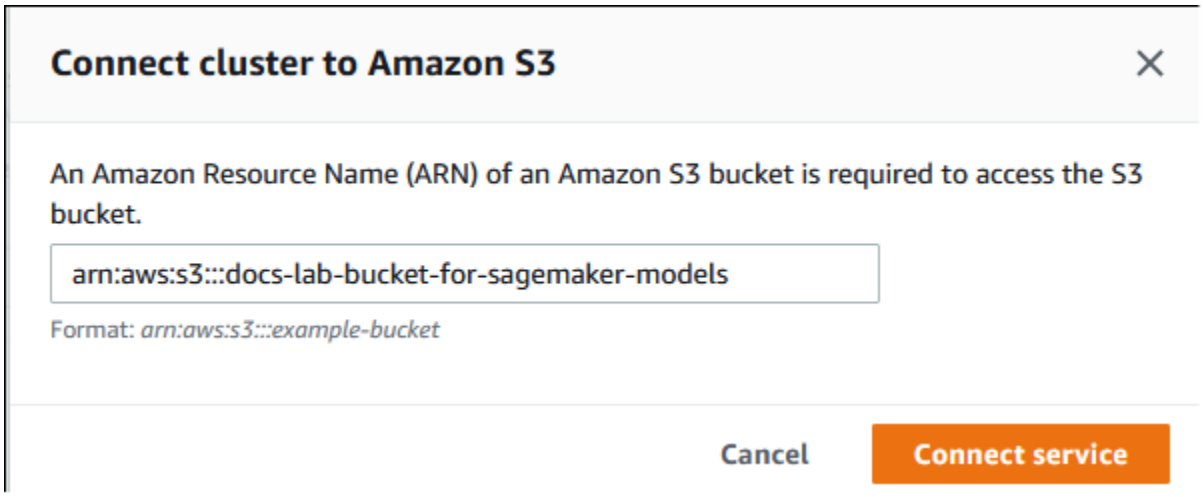
Untuk menyiapkan klaster DB MySQL Aurora Anda untuk menggunakan bucket Amazon S3 SageMaker

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Pilih Database dari menu navigasi Amazon RDS dan kemudian pilih cluster Aurora MySQL DB yang ingin Anda sambungkan ke layanan. SageMaker
3. Pilih tab Konektivitas & keamanan.

4. Gulir ke bagian Kelola peran IAM lalu Pilih layanan untuk dihubungkan ke klaster ini. Pilih Amazon S3 dari pemilih.

5. Pilih Hubungkan layanan.

6. Dalam dialog Connect cluster to Amazon S3, masukkan ARN bucket Amazon S3, seperti yang ditunjukkan pada gambar berikut.



7. Pilih Hubungkan layanan untuk menyelesaikan proses ini.

Untuk informasi selengkapnya tentang menggunakan bucket Amazon S3 dengan SageMaker, lihat Menentukan Bucket Amazon S3 untuk Mengunggah Kumpulan Data Pelatihan dan Menyimpan Data Output di Panduan Pengembang Amazon. SageMaker Untuk mempelajari lebih lanjut tentang bekerja dengan SageMaker, lihat Memulai Instans SageMaker Notebook Amazon di Panduan SageMaker Pengembang Amazon.

Memberikan akses pengguna basis data ke machine learning Aurora

Pengguna database harus diberikan izin untuk menjalankan fungsi pembelajaran mesin Aurora. Cara Anda memberikan izin tergantung versi MySQL yang Anda gunakan untuk klaster DB Aurora MySQL Anda, seperti yang diuraikan sebagai berikut. Bagaimana Anda melakukannya tergantung versi MySQL yang digunakan klaster DB Aurora MySQL Anda.

- Untuk Aurora MySQL versi 3 (MySQL 8.0 kompatibel), pengguna database harus diberikan peran database yang sesuai. Untuk informasi selengkapnya, lihat Menggunakan Peran di Manual Referensi MySQL 8.0.

- Untuk Aurora MySQL versi 2 (MySQL 5.7 kompatibel), pengguna database diberikan hak istimewa. Untuk informasi selengkapnya, lihat Kontrol Akses dan Manajemen Akun di Manual Referensi MySQL 5.7.

Tabel berikut menunjukkan peran dan hak istimewa yang dibutuhkan pengguna database untuk bekerja dengan fungsi pembelajaran mesin.

Aurora MySQL versi 3 (peran)	Aurora MySQL versi 2 (hak istimewa)
AWS_BEDROCK_ACCESS	–
AWS_COMPREHEND_ACCESS	INVOKE COMPREHEND
AWS_SAGEMAKER_ACCESS	INVOKE SAGEMAKER

Memberikan akses ke fungsi Amazon Bedrock

Untuk memberi pengguna database akses ke fungsi Amazon Bedrock, gunakan pernyataan SQL berikut:

```
GRANT AWS_BEDROCK_ACCESS TO user@domain-or-ip-address;
```

Pengguna database juga perlu diberikan EXECUTE izin untuk fungsi yang Anda buat untuk bekerja dengan Amazon Bedrock:

```
GRANT EXECUTE ON FUNCTION database_name.function_name TO user@domain-or-ip-address;
```

Fungsi Amazon Bedrock sekarang tersedia untuk digunakan.

Memberikan akses ke fungsi Amazon Comprehend

Untuk memberi pengguna basis data akses ke fungsi Amazon Comprehend, gunakan pernyataan yang sesuai untuk versi Aurora MySQL Anda.

- Aurora MySQL versi 3 (kompatibel dengan MySQL 8.0)

  ```
  GRANT AWS_COMPREHEND_ACCESS TO user@domain-or-ip-address;
  ```

- Aurora MySQL versi 2 (kompatibel dengan MySQL 5.7)

```
GRANT INVOKE COMPREHEND ON *.* TO user@domain-or-ip-address;
```

Fungsi Amazon Comprehend sekarang tersedia untuk digunakan. Untuk contoh penggunaan, lihat Menggunakan Amazon Comprehend dengan klaster DB Aurora MySQL.

Memberikan akses ke fungsi SageMaker

Untuk memberi pengguna database akses ke SageMaker fungsi, gunakan pernyataan yang sesuai untuk versi MySQL Aurora Anda.

- Aurora MySQL versi 3 (kompatibel dengan MySQL 8.0)

```
GRANT AWS_SAGEMAKER_ACCESS TO user@domain-or-ip-address;
```

- Aurora MySQL versi 2 (kompatibel dengan MySQL 5.7)

```
GRANT INVOKE SAGEMAKER ON *.* TO user@domain-or-ip-address;
```

Pengguna database juga perlu diberikan `EXECUTE` izin untuk fungsi yang Anda buat untuk bekerja dengan SageMaker. Misalkan Anda membuat dua fungsi, `db1.anomoly_score` dan`db2.company_forecasts`, untuk memanggil layanan SageMaker endpoint Anda. Anda memberikan hak istimewa eksekusi seperti yang ditunjukkan pada contoh berikut.

```
GRANT EXECUTE ON FUNCTION db1.anomaly_score TO user1@domain-or-ip-address1;
GRANT EXECUTE ON FUNCTION db2.company_forecasts TO user2@domain-or-ip-address2;
```

SageMaker Fungsi sekarang tersedia untuk digunakan. Untuk contoh penggunaan, lihat Menggunakan SageMaker dengan cluster DB MySQL Aurora Anda.

Menggunakan Amazon Bedrock dengan cluster DB MySQL Aurora Anda

Untuk menggunakan Amazon Bedrock, Anda membuat fungsi yang ditentukan pengguna (UDF) di database Aurora MySQL Anda yang memanggil model. Untuk informasi selengkapnya, lihat Model yang didukung di Amazon Bedrock di Panduan Pengguna Amazon Bedrock.

UDF menggunakan sintaks berikut:

```
CREATE FUNCTION function_name (argument type)
```

```
    [DEFINER = user]
RETURNS mysql_data_type
[SQL SECURITY {DEFINER | INVOKER}]
ALIAS AWS_BEDROCK_INVOKE_MODEL
MODEL ID 'model_id'
[CONTENT_TYPE 'content_type']
[ACCEPT 'content_type']
[TIMEOUT_MS timeout_in_milliseconds];
```

- Fungsi Amazon Bedrock tidak mendukungRETURNS JSON. Anda dapat menggunakan CONVERT atau CAST mengonversi dari TEXT ke JSON jika diperlukan.
- Jika Anda tidak menentukan CONTENT_TYPE atauACCEPT, defaultnya adalahapplication/json.
- Jika Anda tidak menentukanTIMEOUT_MS, nilai untuk aurora_ml_inference_timeout digunakan.

Misalnya, UDF berikut memanggil model Amazon Titan Text Express:

```
CREATE FUNCTION invoke_titan (request_body TEXT)
    RETURNS TEXT
    ALIAS AWS_BEDROCK_INVOKE_MODEL
    MODEL ID 'amazon.titan-text-express-v1'
    CONTENT_TYPE 'application/json'
    ACCEPT 'application/json';
```

Untuk memungkinkan pengguna DB menggunakan fungsi ini, gunakan perintah SQL berikut:

```
GRANT EXECUTE ON FUNCTION database_name.invoke_titan TO user@domain-or-ip-address;
```

Kemudian pengguna dapat memanggil invoke_titan seperti fungsi lainnya, seperti yang ditunjukkan pada contoh berikut. Pastikan untuk memformat badan permintaan sesuai dengan model teks Amazon Titan.

```
CREATE TABLE prompts (request varchar(1024));
INSERT INTO prompts VALUES (
'{
    "inputText": "Generate synthetic data for daily product sales in various categories
 - include row number, product name, category, date of sale and price. Produce output
 in JSON format. Count records and ensure there are no more than 5.",
    "textGenerationConfig": {
```

```
        "maxTokenCount": 1024,
        "stopSequences": [],
        "temperature":0,
        "topP":1
    }
}');

SELECT invoke_titan(request) FROM prompts;

{"inputTextTokenCount":44,"results":[{"tokenCount":296,"outputText":"
```tabular-data-json
{
    "rows": [
        {
            "Row Number": "1",
            "Product Name": "T-Shirt",
            "Category": "Clothing",
            "Date of Sale": "2024-01-01",
            "Price": "$20"
        },
        {
            "Row Number": "2",
            "Product Name": "Jeans",
            "Category": "Clothing",
            "Date of Sale": "2024-01-02",
            "Price": "$30"
        },
        {
            "Row Number": "3",
            "Product Name": "Hat",
            "Category": "Accessories",
            "Date of Sale": "2024-01-03",
            "Price": "$15"
        },
        {
            "Row Number": "4",
            "Product Name": "Watch",
            "Category": "Accessories",
            "Date of Sale": "2024-01-04",
            "Price": "$40"
        },
        {
            "Row Number": "5",
            "Product Name": "Phone Case",
```
```

```
            "Category": "Accessories",
            "Date of Sale": "2024-01-05",
            "Price": "$25"
        }
    ]
}
```","completionReason":"FINISH"}]}

Untuk model lain yang Anda gunakan, pastikan untuk memformat badan permintaan dengan tepat untuk mereka. Untuk informasi selengkapnya, lihat Parameter inferensi untuk model foundation di Panduan Pengguna Amazon Bedrock.

Menggunakan Amazon Comprehend dengan klaster DB Aurora MySQL

Untuk Aurora MySQL, machine learning Aurora menyediakan dua fungsi default berikut untuk bekerja dengan Amazon Comprehend dan data teks Anda. Anda memberikan teks untuk menganalisis (`input_data`) dan menentukan bahasa (`language_code`).

aws_comprehend_detect_sentiment

Fungsi ini mengidentifikasi teks seolah-olah memiliki postur emosional positif, negatif, netral, atau campuran. Dokumentasi referensi fungsi ini adalah sebagai berikut.

```
aws_comprehend_detect_sentiment(
  input_text,
  language_code
  [,max_batch_size]
)
```

Untuk mempelajari selengkapnya, lihat Sentimen di Panduan Developer Amazon Comprehend.

aws_comprehend_detect_sentiment_confidence

Fungsi ini mengukur tingkat kepercayaan sentimen yang terdeteksi untuk teks tertentu. Ia menampilkan nilai (type, `double`) yang menunjukkan kepercayaan sentimen yang ditetapkan oleh fungsi aws_comprehend_detect_sentiment ke teks. Keyakinan adalah metrik statistik antara 0 dan 1. Makin tinggi tingkat kepercayaan, makin berat hasil yang bisa Anda berikan. Ringkasan dokumentasi fungsinya adalah sebagai berikut.

```
aws_comprehend_detect_sentiment_confidence(
  input_text,
  language_code
```

```
  [,max_batch_size]
)
```

Di kedua fungsi (aws_comprehend_detect_sentiment_confidence, aws_comprehend_detect_sentiment), `max_batch_size` menggunakan nilai default 25 jika tidak ada yang ditentukan. Ukuran Batch harus selalu lebih besar dari 0. Anda dapat menggunakan `max_batch_size` untuk menyetel performa dari panggilan fungsi Amazon Comprehend. Ukuran batch yang besar mengorbankan performa yang lebih cepat demi penggunaan memori yang lebih besar pada klaster DB Aurora MySQL. Untuk informasi selengkapnya, lihat Pertimbangan performa untuk machine learning Aurora dengan Aurora MySQL.

Untuk informasi selengkapnya tentang parameter dan tipe pengembalian untuk fungsi deteksi sentimen di Amazon Comprehend, lihat DetectSentiment

Example Contoh: Kueri sederhana menggunakan fungsi Amazon Comprehend

Berikut adalah contoh kueri sederhana yang menginvokasi dua fungsi ini untuk melihat seberapa senang pelanggan Anda dengan tim dukungan Anda. Misalkan Anda memiliki tabel basis data (`support`) yang menyimpan umpan balik pelanggan setelah setiap permintaan bantuan. Contoh kueri ini menerapkan kedua fungsi default ke teks di kolom `feedback` dari tabel dan menampilkan hasilnya. Nilai kepercayaan yang ditampilkan oleh fungsi adalah ganda, antara 0,0 dan 1,0. Untuk output yang lebih mudah dibaca, kueri ini membulatkan hasil menjadi 6 poin desimal. Untuk perbandingan yang lebih mudah, kueri ini juga mengurutkan hasil dalam urutan menurun, dari hasil yang memiliki tingkat kepercayaan tertinggi, pertama.

```
SELECT feedback AS 'Customer feedback',
    aws_comprehend_detect_sentiment(feedback, 'en') AS Sentiment,
    ROUND(aws_comprehend_detect_sentiment_confidence(feedback, 'en'), 6)
    AS Confidence FROM support
    ORDER BY Confidence DESC;
+-----------------------------------------------------------+-----------+------------+
| Customer feedback                                         | Sentiment | Confidence |
+-----------------------------------------------------------+-----------+------------+
| Thank you for the excellent customer support!             | POSITIVE  |   0.999771 |
| The latest version of this product stinks!                | NEGATIVE  |   0.999184 |
| Your support team is just awesome! I am blown away.       | POSITIVE  |   0.997774 |
| Your product is too complex, but your support is great.   | MIXED     |   0.957958 |
| Your support tech helped me in fifteen minutes.           | POSITIVE  |   0.949491 |
| My problem was never resolved!                            | NEGATIVE  |   0.920644 |
| When will the new version of this product be released?    | NEUTRAL   |   0.902706 |
```

```
| I cannot stand that chatbot.                                | NEGATIVE  |   0.895219 |
| Your support tech talked down to me.                        | NEGATIVE  |   0.868598 |
| It took me way too long to get a real person.               | NEGATIVE  |   0.481805 |
+-------------------------------------------------------------+-----------+------------+
 10 rows in set (0.1898 sec)
```

Example Contoh: Menentukan sentimen rata-rata untuk teks di atas tingkat kepercayaan tertentu

Kueri Amazon Comprehend biasanya mencari baris di mana sentimennya adalah nilai tertentu, dengan tingkat kepercayaan yang lebih besar dari angka tertentu. Misalnya, kueri berikut ini memperlihatkan bagaimana Anda bisa menentukan sentimen rata-rata dokumen di basis data Anda. Kueri hanya mempertimbangkan dokumen dengan tingkat kepercayaan penilaian minimal 80%.

```
SELECT AVG(CASE aws_comprehend_detect_sentiment(productTable.document, 'en')
  WHEN 'POSITIVE' THEN 1.0
  WHEN 'NEGATIVE' THEN -1.0
  ELSE 0.0 END) AS avg_sentiment, COUNT(*) AS total
FROM productTable
WHERE productTable.productCode = 1302 AND
    aws_comprehend_detect_sentiment_confidence(productTable.document, 'en') >= 0.80;
```

Menggunakan SageMaker dengan cluster DB MySQL Aurora Anda

Untuk menggunakan SageMaker fungsionalitas dari cluster DB MySQL Aurora Anda, Anda perlu membuat fungsi tersimpan yang menyematkan panggilan Anda ke titik akhir dan fitur inferensinya. SageMaker Ini dapat dilakukan dengan menggunakan `CREATE FUNCTION` MySQL secara umum dengan cara yang sama seperti yang Anda lakukan untuk tugas pemrosesan lainnya di klaster DB Aurora MySQL Anda.

Untuk menggunakan model yang digunakan SageMaker untuk inferensi, Anda membuat fungsi yang ditentukan pengguna menggunakan pernyataan bahasa definisi data MySQL (DDL) untuk fungsi yang disimpan. Setiap fungsi yang disimpan mewakili SageMaker titik akhir yang menghosting model. Saat Anda mendefinisikan fungsi seperti itu, Anda menentukan parameter input ke model, SageMaker titik akhir spesifik yang akan dipanggil, dan jenis pengembalian. Fungsi mengembalikan inferensi yang dihitung oleh SageMaker titik akhir setelah menerapkan model ke parameter input.

Semua fungsi tersimpan machine learning Aurora menampilkan tipe numerik atau `VARCHAR`. Anda dapat menggunakan jenis numerik kecuali `BIT`. Jenis lainnya, seperti `JSON`, `BLOB`, `TEXT`, dan `DATE` tidak diizinkan.

Contoh berikut menunjukkan `CREATE FUNCTION` sintaks untuk bekerja dengan SageMaker.

```
CREATE FUNCTION function_name (
     arg1 type1,
     arg2 type2, ...)
     [DEFINER = user]
  RETURNS mysql_type
  [SQL SECURITY { DEFINER | INVOKER } ]
  ALIAS AWS_SAGEMAKER_INVOKE_ENDPOINT
  ENDPOINT NAME 'endpoint_name'
  [MAX_BATCH_SIZE max_batch_size];
```

Ini adalah perpanjangan dari pernyataan DDL `CREATE FUNCTION` reguler. Dalam `CREATE FUNCTION` pernyataan yang mendefinisikan SageMaker fungsi, Anda tidak menentukan badan fungsi. Alih-alih, Anda menentukan kata kunci `ALIAS` yang biasanya digunakan oleh badan fungsi. Saat ini, machine learning Aurora hanya mendukung `aws_sagemaker_invoke_endpoint` untuk sintaksis yang diperluas ini. Anda harus menentukan parameter `endpoint_name`. SageMaker Titik akhir dapat memiliki karakteristik yang berbeda untuk setiap model.

> **Note**
>
> Untuk informasi selengkapnya tentang `CREATE FUNCTION`, lihat CREATE PROCEDURE and CREATE FUNCTION Statements di Panduan Referensi 8.0 MySQL.

Parameter `max_batch_size` bersifat opsional. Secara default, ukuran batch maksimum adalah 10.000. Anda dapat menggunakan parameter ini dalam fungsi Anda untuk membatasi jumlah maksimum input yang diproses dalam permintaan batch. SageMaker `max_batch_size`Parameter dapat membantu menghindari kesalahan yang disebabkan oleh input yang terlalu besar, atau untuk membuat SageMaker pengembalian respons lebih cepat. Parameter ini mempengaruhi ukuran buffer internal yang digunakan untuk pemrosesan SageMaker permintaan. Menentukan nilai yang terlalu besar untuk `max_batch_size` dapat menyebabkan overhead memori yang besar pada instans DB Anda.

Sebaiknya biarkan pengaturan `MANIFEST` pada nilai default `OFF`. Meskipun Anda dapat menggunakan `MANIFEST ON` opsi ini, beberapa SageMaker fitur tidak dapat langsung menggunakan CSV yang diekspor dengan opsi ini. Format manifes tidak kompatibel dengan format manifes yang diharapkan dari SageMaker.

Anda membuat fungsi tersimpan terpisah untuk masing-masing SageMaker model Anda. Pemetaan fungsi ke model ini diperlukan karena titik akhir dikaitkan dengan model tertentu, dan setiap model

menerima parameter yang berbeda. Menggunakan tipe SQL untuk input model dan tipe keluaran model membantu menghindari kesalahan konversi tipe yang meneruskan data bolak-balik antara layanan. AWS Anda dapat mengontrol siapa yang dapat menerapkan model tersebut. Anda juga dapat mengontrol karakteristik runtime dengan menentukan parameter yang mewakili ukuran batch maksimum.

Saat ini, semua fungsi machine learning Aurora memiliki properti `NOT DETERMINISTIC`. Jika Anda tidak menentukan properti tersebut secara eksplisit, Aurora menyetel `NOT DETERMINISTIC` secara otomatis. Persyaratan ini karena SageMaker model dapat diubah tanpa pemberitahuan apa pun ke database. Jika itu terjadi, panggilan ke fungsi machine learning Aurora mungkin menampilkan hasil yang berbeda untuk input yang sama dalam satu transaksi.

Anda tidak dapat menggunakan karakteristik `CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, atau `MODIFIES SQL DATA` dalam pernyataan `CREATE FUNCTION` Anda.

Berikut ini adalah contoh penggunaan menggunakan SageMaker titik akhir untuk mendeteksi anomali. Ada SageMaker titik akhir`random-cut-forest-model`. Model yang sesuai sudah dilatih oleh algoritma `random-cut-forest`. Untuk setiap input, model menampilkan skor anomali. Contoh ini menunjukkan poin data yang nilainya lebih besar dari 3 deviasi standar (kira-kira persentil ke-99,9) dari skor rata-rata.

```
CREATE FUNCTION anomaly_score(value real) returns real
  alias aws_sagemaker_invoke_endpoint endpoint name 'random-cut-forest-model-demo';

set @score_cutoff = (select avg(anomaly_score(value)) + 3 * std(anomaly_score(value))
 from nyc_taxi);

select *, anomaly_detection(value) score from nyc_taxi
  where anomaly_detection(value) > @score_cutoff;
```

Persyaratan set karakter untuk SageMaker fungsi yang mengembalikan string

Sebaiknya tentukan set karakter `utf8mb4` sebagai tipe pengembalian untuk SageMaker fungsi Anda yang mengembalikan nilai string. Jika itu tidak praktis, gunakan panjang string yang cukup besar agar jenis kembalian dapat menampung nilai yang direpresentasikan dalam kumpulan karakter `utf8mb4`. Contoh berikut menunjukkan cara mendeklarasikan set karakter `utf8mb4` untuk fungsi Anda.

```
CREATE FUNCTION my_ml_func(...) RETURNS VARCHAR(5) CHARSET utf8mb4 ALIAS ...
```

Saat ini, setiap SageMaker fungsi yang mengembalikan string menggunakan set karakter `utf8mb4` untuk nilai kembali. Nilai yang dikembalikan menggunakan set karakter ini bahkan jika SageMaker fungsi Anda mendeklarasikan set karakter yang berbeda untuk tipe pengembaliannya secara implisit atau eksplisit. Jika SageMaker fungsi Anda mendeklarasikan set karakter yang berbeda untuk nilai yang dikembalikan, data yang dikembalikan mungkin terpotong secara diam-diam jika Anda menyimpannya di kolom tabel yang tidak cukup panjang. Misalnya, kueri dengan klausa `DISTINCT` membuat tabel sementara. Dengan demikian, hasil SageMaker fungsi mungkin terpotong karena cara string ditangani secara internal selama kueri.

Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model (Lanjutan)

Kami menyarankan Anda memulai pembelajaran mesin Aurora dan SageMaker dengan menggunakan beberapa algoritme yang disediakan, dan bahwa ilmuwan data di tim Anda memberi Anda SageMaker titik akhir yang dapat Anda gunakan dengan kode SQL Anda. Berikut ini, Anda dapat menemukan informasi minimal tentang penggunaan bucket Amazon S3 Anda sendiri dengan SageMaker model Anda sendiri dan cluster Aurora MySQL DB Anda.

Machine learning terdiri dari dua langkah utama: pelatihan, dan inferensi. Untuk melatih SageMaker model, Anda mengekspor data ke bucket Amazon S3. Bucket Amazon S3 digunakan oleh instance SageMaker notebook Jupyter untuk melatih model Anda sebelum digunakan. Anda dapat menggunakan pernyataan `SELECT INTO OUTFILE S3` untuk mengueri data dari klaster DB Aurora MySQL dan menyimpannya langsung ke dalam file teks yang tersimpan di bucket Amazon S3. Kemudian instans notebook menggunakan data dari bucket Amazon S3 untuk pelatihan.

Machine learning Aurora memperluas sintaksis `SELECT INTO OUTFILE` yang ada di Aurora MySQL untuk mengekspor data ke format CSV. File CSV yang dihasilkan dapat digunakan secara langsung oleh model yang memerlukan format ini untuk tujuan pelatihan.

```
SELECT * INTO OUTFILE S3 's3_uri' [FORMAT {CSV|TEXT} [HEADER]] FROM table_name;
```

Ekstensi mendukung format CSV standar.

- Format `TEXT` sama dengan format ekspor MySQL yang ada. Ini adalah format default.
- Format `CSV` adalah format yang baru diperkenalkan yang mengikuti spesifikasi di RFC-4180.
- Jika Anda menentukan kata kunci opsional `HEADER`, file output akan berisi satu baris header. Label di baris header sesuai dengan nama kolom dari pernyataan `SELECT`.
- Anda masih dapat menggunakan kata kunci `CSV` dan `HEADER` sebagai pengenal.

Sintaks dan tata bahasa `SELECT INTO` yang diperluas sekarang adalah sebagai berikut:

```
INTO OUTFILE S3 's3_uri'
[CHARACTER SET charset_name]
[FORMAT {CSV|TEXT} [HEADER]]
[{FIELDS | COLUMNS}
  [TERMINATED BY 'string']
  [[OPTIONALLY] ENCLOSED BY 'char']
  [ESCAPED BY 'char']
]
[LINES
  [STARTING BY 'string']
  [TERMINATED BY 'string']
]
```

Pertimbangan performa untuk machine learning Aurora dengan Aurora MySQL

Amazon Bedrock, Amazon Comprehend SageMaker , dan layanan melakukan sebagian besar pekerjaan saat dipanggil oleh fungsi pembelajaran mesin Aurora. Itu berarti Anda dapat menskalakan sumber daya tersebut sesuai kebutuhan, secara mandiri. Untuk klaster DB Aurora MySQL Anda, Anda dapat membuat panggilan fungsi seefisien mungkin. Berikut ini, Anda dapat menemukan beberapa pertimbangan performa yang perlu diperhatikan saat bekerja dengan machine learning Aurora.

Model dan prompt

Performa saat menggunakan Amazon Bedrock sangat bergantung pada model dan prompt yang Anda gunakan. Pilih model dan prompt yang optimal untuk kasus penggunaan Anda.

Cache kueri

Cache kueri Aurora MySQL tidak berfungsi untuk fungsi machine learning Aurora. Aurora MySQL tidak menyimpan hasil kueri di cache kueri untuk pernyataan SQL apa pun yang memanggil fungsi machine learning Aurora.

Optimalisasi batch untuk panggilan fungsi machine learning Aurora

Aspek performa utama machine learning Aurora yang dapat Anda pengaruhi dari klaster Aurora adalah pengaturan mode batch untuk panggilan ke fungsi tersimpan machine learning Aurora.

Fungsi machine learning biasanya membutuhkan overhead yang besar, sehingga tidak praktis untuk memanggil layanan eksternal secara terpisah untuk setiap baris. Machine learning Aurora dapat meminimalkan overhead ini dengan menggabungkan panggilan ke layanan machine learning Aurora eksternal untuk banyak baris ke dalam satu batch. Machine learning Aurora menerima respons untuk semua baris input, dan mengirimkan respons, satu baris dalam satu waktu, ke kueri saat dijalankan. Pengoptimalan ini meningkatkan throughput dan latensi kueri Aurora Anda tanpa mengubah hasil.

Saat Anda membuat fungsi tersimpan Aurora yang terhubung ke SageMaker titik akhir, Anda menentukan parameter ukuran batch. Parameter ini memengaruhi berapa banyak baris yang ditransfer untuk setiap panggilan yang mendasarinya SageMaker. Untuk kueri yang memproses sejumlah besar baris, overhead untuk membuat SageMaker panggilan terpisah untuk setiap baris bisa sangat besar. Semakin besar kumpulan data yang diproses oleh prosedur tersimpan, semakin besar Anda dapat membuat ukuran batch.

Jika optimasi mode batch dapat diterapkan ke suatu SageMaker fungsi, Anda dapat mengetahuinya dengan memeriksa rencana kueri yang dihasilkan oleh `EXPLAIN PLAN` pernyataan tersebut. Dalam kasus ini, kolom `extra` dalam rencana eksekusi termasuk `Batched machine learning`. Contoh berikut menunjukkan panggilan ke SageMaker fungsi yang menggunakan mode batch.

```
mysql> CREATE FUNCTION anomaly_score(val real) returns real alias
 aws_sagemaker_invoke_endpoint endpoint name 'my-rcf-model-20191126';
Query OK, 0 rows affected (0.01 sec)

mysql> explain select timestamp, value, anomaly_score(value) from nyc_taxi;
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+-------------------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len |
 ref  | rows | filtered | Extra                   |
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+-------------------------+
|  1 | SIMPLE      | nyc_taxi | NULL       | ALL  | NULL          | NULL | NULL    |
 NULL |   48 |   100.00 | Batched machine learning |
+----+-------------+----------+------------+------+---------------+------+---------
+------+------+----------+-------------------------+
1 row in set, 1 warning (0.01 sec)
```

Saat Anda memanggil salah satu fungsi Amazon Comprehend default, Anda dapat mengontrol ukuran batch dengan menentukan parameter `max_batch_size` opsional. Parameter ini membatasi jumlah maksimum nilai `input_text` yang diproses di setiap batch. Dengan mengirim beberapa item sekaligus, ini mengurangi jumlah perjalanan bolak-balik antara Aurora dan Amazon Comprehend.

Membatasi ukuran batch berguna dalam situasi seperti kueri dengan klausa `LIMIT`. Dengan menggunakan nilai kecil untuk `max_batch_size`, Anda dapat menghindari permintaan Amazon Comprehend lebih sering daripada Anda memiliki teks input.

Optimalisasi batch untuk mengevaluasi fungsi machine learning Aurora berlaku dalam kasus berikut:

- Fungsi panggilan dalam daftar pilih atau `WHERE` klausa pernyataan `SELECT`
- Panggilan fungsi dalam `VALUES` daftar `INSERT` dan `REPLACE` pernyataan
- SageMaker fungsi dalam `SET` nilai dalam `UPDATE` pernyataan:

```
INSERT INTO MY_TABLE (col1, col2, col3) VALUES
  (ML_FUNC(1), ML_FUNC(2), ML_FUNC(3)),
  (ML_FUNC(4), ML_FUNC(5), ML_FUNC(6));
UPDATE MY_TABLE SET col1 = ML_FUNC(col2), SET col3 = ML_FUNC(col4) WHERE ...;
```

Memantau machine learning Aurora

Anda dapat memantau operasi batch pembelajaran mesin Aurora dengan menanyakan beberapa variabel global, seperti yang ditunjukkan pada contoh berikut.

```
show status like 'Aurora_ml%';
```

Anda dapat mengatur ulang variabel status dengan menggunakan pernyataan `FLUSH STATUS`. Jadi, semua angka mewakili total, rata-rata, dan seterusnya, sejak terakhir kali variabel disetel ulang.

`Aurora_ml_logical_request_cnt`

Jumlah permintaan logis yang telah dievaluasi instans DB untuk dikirim ke layanan machine learning Aurora sejak status pengaturan ulang status terakhir. Bergantung pada apakah proses batching telah digunakan, nilai ini dapat lebih tinggi dari `Aurora_ml_actual_request_cnt`.

`Aurora_ml_logical_response_cnt`

Jumlah respons gabungan yang diterima Aurora MySQL dari layanan machine learning Aurora di semua kueri yang dijalankan oleh pengguna instans DB.

`Aurora_ml_actual_request_cnt`

Jumlah permintaan gabungan yang dibuat Aurora MySQL ke layanan machine learning Aurora di semua kueri yang dijalankan oleh pengguna instans DB.

`Aurora_ml_actual_response_cnt`

Jumlah respons gabungan yang diterima Aurora MySQL dari layanan machine learning Aurora di semua kueri yang dijalankan oleh pengguna instans DB.

`Aurora_ml_cache_hit_cnt`

Jumlah klik cache internal gabungan yang diterima Aurora MySQL dari layanan machine learning Aurora di semua kueri yang dijalankan oleh pengguna instans DB.

`Aurora_ml_retry_request_cnt`

Jumlah permintaan yang dicoba ulang yang dikirim instans DB ke layanan machine learning Aurora sejak pengaturan ulang status terakhir.

`Aurora_ml_single_request_cnt`

Jumlah gabungan fungsi machine learning Aurora yang dievaluasi oleh mode non-batch di semua kueri yang dijalankan oleh pengguna instans DB.

Untuk informasi tentang pemantauan kinerja SageMaker operasi yang disebut dari fungsi pembelajaran mesin Aurora, lihat Memantau Amazon. SageMaker

Menggunakan machine learning Amazon Aurora dengan Aurora PostgreSQL

Dengan menggunakan pembelajaran mesin Amazon Aurora dengan cluster DB Aurora PostgreSQL Anda, Anda dapat menggunakan Amazon Comprehend atau Amazon atau Amazon Bedrock, tergantung pada kebutuhan Anda. SageMaker Layanan ini masing-masing mendukung kasus penggunaan pembelajaran mesin tertentu.

Pembelajaran mesin Aurora didukung dalam versi tertentu Wilayah AWS dan khusus dari Aurora PostgreSQL saja. Sebelum mencoba menyiapkan machine learning Aurora, periksa ketersediaan untuk versi Aurora PostgreSQL dan Wilayah Anda. Untuk detailnya, lihat Machine learning Aurora dengan Aurora PostgreSQL.

Topik

- Persyaratan untuk menggunakan machine learning Aurora dengan Aurora PostgreSQL
- Fitur yang didukung dan batasan machine learning Aurora dengan Aurora PostgreSQL
- Menyiapkan klaster DB Aurora PostgreSQL untuk menggunakan machine learning Aurora

- Menggunakan Amazon Bedrock dengan cluster Aurora PostgreSQL DB Anda
- Menggunakan Amazon Comprehend dengan klaster DB Aurora PostgreSQL
- Menggunakan SageMaker dengan cluster DB PostgreSQL Aurora Anda
- Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model (Lanjutan)
- Pertimbangan performa untuk machine learning Aurora dengan Aurora PostgreSQL
- Memantau machine learning Aurora

Persyaratan untuk menggunakan machine learning Aurora dengan Aurora PostgreSQL

AWS Layanan pembelajaran mesin adalah layanan terkelola yang diatur dan dijalankan di lingkungan produksi mereka sendiri. Pembelajaran mesin Aurora mendukung integrasi dengan Amazon Comprehend,, dan Amazon Bedrock SageMaker. Sebelum mencoba menyiapkan klaster DB Aurora PostgreSQL untuk menggunakan machine learning Aurora, pastikan Anda memahami persyaratan dan prasyarat berikut.

- Layanan Amazon SageMaker Comprehend,, dan Amazon Bedrock harus berjalan Wilayah AWS sama dengan cluster DB PostgreSQL Aurora Anda. Anda tidak dapat menggunakan layanan Amazon SageMaker Comprehend atau Amazon Bedrock dari klaster DB PostgreSQL Aurora di Wilayah yang berbeda.
- Jika klaster DB PostgreSQL Aurora Anda berada di cloud publik virtual (VPC) yang berbeda berdasarkan layanan Amazon VPC daripada Amazon Comprehend dan layanan Amazon Comprehend Anda, grup Keamanan VPC perlu mengizinkan koneksi keluar SageMaker ke layanan pembelajaran mesin Aurora target. Untuk informasi selengkapnya, lihat Mengaktifkan komunikasi jaringan dari Amazon Aurora MySQL ke layanan AWS lainnya.
- Untuk SageMaker, komponen pembelajaran mesin yang ingin Anda gunakan untuk kesimpulan harus diatur dan siap digunakan. Selama proses konfigurasi untuk klaster DB PostgreSQL Aurora, Anda harus memiliki Nama Sumber Daya Amazon (ARN) dari titik akhir yang tersedia. SageMaker Para ilmuwan data di tim Anda mungkin paling mampu menangani bekerja dengan SageMaker untuk mempersiapkan model dan menangani tugas-tugas lain semacam itu. Untuk memulai dengan Amazon SageMaker, lihat Memulai dengan Amazon SageMaker. Untuk informasi selengkapnya tentang inferensi dan titik akhir, lihat Inferensi real-time.
- Untuk Amazon Bedrock, Anda harus memiliki ID model model Bedrock yang ingin Anda gunakan untuk inferensi yang tersedia selama proses konfigurasi cluster DB PostgreSQL Aurora Anda.

Ilmuwan data di tim Anda kemungkinan besar paling mampu bekerja dengan Bedrock untuk memutuskan model mana yang akan digunakan, menyempurnakannya jika diperlukan, dan menangani tugas-tugas semacam itu lainnya. Untuk memulai dengan Amazon Bedrock, lihat Cara mengatur Bedrock.

- Pengguna Amazon Bedrock perlu meminta akses ke model sebelum tersedia untuk digunakan. Jika Anda ingin menambahkan model tambahan untuk pembuatan teks, obrolan, dan gambar, Anda perlu meminta akses ke model di Amazon Bedrock. Untuk informasi selengkapnya, lihat Akses model.

Fitur yang didukung dan batasan machine learning Aurora dengan Aurora PostgreSQL

Pembelajaran mesin Aurora mendukung setiap SageMaker titik akhir yang dapat membaca dan menulis format nilai dipisahkan koma (CSV) melalui nilai. `ContentType` `text/csv` SageMaker Algoritma bawaan yang saat ini menerima format ini adalah sebagai berikut.

- Linear Learner
- Random Cut Forest
- XGBoost

  Untuk mempelajari lebih lanjut tentang algoritme ini, lihat Memilih Algoritma di Panduan SageMaker Pengembang Amazon.

Saat menggunakan Amazon Bedrock dengan pembelajaran mesin Aurora, batasan berikut berlaku:

- Fungsi yang ditentukan pengguna (UDF) menyediakan cara asli untuk berinteraksi dengan Amazon Bedrock. UDF tidak memiliki permintaan atau persyaratan respons khusus, sehingga mereka dapat menggunakan model apa pun.
- Anda dapat menggunakan UDF untuk membangun alur kerja apa pun yang diinginkan. Misalnya, Anda dapat menggabungkan primitif dasar seperti `pg_cron` menjalankan kueri, mengambil data, menghasilkan kesimpulan, dan menulis ke tabel untuk menyajikan kueri secara langsung.
- UDF tidak mendukung panggilan batch atau parallel.
- Ekstensi Machine Learning Aurora tidak mendukung antarmuka vektor. Sebagai bagian dari ekstensi, fungsi tersedia untuk menampilkan penyematan respons model dalam `float8[]` format untuk menyimpan embeddings tersebut di Aurora. Untuk informasi lebih lanjut tentang

penggunaan`float8[]`, lihat[Menggunakan Amazon Bedrock dengan cluster Aurora PostgreSQL DB Anda](#).

Menyiapkan klaster DB Aurora PostgreSQL untuk menggunakan machine learning Aurora

Agar pembelajaran mesin Aurora dapat bekerja dengan cluster DB PostgreSQL Aurora Anda, Anda perlu membuat peran AWS Identity and Access Management (IAM) untuk setiap layanan yang ingin Anda gunakan. Peran IAM memungkinkan klaster DB Aurora PostgreSQL Anda menggunakan layanan machine learning Aurora atas nama klaster. Anda juga perlu menginstal ekstensi machine learning Aurora. Dalam topik berikut, Anda dapat menemukan prosedur penyiapan untuk masing-masing layanan machine learning Aurora ini.

Topik

- Menyiapkan Aurora PostgreSQL untuk menggunakan Amazon Bedrock
- Menyiapkan Aurora PostgreSQL untuk menggunakan Amazon Comprehend
- Menyiapkan Aurora PostgreSQL untuk menggunakan Amazon SageMaker
  - Menyiapkan Aurora PostgreSQL untuk menggunakan Amazon S3 untuk (Lanjutan) SageMaker
- Menginstal ekstensi machine learning Aurora

Menyiapkan Aurora PostgreSQL untuk menggunakan Amazon Bedrock

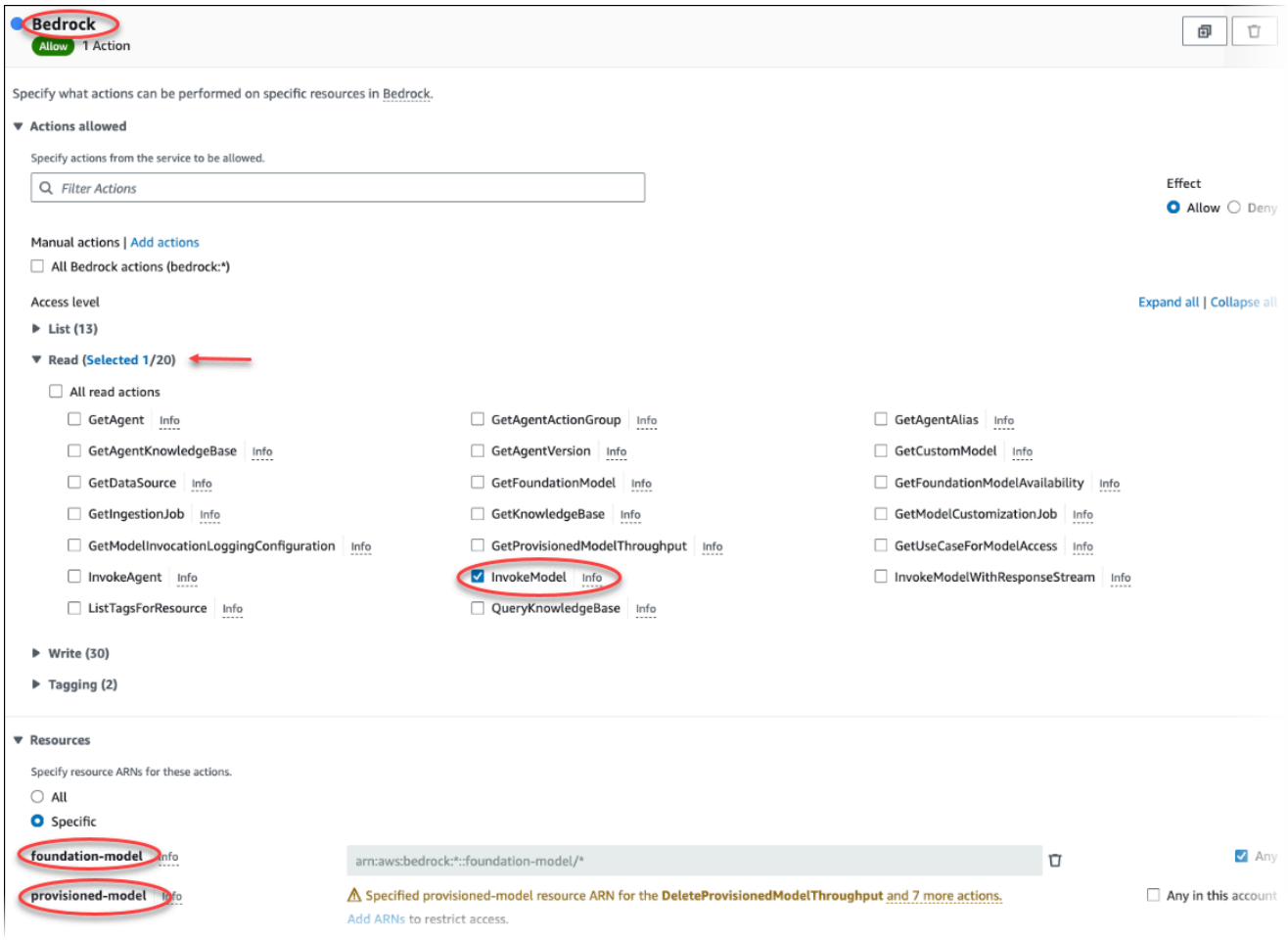
Dalam prosedur berikut, pertama-tama Anda membuat peran dan kebijakan IAM yang memberikan izin Aurora PostgreSQL Anda untuk menggunakan Amazon Bedrock atas nama klaster. Anda kemudian melampirkan kebijakan ke peran IAM yang digunakan klaster Aurora PostgreSQL DB Anda untuk bekerja dengan Amazon Bedrock. Demi kesederhanaan, prosedur ini menggunakan AWS Management Console untuk menyelesaikan semua tugas.

Untuk mengatur cluster DB PostgreSQL Aurora Anda untuk menggunakan Amazon Bedrock

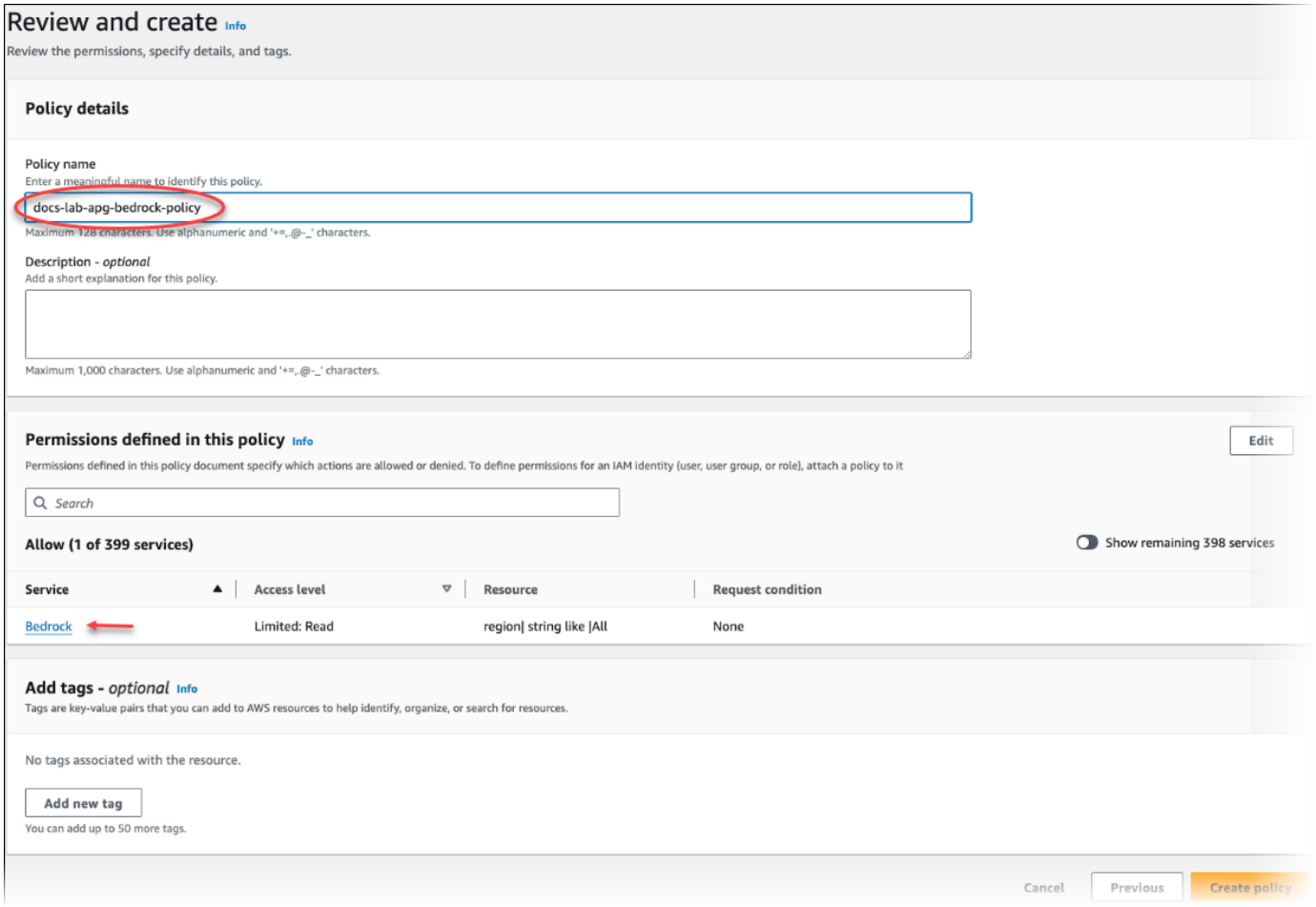
1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Buka konsol IAM di https://console.aws.amazon.com/iam/.
3. Pilih Kebijakan (di bawah Manajemen akses) pada menu Konsol AWS Identity and Access Management (IAM).

a. Pilih Buat kebijakan. Di halaman Editor Visual, pilih Layanan dan kemudian masukkan Bedrock di bidang Pilih layanan. Perluas tingkat akses Baca. Pilih InvokeModeldari pengaturan baca Amazon Bedrock.

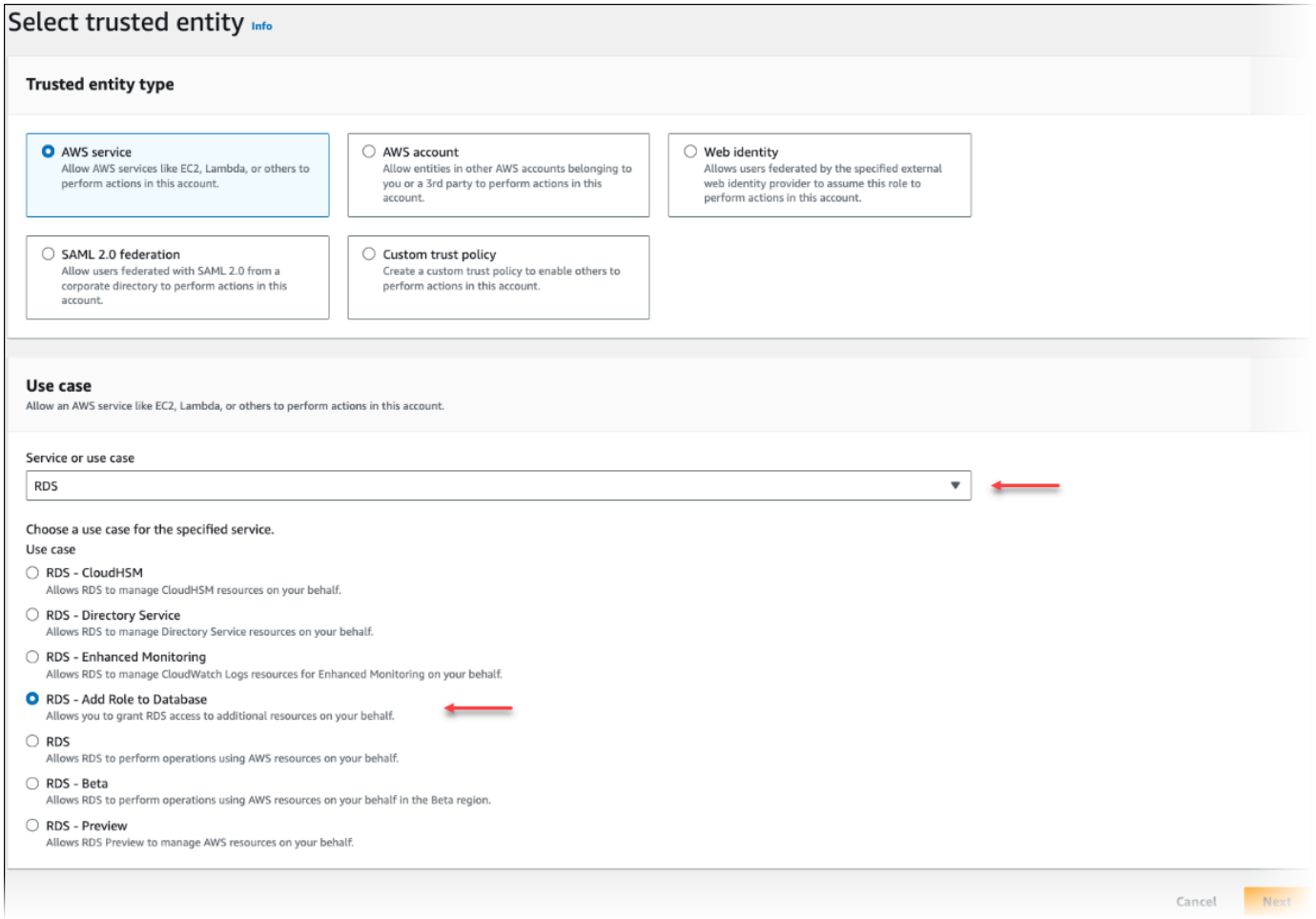
b. Pilih model Foundation/Provisioned yang ingin Anda berikan akses baca melalui kebijakan.



4. Pilih Berikutnya: Tag dan tentukan tag apa pun (opsional). Pilih Berikutnya: Peninjauan. Masukkan nama untuk kebijakan dan deskripsi, seperti diperlihatkan pada gambar.

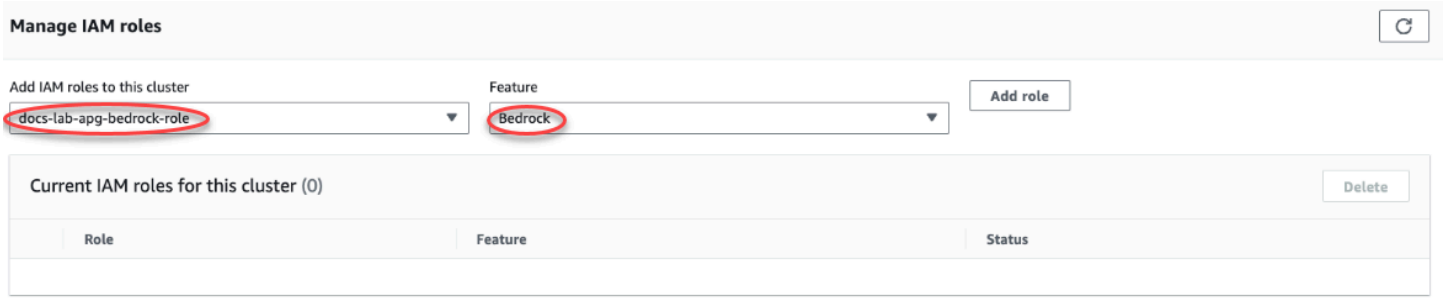


5. Pilih Buat kebijakan. Konsol menampilkan peringatan saat kebijakan telah disimpan. Anda dapat menemukannya di daftar Kebijakan.
6. Pilih Peran (di bagian Manajemen akses) di Konsol IAM.
7. Pilih Buat peran.
8. Di halaman Pilih entitas tepercaya, pilih kotak layanan AWS , lalu pilih RDS untuk membuka pemilih.
9. Pilih RDS – Tambahkan Peran ke Basis Data.



10. Pilih Selanjutnya. Di halaman Tambahkan izin, temukan kebijakan yang Anda buat di langkah sebelumnya dan pilih kebijakan dari daftar tersebut. Pilih Berikutnya.
11. Berikutnya: Tinjau. Masukkan nama peran IAM dan deskripsinya.
12. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
13. Arahkan ke Wilayah AWS tempat cluster Aurora PostgreSQL DB Anda berada.
14. Di panel navigasi, pilih Databases, dan kemudian pilih cluster Aurora PostgreSQL DB yang ingin Anda gunakan dengan Bedrock.
15. Pilih tab Konektivitas & keamanan dan gulir untuk menemukan bagian Kelola peran IAM pada halaman. Dari pemilih Tambahkan peran IAM ke klaster ini, pilih peran yang Anda buat di langkah sebelumnya. Di pemilih Fitur, pilih Batuan Dasar, lalu pilih Tambah peran.

Peran (dengan kebijakannya) dikaitkan dengan klaster DB Aurora PostgreSQL. Setelah proses selesai, peran tersebut dicantumkan dalam peran IAM saat ini untuk daftar klaster ini, seperti yang ditunjukkan berikut.



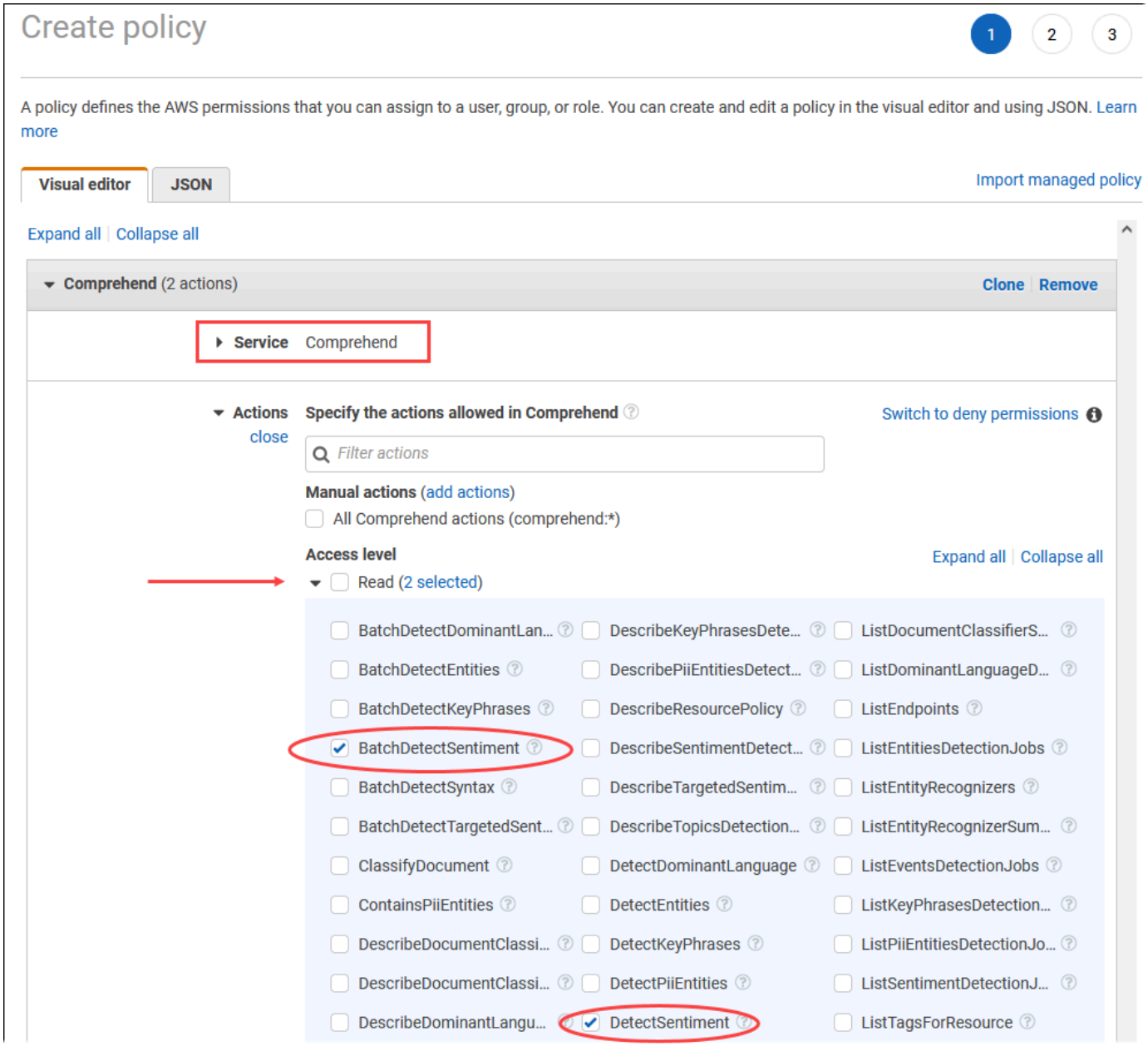
Pengaturan IAM untuk Amazon Bedrock selesai. Lanjutkan penyiapan Aurora PostgreSQL Anda untuk bekerja dengan machine learning Aurora dengan menginstal ekstensi seperti yang dijelaskan dalam Menginstal ekstensi machine learning Aurora

Menyiapkan Aurora PostgreSQL untuk menggunakan Amazon Comprehend

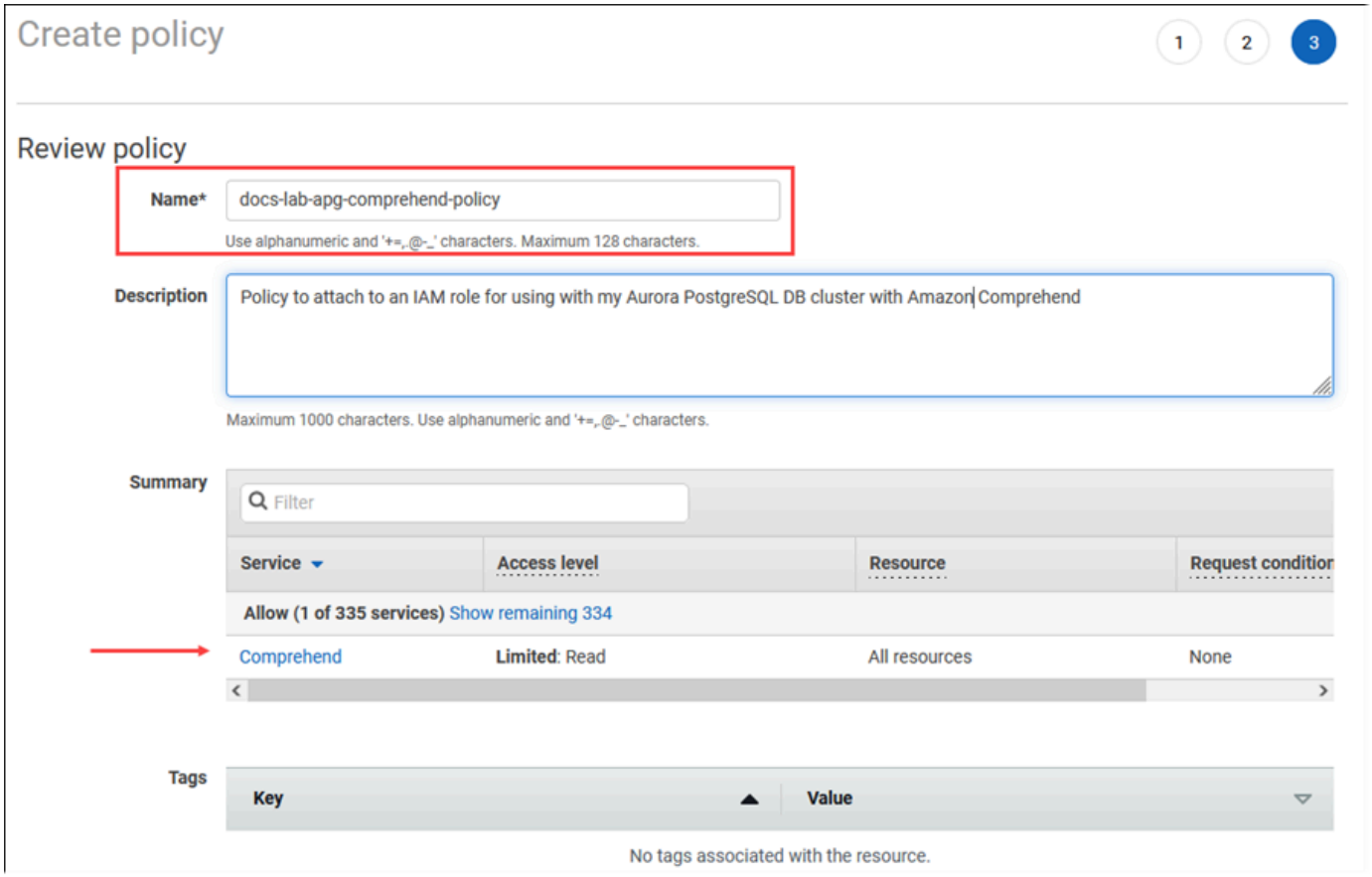
Dalam prosedur berikut, pertama-tama Anda membuat kebijakan dan peran IAM yang memberikan izin kepada Aurora PostgreSQL untuk menggunakan Amazon Comprehend atas nama klaster. Anda kemudian melampirkan kebijakan ke peran IAM yang digunakan klaster DB Aurora PostgreSQL untuk bekerja dengan Amazon Comprehend. Untuk memudahkan, prosedur ini menggunakan AWS Management Console untuk menyelesaikan semua tugas.

Untuk menyiapkan klaster DB Aurora PostgreSQL untuk menggunakan Amazon Comprehend

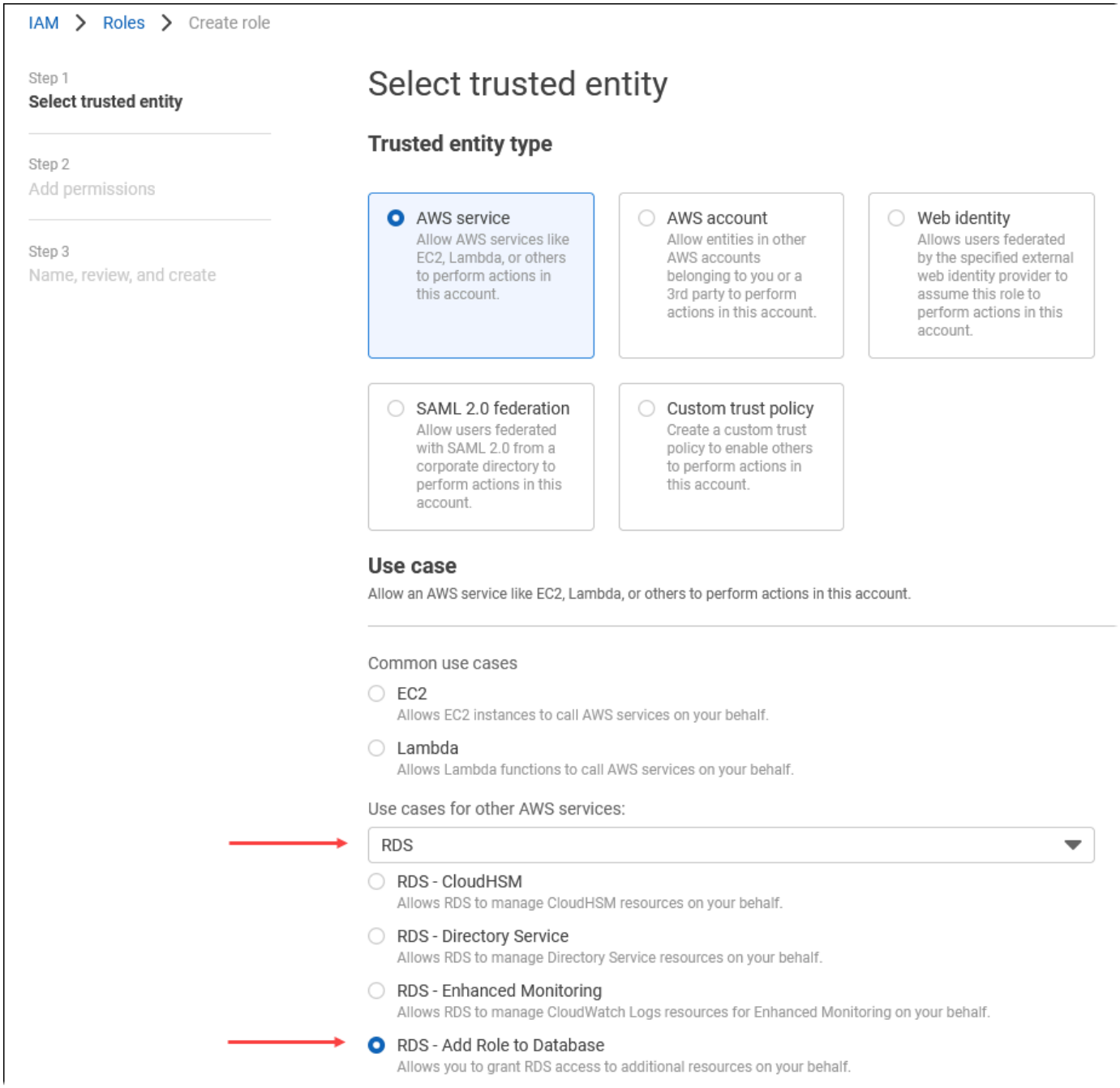
1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Buka konsol IAM di https://console.aws.amazon.com/iam/.
3. Pilih Kebijakan (di bawah Manajemen akses) pada menu Konsol AWS Identity and Access Management (IAM).



4. Pilih Buat kebijakan. Di halaman Editor Visual, pilih Layanan, lalu masukkan Comprehend di kolom Pilih layanan. Perluas tingkat akses Baca. Pilih BatchDetectSentimentdan DetectSentimentdari pengaturan baca Amazon Comprehend.
5. Pilih Berikutnya: Tag dan tentukan tag apa pun (opsional). Pilih Berikutnya: Peninjauan. Masukkan nama untuk kebijakan dan deskripsi, seperti diperlihatkan pada gambar.



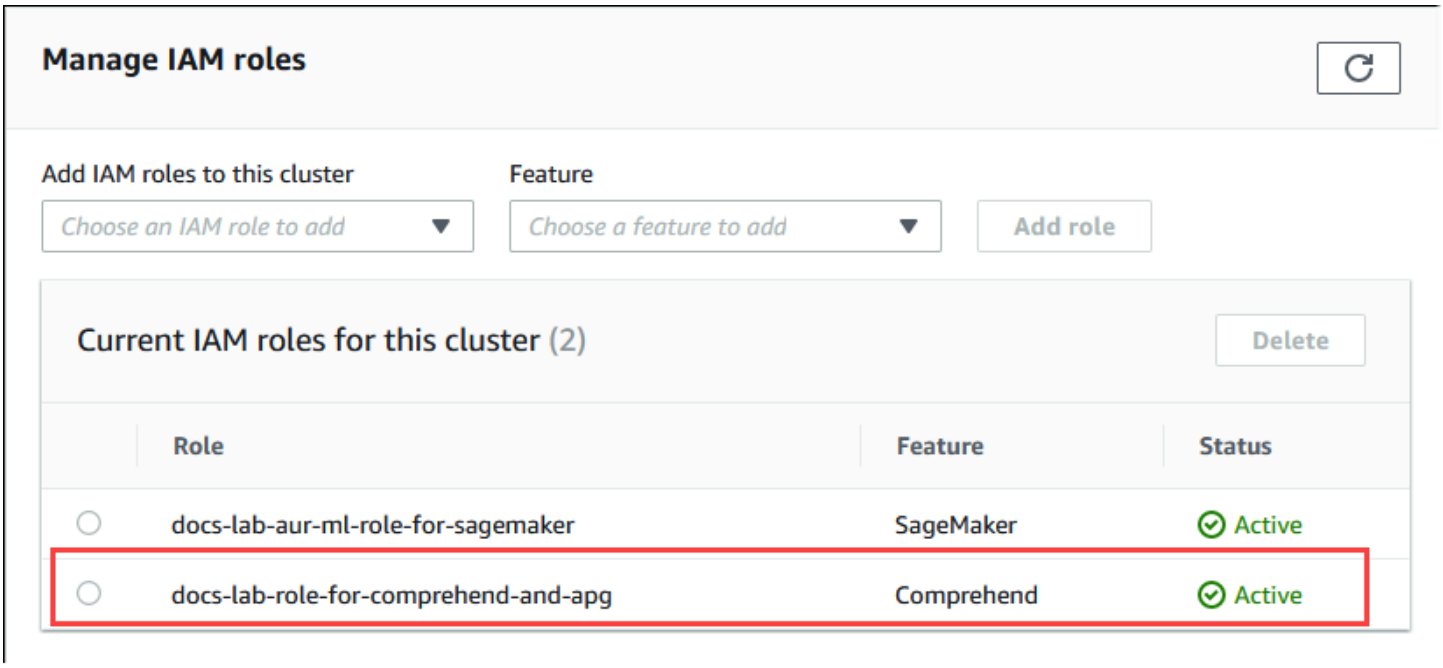
6. Pilih Buat kebijakan. Konsol menampilkan peringatan saat kebijakan telah disimpan. Anda dapat menemukannya di daftar Kebijakan.
7. Pilih Peran (di bagian Manajemen akses) di Konsol IAM.
8. Pilih Buat peran.
9. Di halaman Pilih entitas tepercaya, pilih kotak layanan AWS , lalu pilih RDS untuk membuka pemilih.
10. Pilih RDS – Tambahkan Peran ke Basis Data.



11. Pilih Selanjutnya. Di halaman Tambahkan izin, temukan kebijakan yang Anda buat di langkah sebelumnya dan pilih kebijakan dari daftar tersebut. Pilih Selanjutnya
12. Berikutnya: Tinjau. Masukkan nama peran IAM dan deskripsinya.
13. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
14. Arahkan ke Wilayah AWS tempat cluster Aurora PostgreSQL DB Anda berada.
15. Di panel navigasi, pilih Basis data, lalu pilih klaster DB Aurora PostgreSQL yang ingin digunakan dengan Amazon Comprehend.

16. Pilih tab Konektivitas & keamanan dan gulir untuk menemukan bagian Kelola peran IAM pada halaman. Dari pemilih Tambahkan peran IAM ke klaster ini, pilih peran yang Anda buat di langkah sebelumnya. Di pemilih Fitur, pilih Comprehend, lalu pilih Tambah peran.

Peran (dengan kebijakannya) dikaitkan dengan klaster DB Aurora PostgreSQL. Setelah proses selesai, peran tersebut dicantumkan dalam peran IAM saat ini untuk daftar klaster ini, seperti yang ditunjukkan berikut.



Penyiapan IAM untuk Amazon Comprehend selesai. Lanjutkan penyiapan Aurora PostgreSQL Anda untuk bekerja dengan machine learning Aurora dengan menginstal ekstensi seperti yang dijelaskan dalam Menginstal ekstensi machine learning Aurora

Menyiapkan Aurora PostgreSQL untuk menggunakan Amazon SageMaker

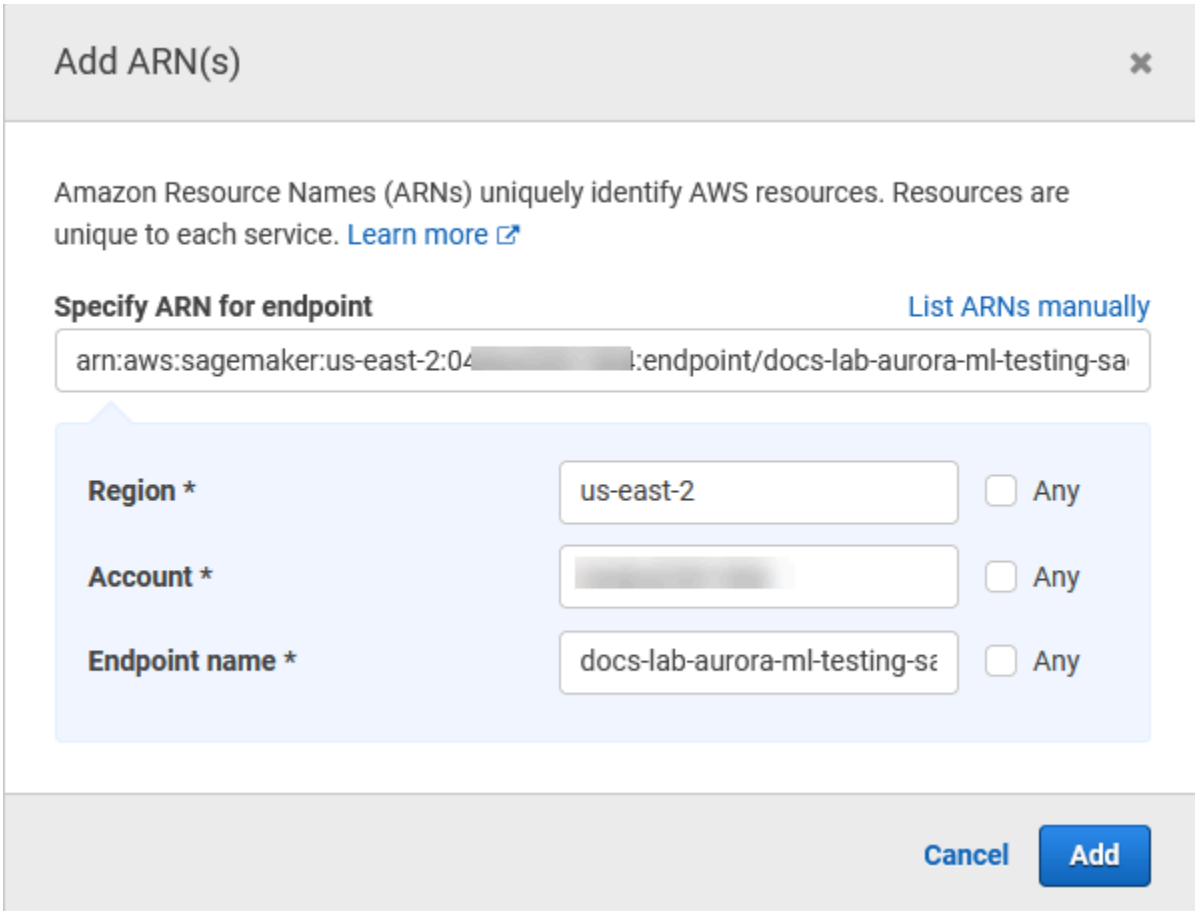
Sebelum Anda dapat membuat kebijakan dan peran IAM untuk klaster Aurora PostgreSQL DB Anda, Anda harus memiliki pengaturan model dan titik akhir yang tersedia. SageMaker

Untuk mengatur cluster DB PostgreSQL Aurora Anda untuk digunakan SageMaker

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Pilih Kebijakan (di bawah Manajemen akses) pada menu Konsol AWS Identity and Access Management (IAM), lalu pilih Buat kebijakan. Di editor Visual, pilih SageMakeruntuk Layanan.

Untuk Tindakan, buka pemilih Baca (di bawah tingkat Akses) dan pilih InvokeEndpoint. Saat Anda melakukannya, ikon peringatan akan muncul.

3. Buka pemilih Resources dan pilih tautan Tambahkan ARN untuk membatasi akses di bawah Tentukan ARN sumber daya titik akhir untuk tindakan tersebut. InvokeEndpoint
4. Masukkan Wilayah AWS sumber SageMaker daya Anda dan nama titik akhir Anda. AWS Akun Anda sudah diisi sebelumnya.



5. Pilih Tambah untuk menyimpan. Pilih Berikutnya: Tag dan Berikutnya: Tinjau untuk membuka halaman terakhir proses pembuatan kebijakan.
6. Masukkan Nama dan Deskripsi untuk kebijakan ini, lalu pilih Buat kebijakan. Kebijakan dibuat dan ditambahkan ke daftar Kebijakan. Anda melihat peringatan di Konsol saat ini terjadi.
7. Di Konsol IAM, pilih Peran.
8. Pilih Buat peran.
9. Di halaman Pilih entitas tepercaya, pilih kotak layanan AWS , lalu pilih RDS untuk membuka pemilih.
10. Pilih RDS – Tambahkan Peran ke Basis Data.

11. Pilih Selanjutnya. Di halaman Tambahkan izin, temukan kebijakan yang Anda buat di langkah sebelumnya dan pilih kebijakan dari daftar tersebut. Pilih Selanjutnya
12. Berikutnya: Tinjau. Masukkan nama peran IAM dan deskripsinya.
13. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
14. Arahkan ke Wilayah AWS tempat cluster Aurora PostgreSQL DB Anda berada.
15. Di panel navigasi, pilih Databases, lalu pilih cluster Aurora PostgreSQL DB yang ingin Anda gunakan. SageMaker
16. Pilih tab Konektivitas & keamanan dan gulir untuk menemukan bagian Kelola peran IAM pada halaman. Dari pemilih Tambahkan peran IAM ke klaster ini, pilih peran yang Anda buat di langkah sebelumnya. Di pemilih Fitur, pilih SageMaker, lalu pilih Tambah peran.

Peran (dengan kebijakannya) dikaitkan dengan klaster DB Aurora PostgreSQL. Saat proses selesai, peran tersebut tercantum dalam peran IAM saat ini untuk daftar klaster ini.

Pengaturan IAM untuk SageMaker selesai. Lanjutkan penyiapan Aurora PostgreSQL Anda untuk bekerja dengan machine learning Aurora dengan menginstal ekstensi seperti yang dijelaskan dalam Menginstal ekstensi machine learning Aurora.

Menyiapkan Aurora PostgreSQL untuk menggunakan Amazon S3 untuk (Lanjutan) SageMaker

Untuk menggunakan SageMaker model Anda sendiri daripada menggunakan komponen bawaan yang disediakan oleh SageMaker, Anda perlu menyiapkan bucket Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3) untuk klaster Aurora PostgreSQL DB untuk digunakan. Ini adalah topik lanjutan, dan tidak sepenuhnya didokumentasikan dalam Panduan Pengguna Amazon Aurora ini. Proses umumnya sama dengan mengintegrasikan dukungan untuk SageMaker, sebagai berikut.

1. Buat kebijakan dan peran IAM untuk Amazon S3.
2. Tambahkan peran IAM dan impor atau ekspor Amazon S3 sebagai fitur pada tab Konektivitas & keamanan klaster Aurora PostgreSQL DB Anda.
3. Tambahkan ARN peran ke grup parameter klaster DB kustom Anda untuk klaster DB Aurora Anda.

Untuk informasi penggunaan dasar, lihat Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model (Lanjutan).

Menginstal ekstensi machine learning Aurora

Ekstensi pembelajaran mesin Aurora `aws_ml 1.0` menyediakan dua fungsi yang dapat Anda gunakan untuk memanggil Amazon Comprehend, SageMaker layanan, dan `aws_ml 2.0` menyediakan dua fungsi tambahan yang dapat Anda gunakan untuk menjalankan layanan Amazon Bedrock. Menginstal ekstensi ini di cluster Aurora PostgreSQL DB Anda juga menciptakan peran administratif untuk fitur tersebut.

> **Note**
>
> Menggunakan fungsi-fungsi ini tergantung pada penyiapan IAM untuk layanan pembelajaran mesin Aurora (Amazon Comprehend,, Amazon Bedrock) yang lengkap SageMaker, seperti yang dijelaskan dalam. Menyiapkan klaster DB Aurora PostgreSQL untuk menggunakan machine learning Aurora

- aws_comprehend.detect_sentiment – Anda menggunakan fungsi ini untuk menerapkan analisis sentimen ke teks yang disimpan dalam basis data pada klaster DB Aurora PostgreSQL Anda.
- aws_sagemaker.invoke_endpoint — Anda menggunakan fungsi ini dalam kode SQL Anda untuk berkomunikasi dengan titik akhir dari cluster Anda. SageMaker
- aws_bedrock.invoke_model — Anda menggunakan fungsi ini dalam kode SQL Anda untuk berkomunikasi dengan Model Batuan Dasar dari cluster Anda. Respon fungsi ini akan dalam format TEXT, jadi jika model merespons dalam format badan JSON maka output dari fungsi ini akan diteruskan dalam format string ke pengguna akhir.
- aws_bedrock.invoke_model_get_embeddings - Anda menggunakan fungsi ini dalam kode SQL Anda untuk memanggil Model Bedrock yang mengembalikan penyematan keluaran dalam respons JSON. Ini dapat dimanfaatkan saat Anda ingin mengekstrak penyematan yang terkait langsung dengan json-key untuk merampingkan respons dengan alur kerja yang dikelola sendiri.

Menyiapkan ekstensi machine learning Aurora dalam klaster DB Aurora PostgreSQL Anda

- Gunakan `psql` untuk terhubung ke instans penulis klaster DB Aurora PostgreSQL. Hubungkan ke basis data tertentu tempat untuk menginstal ekstensi `aws_ml`.

```
psql --host=cluster-instance-1.111122223333.aws-region.rds.amazonaws.com --
port=5432 --username=postgres --password --dbname=labdb
```

```
labdb=> CREATE EXTENSION IF NOT EXISTS aws_ml CASCADE;
NOTICE:  installing required extension "aws_commons"
CREATE EXTENSION
labdb=>
```

Menginstal `aws_ml` ekstensi juga menciptakan peran `aws_ml` administratif dan dua skema baru, sebagai berikut.

- `aws_comprehend` – Skema untuk layanan Amazon Comprehend dan sumber fungsi `detect_sentiment` (`aws_comprehend.detect_sentiment`).
- `aws_sagemaker`— Skema untuk SageMaker layanan dan sumber `invoke_endpoint` fungsi (`aws_sagemaker.invoke_endpoint`).
- `aws_bedrock`— Skema untuk layanan Amazon Bedrock dan sumber `invoke_model(aws_bedrock.invoke_model)` dan `invoke_model_get_embeddings(aws_bedrock.invoke_model_get_embeddings)` fungsi.

Peran `rds_superuser` diberi peran administratif `aws_ml` dan dibuat `OWNER` dari dua skema machine learning Aurora ini. Untuk mengizinkan pengguna basis data lain mengakses fungsi machine learning Aurora, `rds_superuser` harus memberikan hak istimewa `EXECUTE` pada fungsi machine learning Aurora. Secara default, hak istimewa `EXECUTE` dicabut dari `PUBLIC` pada fungsi dalam dua skema machine learning Aurora.

Dalam konfigurasi basis data multi-penghuni, Anda dapat mencegah penghuni mengakses fungsi machine learning Aurora dengan menggunakan `REVOKE USAGE` pada skema machine learning Aurora tertentu yang ingin dilindungi.

Menggunakan Amazon Bedrock dengan cluster Aurora PostgreSQL DB Anda

Untuk Aurora PostgreSQL, pembelajaran mesin Aurora menyediakan fungsi Amazon Bedrock berikut untuk bekerja dengan data teks Anda. Fungsi ini hanya tersedia setelah Anda menginstal ekstensi `aws_ml` 2.0 dan menyelesaikan semua prosedur pengaturan. Untuk informasi selengkapnya, lihat Menyiapkan klaster DB Aurora PostgreSQL untuk menggunakan machine learning Aurora.

aws_bedrock.invoke_model

Fungsi ini mengambil teks yang diformat dalam JSON sebagai input dan memprosesnya untuk berbagai model yang dihosting di Amazon Bedrock dan mendapatkan kembali respons teks JSON dari model. Respons ini bisa berisi teks, gambar, atau embeddings. Ringkasan dokumentasi fungsinya adalah sebagai berikut.

```
aws_bedrock.invoke_model(
   IN model_id      varchar,
   IN content_type  text,
   IN accept_type   text,
   IN model_input   text,
   OUT model_output varchar)
```

Input dan output dari fungsi ini adalah sebagai berikut.

- `model_id`— Pengidentifikasi model.
- `content_type`— Jenis permintaan untuk model Bedrock.
- `accept_type`— Jenis respons yang diharapkan dari model Bedrock. Biasanya aplikasi/JSON untuk sebagian besar model.
- `model_input`— Prompts; satu set input tertentu ke model dalam format seperti yang ditentukan oleh content_type. Untuk informasi lebih lanjut tentang format/struktur permintaan yang diterima model, lihat Parameter inferensi untuk model dasar.
- `model_output`— Output model Bedrock sebagai teks.

Contoh berikut menunjukkan cara memanggil model Anthropic Claude 2 untuk Bedrock menggunakan invoke_model.

Example Contoh: Kueri sederhana menggunakan fungsi Amazon Bedrock

```
SELECT aws_bedrock.invoke_model (
    model_id    := 'anthropic.claude-v2',
    content_type:= 'application/json',
    accept_type := 'application/json',
    model_input := '{"prompt": "\n\nHuman: You are a helpful assistant that answers
 questions directly and only using the information provided in the context below.
\nDescribe the answer
```

```
    in detail.\n\nContext: %s \n\nQuestion: %s \n
\nAssistant:","max_tokens_to_sample":4096,"temperature":0.5,"top_k":250,"top_p":0.5,"stop_seque
[]}'
);
```

aws_bedrock.invoke_model_get_embeddings

Output model dapat menunjuk ke embeddings vektor untuk beberapa kasus. Mengingat respons bervariasi per model, fungsi lain invoke_model_get_embeddings dapat dimanfaatkan yang berfungsi persis seperti invoke_model tetapi mengeluarkan penyematan dengan menentukan json-key yang sesuai.

```
aws_bedrock.invoke_model_get_embeddings(
   IN model_id      varchar,
   IN content_type  text,
   IN json_key      text,
   IN model_input   text,
   OUT model_output float8[])
```

Input dan output dari fungsi ini adalah sebagai berikut.

- `model_id`— Pengidentifikasi model.
- `content_type`— Jenis permintaan untuk model Bedrock. Di sini, accept_type diatur ke nilai default. `application/json`
- `model_input`— Prompts; satu set input tertentu ke Model dalam format seperti yang ditentukan oleh content_type. Untuk informasi lebih lanjut tentang format/struktur permintaan yang diterima Model, lihat Parameter inferensi untuk model dasar.
- `json_key`— Referensi ke bidang untuk mengekstrak embedding dari. Ini dapat bervariasi jika model penyematan berubah.
- `model_output`— Output model Bedrock sebagai array embeddings yang memiliki desimal 16 bit.

Contoh berikut menunjukkan cara menghasilkan embedding menggunakan Titan Embeddings G1 — Model penyematan teks untuk frase PostgreSQL I/O monitoring views.

Example Contoh: Kueri sederhana menggunakan fungsi Amazon Bedrock

```
SELECT aws_bedrock.invoke_model_get_embeddings(
```

```
    model_id      := 'amazon.titan-embed-text-v1',
    content_type  := 'application/json',
    json_key      := 'embedding',
    model_input   := '{ "inputText": "PostgreSQL I/O monitoring views"}') AS embedding;
```

Menggunakan Amazon Comprehend dengan klaster DB Aurora PostgreSQL

Untuk Aurora PostgreSQL, machine learning Aurora menyediakan fungsi Amazon Comprehend berikut agar berfungsi dengan data teks Anda. Fungsi ini hanya tersedia setelah Anda menginstal ekstensi `aws_ml` dan menyelesaikan semua prosedur penyiapan. Untuk informasi selengkapnya, lihat Menyiapkan klaster DB Aurora PostgreSQL untuk menggunakan machine learning Aurora.

aws_comprehend.detect_sentiment

Fungsi ini mengambil teks sebagai input dan mengevaluasi apakah teks memiliki postur emosional positif, negatif, netral, atau campuran. Fungsi ini menghasilkan sentimen ini bersama dengan tingkat kepercayaan untuk evaluasinya. Ringkasan dokumentasi fungsinya adalah sebagai berikut.

```
aws_comprehend.detect_sentiment(
   IN input_text varchar,
   IN language_code varchar,
   IN max_rows_per_batch int,
   OUT sentiment varchar,
   OUT confidence real)
```

Input dan output dari fungsi ini adalah sebagai berikut.

- `input_text` – Teks untuk mengevaluasi dan menetapkan sentimen (negatif, positif, netral, campuran).
- `language_code` – Bahasa `input_text` yang diidentifikasi menggunakan ID 2-huruf ISO 639-1 dengan sub-tag regional (sesuai kebutuhan) atau kode tiga huruf ISO 639-2, yang sesuai. Misalnya, `en` adalah kode untuk bahasa Inggris, `zh` adalah kode untuk bahasa Mandarin yang disederhanakan. Untuk informasi selengkapnya, lihat Bahasa yang didukung di Panduan Developer Amazon Comprehend.
- `max_rows_per_batch` – Jumlah baris maksimum per batch untuk pemrosesan mode batch. Untuk informasi selengkapnya, lihat Memahami mode batch dan fungsi machine learning Aurora.

- `sentiment` – Sentimen teks input, yang diidentifikasi sebagai POSITIVE, NEGATIVE, NEUTRAL, atau MIXED.
- `confidence` – Tingkat kepercayaan dalam keakuratan yang ditentukan `sentiment`. Rentang nilai dari 0,0 hingga 1,0.

Berikut ini, Anda dapat menemukan contoh cara menggunakan fungsi ini.

Example Contoh: Kueri sederhana menggunakan fungsi Amazon Comprehend

Berikut adalah contoh kueri sederhana yang menginvokasi fungsi ini untuk menilai kepuasan pelanggan dengan tim dukungan Anda. Misalkan Anda memiliki tabel basis data (`support`) yang menyimpan umpan balik pelanggan setelah setiap permintaan bantuan. Contoh kueri ini menerapkan fungsi `aws_comprehend.detect_sentiment` ke teks di kolom `feedback` tabel dan menghasilkan sentimen dan tingkat kepercayaan untuk sentimen tersebut. Kueri ini juga menghasilkan urutan menurun.

```
SELECT feedback, s.sentiment,s.confidence
    FROM support,aws_comprehend.detect_sentiment(feedback, 'en') s
    ORDER BY s.confidence DESC;
 feedback                          | sentiment | confidence
 ------------------------------------------------------------+-----------+------------
 Thank you for the excellent customer support!              | POSITIVE  |   0.999771
 The latest version of this product stinks!                 | NEGATIVE  |   0.999184
 Your support team is just awesome! I am blown away.        | POSITIVE  |   0.997774
 Your product is too complex, but your support is great.    | MIXED     |   0.957958
 Your support tech helped me in fifteen minutes.            | POSITIVE  |   0.949491
 My problem was never resolved!                             | NEGATIVE  |   0.920644
 When will the new version of this product be released?     | NEUTRAL   |   0.902706
 I cannot stand that chatbot.                               | NEGATIVE  |   0.895219
 Your support tech talked down to me.                       | NEGATIVE  |   0.868598
 It took me way too long to get a real person.              | NEGATIVE  |   0.481805

 (10 rows)
```

Agar tidak ditagih untuk deteksi sentimen lebih dari sekali per baris tabel, Anda dapat mewujudkan hasilnya. Lakukan ini di baris yang diinginkan. Misalnya, catatan dokter sedang diperbarui sehingga hanya yang berbahasa Prancis (`fr`) yang menggunakan fungsi deteksi sentimen.

```
UPDATE clinician_notes
```

```
SET sentiment = (aws_comprehend.detect_sentiment (french_notes, 'fr')).sentiment,
    confidence = (aws_comprehend.detect_sentiment (french_notes, 'fr')).confidence
WHERE
    clinician_notes.french_notes IS NOT NULL AND
    LENGTH(TRIM(clinician_notes.french_notes)) > 0 AND
    clinician_notes.sentiment IS NULL;
```

Untuk informasi selengkapnya tentang cara mengoptimalkan panggilan fungsi Anda, lihat Pertimbangan performa untuk machine learning Aurora dengan Aurora PostgreSQL.

Menggunakan SageMaker dengan cluster DB PostgreSQL Aurora Anda

Setelah menyiapkan SageMaker lingkungan Anda dan mengintegrasikan dengan Aurora PostgreSQL seperti yang Menyiapkan Aurora PostgreSQL untuk menggunakan Amazon SageMaker diuraikan dalam, Anda dapat memanggil operasi dengan menggunakan fungsi. `aws_sagemaker.invoke_endpoint` Fungsi `aws_sagemaker.invoke_endpoint` hanya terhubung ke titik akhir model dalam Wilayah AWS yang sama. Jika instance database Anda memiliki replika dalam beberapa, Wilayah AWS pastikan Anda menyiapkan dan menerapkan setiap SageMaker model ke setiap model. Wilayah AWS

Panggilan ke `aws_sagemaker.invoke_endpoint` diautentikasi menggunakan peran IAM yang Anda atur untuk mengaitkan klaster DB PostgreSQL Aurora Anda dengan SageMaker layanan dan titik akhir yang Anda berikan selama proses penyiapan. SageMaker titik akhir model dicakup ke akun individu dan tidak bersifat publik. `endpoint_name`URL tidak berisi ID akun. SageMaker menentukan ID akun dari token otentikasi yang disediakan oleh peran SageMaker IAM dari instance database.

aws_sagemaker.invoke_endpoint

Fungsi ini mengambil SageMaker endpoint sebagai input dan jumlah baris yang harus diproses sebagai batch. Ini juga mengambil sebagai masukan berbagai parameter yang diharapkan oleh titik akhir SageMaker model. Dokumentasi referensi fungsi ini adalah sebagai berikut.

```
aws_sagemaker.invoke_endpoint(
  IN endpoint_name varchar,
  IN max_rows_per_batch int,
  VARIADIC model_input "any",
  OUT model_output varchar
  )
```

Input dan output dari fungsi ini adalah sebagai berikut.

- `endpoint_name`— URL titik akhir yang Wilayah AWS—independen.
- `max_rows_per_batch` – Jumlah baris maksimum per batch untuk pemrosesan mode batch. Untuk informasi selengkapnya, lihat Memahami mode batch dan fungsi machine learning Aurora.
- `model_input` – Satu atau beberapa parameter input untuk model. Ini bisa berupa tipe data apa pun yang dibutuhkan oleh SageMaker model. PostgreSQL memungkinkan Anda menentukan hingga 100 parameter input untuk satu fungsi. Tipe data array harus satu dimensi, tetapi dapat berisi elemen sebanyak yang diharapkan oleh SageMaker model. Jumlah input ke SageMaker model hanya dibatasi oleh batas ukuran pesan SageMaker 6 MB.
- `model_output`— Output SageMaker model sebagai teks.

Membuat fungsi yang ditentukan pengguna untuk memanggil model SageMaker

Buat fungsi yang ditentukan pengguna terpisah `aws_sagemaker.invoke_endpoint` untuk memanggil setiap model Anda SageMaker . Fungsi yang ditentukan pengguna Anda mewakili SageMaker titik akhir yang menghosting model. Fungsi `aws_sagemaker.invoke_endpoint` berjalan dalam fungsi yang ditentukan pengguna. Fungsi yang ditentukan pengguna memberikan banyak keuntungan:

- Anda dapat memberikan nama SageMaker model Anda sendiri alih-alih hanya memanggil `aws_sagemaker.invoke_endpoint` semua SageMaker model Anda.
- Anda dapat menentukan URL titik akhir model hanya di satu tempat di kode aplikasi SQL Anda.
- Anda dapat mengontrol hak istimewa `EXECUTE` untuk setiap fungsi machine learning Aurora secara mandiri.
- Anda dapat menyatakan jenis input dan output model menggunakan jenis SQL. SQL memberlakukan jumlah dan jenis argumen yang diteruskan ke SageMaker model Anda dan melakukan konversi tipe jika perlu. Menggunakan tipe SQL juga akan menerjemahkan `SQL NULL` ke nilai default yang sesuai yang diharapkan oleh SageMaker model Anda.
- Anda dapat mengurangi ukuran batch maksimum jika Anda ingin menampilkan beberapa baris pertama sedikit lebih cepat.

Untuk menentukan fungsi yang ditentukan pengguna, gunakan pernyataan bahasa definisi data (DDL) SQL `CREATE FUNCTION`. Saat menentukan fungsi, Anda juga menentukan:

- Parameter input ke model.
- SageMaker Titik akhir spesifik untuk dipanggil.

- Jenis yang ditampilkan.

Fungsi yang ditentukan pengguna mengembalikan inferensi yang dihitung oleh SageMaker titik akhir setelah menjalankan model pada parameter input. Contoh berikut membuat fungsi yang ditentukan pengguna untuk SageMaker model dengan dua parameter input.

```
CREATE FUNCTION classify_event (IN arg1 INT, IN arg2 DATE, OUT category INT)
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', NULL,
        arg1, arg2                          -- model inputs are separate arguments
        )::INT                              -- cast the output to INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Perhatikan hal berikut:

- Input fungsi `aws_sagemaker.invoke_endpoint` dapat berupa satu parameter atau lebih dari jenis data apa pun.
- Contoh ini menggunakan jenis output INT. Jika Anda mengirimkan output dari jenis `varchar` ke jenis yang berbeda, output ini harus dikirimkan ke jenis tipe skalar default PostgreSQL seperti `INTEGER`, `REAL`, `FLOAT`, atau `NUMERIC`. Untuk informasi selengkapnya tentang jenis ini, lihat Data types dalam dokumentasi PostgreSQL.
- Tentukan `PARALLEL SAFE` untuk mengaktifkan pemrosesan kueri paralel. Untuk informasi selengkapnya, lihat Meningkatkan waktu respons dengan pemrosesan kueri paralel.
- Tentukan `COST 5000` untuk mengestimasi biaya menjalankan fungsi. Gunakan bilangan positif yang memberikan estimasi biaya eksekusi untuk fungsi tersebut, dalam unit `cpu_operator_cost`.

Melewati array sebagai masukan ke SageMaker model

Fungsi `aws_sagemaker.invoke_endpoint` dapat memiliki hingga 100 parameter input, yang merupakan batas untuk fungsi PostgreSQL. Jika SageMaker model membutuhkan lebih dari 100 parameter dari jenis yang sama, berikan parameter model sebagai array.

Contoh berikut mendefinisikan fungsi yang melewati array sebagai masukan ke model SageMaker regresi. Output dikirim ke nilai `REAL`.

```
CREATE FUNCTION regression_model (params REAL[], OUT estimate REAL)
```

```
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
      'sagemaker_model_endpoint_name',
      NULL,
      params
      )::REAL
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Menentukan ukuran batch saat menjalankan model SageMaker

Contoh berikut membuat fungsi yang ditentukan pengguna untuk SageMaker model yang menetapkan ukuran batch default ke NULL. Fungsi ini juga memungkinkan Anda untuk memberikan ukuran batch yang berbeda saat Anda menginvokasinya.

```
CREATE FUNCTION classify_event (
    IN event_type INT, IN event_day DATE, IN amount REAL, -- model inputs
    max_rows_per_batch INT DEFAULT NULL,  -- optional batch size limit
    OUT category INT)                     -- model output
AS $$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', max_rows_per_batch,
        event_type, event_day, COALESCE(amount, 0.0)
        )::INT              -- casts output to type INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Perhatikan hal berikut:

- Gunakan parameter `max_rows_per_batch` opsional untuk memberikan kontrol jumlah baris untuk invokasi fungsi mode batch. Jika Anda menggunakan nilai NULL, maka pengoptimal kueri secara otomatis memilih ukuran batch maksimum. Untuk informasi selengkapnya, lihat Memahami mode batch dan fungsi machine learning Aurora.
- Secara default, meneruskan NULL sebagai nilai parameter diterjemahkan ke string kosong sebelum diteruskan ke SageMaker. Untuk contoh ini, input memiliki jenis-jenis yang berbeda.
- Jika Anda memiliki masukan non-teks, atau input teks yang perlu diatur ke default ke beberapa nilai selain string kosong, gunakan pernyataan `COALESCE`. Gunakan `COALESCE` untuk menerjemahkan NULL ke nilai pengganti null yang diinginkan dalam panggilan ke `aws_sagemaker.invoke_endpoint`. Untuk parameter `amount` dalam contoh ini, nilai NULL dikonversi menjadi 0,0.

Memohon SageMaker model yang memiliki banyak output

Contoh berikut menciptakan fungsi yang ditentukan pengguna untuk SageMaker model yang mengembalikan beberapa output. Fungsi Anda perlu mentransmisikan output dari fungsi `aws_sagemaker.invoke_endpoint` ke jenis data yang terkait. Misalnya, Anda dapat menggunakan jenis titik PostgreSQL default untuk pasangan (x,y) atau jenis komposit yang ditentukan oleh pengguna.

Fungsi yang ditentukan oleh pengguna ini menampilkan nilai dari model yang menampilkan beberapa output menggunakan jenis gabungan untuk output.

```
CREATE TYPE company_forecasts AS (
    six_month_estimated_return real,
    one_year_bankruptcy_probability float);
CREATE FUNCTION analyze_company (
    IN free_cash_flow NUMERIC(18, 6),
    IN debt NUMERIC(18,6),
    IN max_rows_per_batch INT DEFAULT NULL,
    OUT prediction company_forecasts)
AS $$
    SELECT (aws_sagemaker.invoke_endpoint('endpt_name',
       max_rows_per_batch,free_cash_flow, debt))::company_forecasts;


$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Untuk jenis komposit, gunakan kolom dalam urutan yang sama seperti yang muncul di output model dan transmisikan keluaran `aws_sagemaker.invoke_endpoint` ke jenis komposit Anda. Pemanggil dapat mengekstrak kolom satu per satu, baik dengan nama maupun dengan notasi ".*" PostgreSQL.

Mengekspor data ke Amazon S3 SageMaker untuk pelatihan model (Lanjutan)

Kami menyarankan agar Anda terbiasa dengan pembelajaran mesin Aurora dan SageMaker dengan menggunakan algoritme dan contoh yang disediakan daripada mencoba melatih model Anda sendiri. Untuk informasi selengkapnya, lihat Memulai Amazon SageMaker

Untuk melatih SageMaker model, Anda mengekspor data ke bucket Amazon S3. Bucket Amazon S3 digunakan oleh SageMaker untuk melatih model Anda sebelum digunakan. Anda dapat membuat

kueri data dari klaster DB Aurora PostgreSQL dan menyimpannya langsung ke dalam file teks yang disimpan dalam bucket Amazon S3. Kemudian SageMaker mengkonsumsi data dari bucket Amazon S3 untuk pelatihan. Untuk informasi lebih lanjut tentang pelatihan SageMaker model, lihat Melatih model dengan Amazon SageMaker.

> **Note**
>
> Saat Anda membuat bucket Amazon S3 untuk pelatihan SageMaker model atau penilaian batch, gunakan `sagemaker` nama bucket Amazon S3. Untuk informasi selengkapnya, lihat Menentukan Bucket Amazon S3 untuk Mengunggah Kumpulan Data Pelatihan dan Menyimpan Data Output di Panduan Pengembang Amazon. SageMaker

Untuk informasi lebih lanjut tentang mengekspor data Anda, lihat Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3.

Pertimbangan performa untuk machine learning Aurora dengan Aurora PostgreSQL

Amazon SageMaker Comprehend dan layanan melakukan sebagian besar pekerjaan ketika dipanggil oleh fungsi pembelajaran mesin Aurora. Itu berarti Anda dapat menskalakan sumber daya tersebut sesuai kebutuhan, secara mandiri. Untuk klaster DB Aurora PostgreSQL, Anda dapat membuat panggilan fungsi seefisien mungkin. Berikut ini, Anda dapat menemukan beberapa pertimbangan performa yang perlu diperhatikan saat bekerja dengan machine learning Aurora dari Aurora PostgreSQL.

Topik

- Memahami mode batch dan fungsi machine learning Aurora
- Meningkatkan waktu respons dengan pemrosesan kueri paralel
- Menggunakan tampilan terwujud dan kolom terwujud

Memahami mode batch dan fungsi machine learning Aurora

Biasanya, PostgreSQL menjalankan fungsi satu baris dalam satu waktu. Machine learning Aurora dapat mengurangi overhead ini dengan menggabungkan panggilan ke layanan machine learning Aurora eksternal untuk banyak baris menjadi batch dengan pendekatan yang disebut eksekusi mode-

batch. Dalam mode batch, machine learning Aurora menerima respons untuk batch baris input, lalu mengirimkan respons kembali ke kueri yang berjalan satu baris dalam satu waktu. Pengoptimalan ini meningkatkan throughput kueri Aurora Anda tanpa membatasi pengoptimal kueri PostgreSQL.

Aurora secara otomatis menggunakan mode batch jika fungsi direferensikan dari daftar `SELECT`, klausul `WHERE`, atau klausul `HAVING`. Perhatikan bahwa ekspresi `CASE` sederhana tingkat atas memenuhi syarat untuk eksekusi mode batch. Ekspresi `CASE` tingkat atas yang ditelusuri juga memenuhi syarat untuk eksekusi mode-batch asalkan klausul `WHEN` pertama adalah predikat sederhana dengan panggilan fungsi mode-batch.

Fungsi yang Anda tentukan harus berupa fungsi `LANGUAGE SQL` dan harus mencantumkan `PARALLEL SAFE` dan `COST 5000`.

Migrasi fungsi dari pernyataan SELECT ke klausul FROM

Biasanya, fungsi `aws_ml` yang memenuhi syarat untuk eksekusi mode-batch secara otomatis dimigrasikan oleh Aurora ke klausul `FROM`.

Migrasi fungsi mode batch yang memenuhi syarat ke klausul FROM dapat diperiksa secara manual pada tingkat per kueri. Untuk melakukan ini, gunakan pernyataan EXPLAIN (serta ANALYZE dan VERBOSE) dan cari informasi "Pemrosesan Batch" di bawah setiap mode batch `Function Scan`. Anda juga dapat menggunakan EXPLAIN (dengan VERBOSE) tanpa menjalankan kueri. Selanjutnya, amati apakah panggilan ke fungsi tersebut muncul sebagai `Function Scan` di dalam loop join bertingkat yang tidak ditentukan dalam pernyataan asli.

Dalam contoh berikut, operator loop join bertingkat dalam rencana menunjukkan bahwa Aurora memigrasi fungsi `anomaly_score`. Fungsi ini dimigrasikan dari daftar SELECT ke klausul FROM, yang memenuhi syarat untuk eksekusi mode-batch.

```
EXPLAIN (VERBOSE, COSTS false)
SELECT anomaly_score(ts.R.description) from ts.R;
                          QUERY PLAN
---------------------------------------------------------------
 Nested Loop
   Output: anomaly_score((r.description)::text)
   ->  Seq Scan on ts.r
         Output: r.id, r.description, r.score
   ->  Function Scan on public.anomaly_score
         Output: anomaly_score.anomaly_score
         Function Call: anomaly_score((r.description)::text)
```

Untuk menonaktifkan eksekusi mode-batch, atur parameter `apg_enable_function_migration` ke `false`. Hal ini mencegah migrasi dari fungsi `aws_ml` dari SELECT ke klausul `FROM`. Berikut caranya.

```
SET apg_enable_function_migration = false;
```

Parameter `apg_enable_function_migration` adalah parameter Grand Unified Configuration (GUC) yang dikenali oleh ekstensi `apg_plan_mgmt` Aurora PostgreSQL untuk manajemen rencana kueri. Untuk menonaktifkan migrasi fungsi dalam sesi, gunakan manajemen rencana kueri untuk menyimpan rencana yang dihasilkan sebagai rencana `approved`. Saat runtime, manajemen rencana kueri menerapkan rencana `approved` dengan pengaturan `apg_enable_function_migration`-nya. Penerapan ini terjadi, terlepas dari pengaturan parameter GUC `apg_enable_function_migration`. Untuk informasi selengkapnya, lihat Mengelola rencana eksekusi kueri untuk Aurora PostgreSQL.

Menggunakan parameter max_rows_per_batch

Baik fungsi `aws_comprehend.detect_sentiment` maupun fungsi `aws_sagemaker.invoke_endpoint` memiliki parameter `max_rows_per_batch`. Parameter ini menentukan jumlah baris yang dapat dikirim ke layanan machine learning Aurora. Semakin besar set data yang diproses oleh fungsi Anda, semakin besar ukuran batch yang bisa Anda buat.

Fungsi mode batch meningkatkan efisiensi dengan membangun batch baris yang menyebarkan biaya panggilan fungsi machine learning Aurora melalui banyak baris. Namun, jika pernyataan `SELECT` selesai lebih awal karena klausul `LIMIT`, maka konsep batch dapat dibuat di lebih banyak baris daripada yang digunakan kueri. Pendekatan ini dapat mengakibatkan biaya tambahan ke AWS akun Anda. Untuk memanfaatkan eksekusi mode batch, tetapi menghindari pembuatan batch yang terlalu besar, gunakan nilai yang lebih kecil untuk parameter `max_rows_per_batch` dalam panggilan fungsi Anda.

Jika Anda melakukan `EXPLAIN` (`VERBOSE`, `ANALYZE`) kueri yang menggunakan eksekusi mode batch, Anda akan melihat operator `FunctionScan` yang berada di bawah loop join bertingkat. Jumlah loop yang dilaporkan `EXPLAIN` sama dengan frekuensi sebuah baris diambil dari operator `FunctionScan`. Jika pernyataan menggunakan klausul LIMIT, jumlah pengambilannya konsisten. Untuk mengoptimalkan ukuran batch, atur parameter `max_rows_per_batch` ke nilai ini. Namun, jika fungsi mode batch direferensikan dalam predikat dalam klausul `WHERE` atau klausul `HAVING`, Anda mungkin tidak dapat mengetahui jumlah pengambilan sebelumnya. Dalam kasus ini, gunakan loop

sebagai pedoman dan percobaan dengan `max_rows_per_batch` untuk menemukan pengaturan yang mengoptimalkan performa.

Memverifikasi eksekusi mode batch

Untuk melihat apakah fungsi berjalan dalam mode batch, gunakan `EXPLAIN ANALYZE`. Jika eksekusi mode batch digunakan, rencana kueri akan menyertakan informasi di bagian "Pemrosesan Batch".

```
EXPLAIN ANALYZE SELECT user-defined-function();
 Batch Processing: num batches=1 avg/min/max batch size=3333.000/3333.000/3333.000
                                 avg/min/max batch call time=146.273/146.273/146.273
```

Dalam contoh ini, ada 1 batch yang berisi 3.333 baris, yang memerlukan waktu proses 146,273 md. Bagian "Pemrosesan Batch" menunjukkan berikut:

- Banyaknya batch yang tersedia untuk operasi pemindaian fungsi ini
- Ukuran batch rata-rata, minimum, dan maksimum
- Rata-rata waktu eksekusi batch, minimum, dan maksimum

Biasanya batch akhir lebih kecil daripada batch lain, yang sering kali menghasilkan ukuran batch minimum yang jauh lebih kecil dari rata-rata.

Untuk menampilkan beberapa baris pertama dengan lebih cepat, atur parameter `max_rows_per_batch` ke nilai yang lebih kecil.

Untuk mengurangi jumlah panggilan mode batch ke layanan ML saat Anda menggunakan `LIMIT` dalam fungsi yang ditentukan pengguna, atur parameter `max_rows_per_batch` ke nilai yang lebih kecil.

Meningkatkan waktu respons dengan pemrosesan kueri paralel

Untuk mendapatkan hasil secepat mungkin dari banyak baris, Anda dapat menggabungkan pemrosesan kueri paralel dengan pemrosesan mode batch. Anda dapat menggunakan pemrosesan kueri paralel untuk pernyataan `SELECT`, `CREATE TABLE AS SELECT`, dan `CREATE MATERIALIZED VIEW`.

> **Note**
>
> PostgreSQL belum mendukung kueri paralel untuk pernyataan bahasa manipulasi data (DML).

Pemrosesan kueri paralel terjadi baik dalam basis data maupun dalam layanan ML. Jumlah inti dalam kelas instans basis data membatasi tingkat paralelisme yang dapat digunakan saat menjalankan kueri. Server basis data dapat membuat konsep rencana eksekusi kueri paralel yang membagi tugas di antara sekumpulan pekerja paralel. Selanjutnya, setiap pekerja ini bisa membangun permintaan batch yang berisi puluhan ribu baris (atau sebanyak yang diizinkan oleh setiap layanan).

Permintaan batch dari semua pekerja paralel dikirim ke titik SageMaker akhir. Tingkat paralelisme yang dapat didukung titik akhir dibatasi oleh jumlah dan jenis instans yang mendukungnya. Untuk K tingkat paralelisme, Anda memerlukan kelas instans basis data yang memiliki setidaknya K inti. Anda juga perlu mengonfigurasi SageMaker titik akhir agar model Anda memiliki K instance awal dari kelas instance berkinerja cukup tinggi.

Untuk menggunakan pemrosesan kueri paralel, Anda dapat mengatur parameter `parallel_workers` penyimpanan dari tabel yang berisi data yang ingin Anda teruskan. Anda mengatur `parallel_workers` ke fungsi mode-batch seperti `aws_comprehend.detect_sentiment`. Jika pengoptimal memilih paket query paralel, layanan AWS ML dapat dipanggil baik dalam batch maupun paralel.

Anda dapat menggunakan parameter berikut dengan fungsi `aws_comprehend.detect_sentiment` untuk mendapatkan rencana dengan paralelisme empat arah. Jika Anda mengubah salah satu dari dua parameter berikut, Anda harus memulai ulang instans basis data untuk menerapkan perubahan.

```
-- SET max_worker_processes to 8;  -- default value is 8
-- SET max_parallel_workers to 8;  -- not greater than max_worker_processes
SET max_parallel_workers_per_gather to 4;  -- not greater than max_parallel_workers

-- You can set the parallel_workers storage parameter on the table that the data
-- for the Aurora machine learning function is coming from in order to manually
 override the degree of
-- parallelism that would otherwise be chosen by the query optimizer
--
ALTER TABLE yourTable SET (parallel_workers = 4);
```

```
-- Example query to exploit both batch-mode execution and parallel query
EXPLAIN (verbose, analyze, buffers, hashes)
SELECT aws_comprehend.detect_sentiment(description, 'en')).*
FROM yourTable
WHERE id < 100;
```

Untuk informasi selengkapnya tentang cara mengontrol kueri paralel, lihat Parallel plans di dokumentasi PostgreSQL.

Menggunakan tampilan terwujud dan kolom terwujud

Ketika Anda memanggil AWS layanan seperti SageMaker atau Amazon Comprehend dari database Anda, akun Anda dibebankan sesuai dengan kebijakan harga layanan tersebut. Untuk meminimalkan biaya ke akun Anda, Anda dapat mewujudkan hasil pemanggilan AWS layanan ke kolom yang terwujud sehingga AWS layanan tidak dipanggil lebih dari sekali per baris input. Jika diinginkan, Anda dapat menambahkan kolom stempel waktu `materializedAt` untuk mencatat waktu saat kolom tersebut dibuat.

Latensi pernyataan `INSERT` baris tunggal biasa biasanya jauh lebih kecil daripada latensi pemanggilan fungsi mode batch. Dengan demikian, Anda mungkin tidak dapat memenuhi persyaratan latensi aplikasi Anda jika menginvokasi fungsi mode batch untuk setiap baris tunggal `INSERT` yang dijalankan oleh aplikasi Anda. Untuk mewujudkan hasil pemanggilan AWS layanan ke dalam kolom yang terwujud, aplikasi berkinerja tinggi umumnya perlu mengisi kolom yang terwujud. Untuk melakukannya, aplikasi tersebut secara berkala mengeluarkan pernyataan `UPDATE` yang beroperasi pada batch baris yang besar secara bersamaan.

`UPDATE` menggunakan penguncian level baris yang dapat memengaruhi aplikasi yang sedang berjalan. Jadi, Anda mungkin perlu menggunakan `SELECT ... FOR UPDATE SKIP LOCKED`, atau menggunakan `MATERIALIZED VIEW`.

Kueri analitik yang beroperasi pada banyak baris secara real-time dapat menggabungkan perwujudan mode batch dengan pemrosesan real-time. Untuk melakukan ini, kueri ini mengumpulkan `UNION ALL` dari hasil yang telah diwujudkan sebelumnya dengan kueri di atas baris yang belum memiliki hasil yang terwujud. Dalam beberapa kasus, `UNION ALL` seperti itu diperlukan di banyak tempat, atau kueri dibuat oleh aplikasi pihak ketiga. Jika demikian, Anda dapat membuat `VIEW` untuk merangkum operasi `UNION ALL` sehingga detail ini tidak diekspos ke aplikasi SQL lainnya.

Anda bisa menggunakan tampilan terwujud untuk mewujudkan hasil dari pernyataan `SELECT` arbitrer pada snapshot tepat waktu. Anda juga dapat menggunakannya untuk menyegarkan tampilan

terwujud kapan saja di masa mendatang. Saat ini, PostgreSQL tidak mendukung refresh inkremental, sehingga setiap kali tampilan terwujud di-refresh, tampilan terwujud akan dihitung ulang sepenuhnya.

Anda dapat me-refresh tampilan terwujud dengan opsi `CONCURRENTLY`, yang memperbarui konten tampilan terwujud tanpa mengambil kunci eksklusif. Tindakan ini memungkinkan aplikasi SQL membaca dari tampilan terwujud saat sedang di-refresh.

Memantau machine learning Aurora

Anda dapat memantau fungsi `aws_ml` dengan mengatur parameter `track_functions` di grup parameter klaster DB kustom Anda ke `all`. Secara default, parameter ini diatur ke `pl` yang berarti bahwa hanya fungsi bahasa prosedur yang akan dilacak. Dengan mengubah ini menjadi `all`, fungsi `aws_ml` juga akan dilacak. Untuk informasi selengkapnya, lihat Run-time Statistics dalam dokumentasi PostgreSQL.

Untuk informasi tentang pemantauan kinerja SageMaker operasi yang dipanggil dari fungsi pembelajaran mesin Aurora, lihat Memantau Amazon SageMaker di Panduan SageMaker Pengembang Amazon.

Dengan `track_functions` diatur ke `all`, Anda dapat mengueri tampilan `pg_stat_user_functions` untuk mendapatkan statistik tentang fungsi yang Anda tentukan dan gunakan untuk menginvokasi layanan machine learning Aurora. Untuk setiap fungsi, tampilan ini memberikan jumlah `calls`, `total_time`, dan `self_time`.

Untuk melihat statistik untuk fungsi `aws_sagemaker.invoke_endpoint` dan `aws_comprehend.detect_sentiment`, Anda dapat memfilter hasil berdasarkan nama skema menggunakan kueri berikut.

```
SELECT * FROM pg_stat_user_functions
    WHERE schemaname
    LIKE 'aws_%';
```

Untuk menghapus statistik, ikuti langkah berikut.

```
SELECT pg_stat_reset();
```

Anda bisa mendapatkan nama fungsi SQL Anda yang memanggil fungsi `aws_sagemaker.invoke_endpoint` dengan mengueri katalog sistem `pg_proc` PostgreSQL. Katalog ini menyimpan informasi tentang fungsi, prosedur, dan banyak lagi. Untuk informasi

selengkapnya, lihat pg_proc dalam dokumentasi PostgreSQL. Berikut ini adalah contoh mengueri tabel untuk mendapatkan nama-nama fungsi (`proname`) yang sumbernya (`prosrc`) mencakup teks invoke_endpoint.

```
SELECT proname FROM pg_proc WHERE prosrc LIKE '%invoke_endpoint%';
```

Contoh kode untuk Aurora menggunakan SDK AWS

Contoh kode berikut menunjukkan cara menggunakan Aurora dengan kit pengembangan AWS perangkat lunak (SDK).

Tindakan merupakan kutipan kode dari program yang lebih besar dan harus dijalankan dalam konteks. Meskipun tindakan menunjukkan cara memanggil setiap fungsi layanan, Anda dapat melihat tindakan dalam konteks pada skenario yang terkait dan contoh lintas layanan.

Skenario adalah contoh kode yang menunjukkan cara untuk menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam layanan yang sama.

Contoh lintas layanan adalah contoh aplikasi yang bekerja di beberapa Layanan AWS.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga berisi informasi tentang cara memulai dan detail tentang versi SDK sebelumnya.

Memulai

Halo Aurora

Contoh kode berikut ini menunjukkan cara mulai menggunakan Aurora.

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

```
namespace AuroraActions;

public static class HelloAurora
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the
        // Amazon Relational Database Service (Amazon RDS).
        // Use your AWS profile name, or leave it blank to use the default
 profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
            ).Build();

        // Now the client is available for injection. Fetching it directly here
 for example purposes only.
        var rdsClient = host.Services.GetRequiredService<IAmazonRDS>();

        // You can use await and any of the async methods to get a response.
        var response = await rdsClient.DescribeDBClustersAsync(new
 DescribeDBClustersRequest { IncludeShared = true });
        Console.WriteLine($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        foreach (var cluster in response.DBClusters)
        {
            Console.WriteLine($"\tCluster: database: {cluster.DatabaseName}
 identifier: {cluster.DBClusterIdentifier}.");
        }
    }
}
```

- Untuk detail API, lihat DescribeDBClusters di Referensi AWS SDK for .NET API.

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Kode untuk file CMake MakeLists C.txt.

```
# Set the minimum required version of CMake for this project.
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS rds)

# Set this project's name.
project("hello_aurora")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.
```

```
    # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                    # and set the proper subdirectory to the
 executables' location.

    AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_aurora.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Kode untuk file sumber hello_aurora.cpp.

```
#include <aws/core/Aws.h>
#include <aws/rds/RDSClient.h>
#include <aws/rds/model/DescribeDBClustersRequest.h>
#include <iostream>

/*
 *  A "Hello Aurora" starter application which initializes an Amazon Relational
 Database Service (Amazon RDS) client
 *  and describes the Amazon Aurora (Aurora) clusters.
 *
 *  main function
 *
 *  Usage: 'hello_aurora'
 *
 */
int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";
```

```
        Aws::RDS::RDSClient rdsClient(clientConfig);

        Aws::String marker; // Used for pagination.
        std::vector<Aws::String> clusterIds;
        do {
            Aws::RDS::Model::DescribeDBClustersRequest request;

            Aws::RDS::Model::DescribeDBClustersOutcome outcome =
                    rdsClient.DescribeDBClusters(request);

            if (outcome.IsSuccess()) {
                for (auto &cluster: outcome.GetResult().GetDBClusters()) {
                    clusterIds.push_back(cluster.GetDBClusterIdentifier());
                }
                marker = outcome.GetResult().GetMarker();
            } else {
                result = 1;
                std::cerr << "Error with Aurora::GDescribeDBClusters. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                break;
            }
        } while (!marker.empty());

        std::cout << clusterIds.size() << " Aurora clusters found." << std::endl;
        for (auto &clusterId: clusterIds) {
            std::cout << "  clusterId " << clusterId << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return 0;
}
```

- Untuk detail API, lihat DescribeDBClusters di Referensi AWS SDK for C++ API.

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rds"
)

// main uses the AWS SDK for Go V2 to create an Amazon Aurora client and list up
 to 20
// DB clusters in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 sdkConfig, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 auroraClient := rds.NewFromConfig(sdkConfig)
 const maxClusters = 20
 fmt.Printf("Let's list up to %v DB clusters.\n", maxClusters)
 output, err := auroraClient.DescribeDBClusters(context.TODO(),
  &rds.DescribeDBClustersInput{MaxRecords: aws.Int32(maxClusters)})
 if err != nil {
  fmt.Printf("Couldn't list DB clusters: %v\n", err)
```

```
  return
 }
 if len(output.DBClusters) == 0 {
  fmt.Println("No DB clusters found.")
 } else {
  for _, cluster := range output.DBClusters {
   fmt.Printf("DB cluster %v has database %v.\n", *cluster.DBClusterIdentifier,
    *cluster.DatabaseName)
  }
 }
}
```

- Untuk detail API, lihat DescribeDBClusters di Referensi API AWS SDK for Go .

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rds.RdsClient;
import software.amazon.awssdk.services.rds.paginators.DescribeDBClustersIterable;

public class DescribeDbClusters {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        describeClusters(rdsClient);
        rdsClient.close();
    }

    public static void describeClusters(RdsClient rdsClient) {
```

```
        DescribeDBClustersIterable clustersIterable =
 rdsClient.describeDBClustersPaginator();
        clustersIterable.stream()
                .flatMap(r -> r.dbClusters().stream())
                .forEach(cluster -> System.out
                        .println("Database name: " + cluster.databaseName() + "
 Arn = " + cluster.dbClusterArn()));
    }
}
```

- Untuk detail API, lihat DescribeDBClusters di Referensi AWS SDK for Java 2.x API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_sdk_rds::Client;

#[derive(Debug)]
struct Error(String);
impl std::fmt::Display for Error {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{}", self.0)
    }
}
impl std::error::Error for Error {}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);
```

```
    let describe_db_clusters_output = client
        .describe_db_clusters()
        .send()
        .await
        .map_err(|e| Error(e.to_string()))?;
    println!(
        "Found {} clusters:",
        describe_db_clusters_output.db_clusters().len()
    );
    for cluster in describe_db_clusters_output.db_clusters() {
        let name = cluster.database_name().unwrap_or("Unknown");
        let engine = cluster.engine().unwrap_or("Unknown");
        let id = cluster.db_cluster_identifier().unwrap_or("Unknown");
        let class = cluster.db_cluster_instance_class().unwrap_or("Unknown");
        println!("\tDatabase: {name}",);
        println!("\t  Engine: {engine}",);
        println!("\t      ID: {id}",);
        println!("\tInstance: {class}",);
    }

    Ok(())
}
```

- Untuk detail API, lihat DescribeDBClusters di referensi AWS SDK for Rust API.

Contoh kode

- Tindakan untuk Aurora menggunakan SDK AWS
  - Buat cluster Aurora DB menggunakan SDK AWS
  - Buat grup parameter cluster Aurora DB menggunakan SDK AWS
  - Buat snapshot cluster Aurora DB menggunakan SDK AWS
  - Buat instans DB di cluster Aurora DB menggunakan SDK AWS
  - Menghapus klaster Aurora DB menggunakan SDK AWS
  - Menghapus grup parameter cluster Aurora DB menggunakan SDK AWS
  - Menghapus instans Aurora DB menggunakan SDK AWS
  - Jelaskan grup parameter cluster Aurora DB menggunakan SDK AWS
  - Jelaskan snapshot cluster Aurora DB menggunakan SDK AWS
  - Jelaskan cluster Aurora DB menggunakan SDK AWS

- Jelaskan instans Aurora DB menggunakan SDK AWS
- Jelaskan versi mesin database Aurora menggunakan SDK AWS
- Jelaskan opsi untuk instans Aurora DB menggunakan SDK AWS
- Jelaskan parameter dari grup parameter cluster Aurora DB menggunakan SDK AWS
- Perbarui parameter dalam grup parameter cluster Aurora DB menggunakan SDK AWS
- Skenario untuk Aurora menggunakan SDK AWS
  - Memulai cluster Aurora DB menggunakan SDK AWS
- Contoh lintas layanan untuk AWS Aurora menggunakan SDK
  - Membuat API REST pustaka peminjaman
  - Buat pelacak butir kerja Aurora Nirserver

Tindakan untuk Aurora menggunakan SDK AWS

Contoh kode berikut menunjukkan cara melakukan tindakan Aurora individual dengan AWS SDK. Kutipan ini memanggil Aurora API dan merupakan kutipan kode dari program yang lebih besar yang harus dijalankan dalam konteks. Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan instruksi untuk mengatur dan menjalankan kode.

Contoh berikut hanya mencakup tindakan yang paling umum digunakan. Untuk daftar lengkapnya, lihat Referensi Amazon Aurora API.

Contoh-contoh

- Buat cluster Aurora DB menggunakan SDK AWS
- Buat grup parameter cluster Aurora DB menggunakan SDK AWS
- Buat snapshot cluster Aurora DB menggunakan SDK AWS
- Buat instans DB di cluster Aurora DB menggunakan SDK AWS
- Menghapus klaster Aurora DB menggunakan SDK AWS
- Menghapus grup parameter cluster Aurora DB menggunakan SDK AWS
- Menghapus instans Aurora DB menggunakan SDK AWS
- Jelaskan grup parameter cluster Aurora DB menggunakan SDK AWS
- Jelaskan snapshot cluster Aurora DB menggunakan SDK AWS
- Jelaskan cluster Aurora DB menggunakan SDK AWS

- Jelaskan instans Aurora DB menggunakan SDK AWS
- Jelaskan versi mesin database Aurora menggunakan SDK AWS
- Jelaskan opsi untuk instans Aurora DB menggunakan SDK AWS
- Jelaskan parameter dari grup parameter cluster Aurora DB menggunakan SDK AWS
- Perbarui parameter dalam grup parameter cluster Aurora DB menggunakan SDK AWS

Buat cluster Aurora DB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara membuat klaster Aurora DB.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/// <summary>
/// Create a new cluster and database.
/// </summary>
/// <param name="dbName">The name of the new database.</param>
/// <param name="clusterIdentifier">The identifier of the cluster.</param>
/// <param name="parameterGroupName">The name of the parameter group.</param>
/// <param name="dbEngine">The engine to use for the new cluster.</param>
/// <param name="dbEngineVersion">The version of the engine to use.</param>
/// <param name="adminName">The admin username.</param>
/// <param name="adminPassword">The primary admin password.</param>
/// <returns>The cluster object.</returns>
public async Task<DBCluster> CreateDBClusterWithAdminAsync(
    string dbName,
```

```
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }
```

- Untuk detail API, lihat CreateDBCluster di Referensi API AWS SDK for .NET .

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region (overrides config file).
    // clientConfig.region = "us-east-1";

Aws::RDS::RDSClient client(clientConfig);

    Aws::RDS::Model::CreateDBClusterRequest request;
```

```
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetEngine(engineName);
        request.SetEngineVersion(engineVersionName);
        request.SetMasterUsername(administratorName);
        request.SetMasterUserPassword(administratorPassword);

        Aws::RDS::Model::CreateDBClusterOutcome outcome =
                client.CreateDBCluster(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBCluster. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }
```

- Untuk detail API, lihat CreateDBCluster di Referensi API AWS SDK for C++ .

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(clusterName string,
 parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(context.TODO(),
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}
```

- Untuk detail API, lihat CreateDBCluster di Referensi API AWS SDK for Go .

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static String createDBCluster(RdsClient rdsClient, String
dbParameterGroupFamily, String dbName,
           String dbClusterIdentifier, String userName, String password) {
       try {
           CreateDbClusterRequest clusterRequest =
CreateDbClusterRequest.builder()
                   .databaseName(dbName)
                   .dbClusterIdentifier(dbClusterIdentifier)
                   .dbClusterParameterGroupName(dbParameterGroupFamily)
                   .engine("aurora-mysql")
                   .masterUsername(userName)
                   .masterUserPassword(password)
                   .build();

           CreateDbClusterResponse response =
rdsClient.createDBCluster(clusterRequest);
           return response.dbCluster().dbClusterArn();

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
       return "";
   }
```

- Untuk detail API, lihat CreateDBCluster di Referensi API AWS SDK for Java 2.x .

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun createDBCluster(dbParameterGroupFamilyVal: String?, dbName: String?,
 dbClusterIdentifierVal: String?, userName: String?, password: String?): String?
 {
```

```
    val clusterRequest = CreateDbClusterRequest {
        databaseName = dbName
        dbClusterIdentifier = dbClusterIdentifierVal
        dbClusterParameterGroupName = dbParameterGroupFamilyVal
        engine = "aurora-mysql"
        masterUsername = userName
        masterUserPassword = password
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
    }
}
```

- Untuk detail API, lihat CreateDBCluster di Referensi API AWS SDK for Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
```

```
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_db_cluster(
        self,
        cluster_name,
        parameter_group_name,
        db_name,
        db_engine,
        db_engine_version,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB cluster that is configured to use the specified parameter
group.
        The newly created DB cluster contains a database that uses the specified
engine and
        engine version.

        :param cluster_name: The name of the DB cluster to create.
        :param parameter_group_name: The name of the parameter group to associate
with
                                     the DB cluster.
        :param db_name: The name of the database to create.
        :param db_engine: The database engine of the database that is created,
such as MySql.
        :param db_engine_version: The version of the database engine.
        :param admin_name: The user name of the database administrator.
        :param admin_password: The password of the database administrator.
        :return: The newly created DB cluster.
        """
        try:
            response = self.rds_client.create_db_cluster(
                DatabaseName=db_name,
                DBClusterIdentifier=cluster_name,
                DBClusterParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
```

```
            cluster = response["DBCluster"]
        except ClientError as err:
            logger.error(
                "Couldn't create database %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return cluster
```

- Untuk detail API, lihat CreateDBCluster di Referensi API AWS SDK for Python (Boto3).

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
```

```
        if self.password.is_none() {
            return Err(ScenarioError::with(
                "Must set Secret Password before starting a cluster",
            ));
        }
        let create_db_cluster = self
            .rds
            .create_db_cluster(
                DB_CLUSTER_IDENTIFIER,
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
```

```
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
```

```
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
```

```
    }

    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
        username: &str,
        password: SecretString,
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });
```

```
    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
```

```
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
```

```
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}
```

```
#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
```

```
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
```

```
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
```

```
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat CreateDBCluster di Referensi API AWS SDK for Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Buat grup parameter cluster Aurora DB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara membuat grup parameter klaster DB Aurora.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
 _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }
```

- Untuk detail API, lihat CreateDB ClusterParameterGroup di AWS SDK for .NET Referensi API.

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- Untuk detail API, lihat CreateDB ClusterParameterGroup di AWS SDK for C++ Referensi API.

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 parameterGroupName string, parameterGroupFamily string, description string) (
 *types.DBClusterParameterGroup, error) {

 output, err :=
 clusters.AuroraClient.CreateDBClusterParameterGroup(context.TODO(),
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}
```

- Untuk detail API, lihat CreateDB ClusterParameterGroup di AWS SDK for Go Referensi API.

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void createDBClusterParameterGroup(RdsClient rdsClient, String
dbClusterGroupName,
           String dbParameterGroupFamily) {
       try {
           CreateDbClusterParameterGroupRequest groupRequest =
CreateDbClusterParameterGroupRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .dbParameterGroupFamily(dbParameterGroupFamily)
                   .description("Created by using the AWS SDK for Java")
                   .build();

           CreateDbClusterParameterGroupResponse response =
rdsClient.createDBClusterParameterGroup(groupRequest);
           System.out.println("The group name is " +
response.dbClusterParameterGroup().dbClusterParameterGroupName());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Untuk detail API, lihat CreateDB ClusterParameterGroup di AWS SDK for Java 2.x Referensi API.

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun createDBClusterParameterGroup(dbClusterGroupNameVal: String?,
 dbParameterGroupFamilyVal: String?) {
    val groupRequest = CreateDbClusterParameterGroupRequest {
        dbClusterParameterGroupName = dbClusterGroupNameVal
        dbParameterGroupFamily = dbParameterGroupFamilyVal
        description = "Created by using the AWS SDK for Kotlin"
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}
```

- Untuk detail API, lihat CreateDB ClusterParameterGroup di AWS SDK untuk referensi API Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
```

```
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB cluster parameter group that is based on the specified
parameter group
        family.

        :param parameter_group_name: The name of the newly created parameter
group.
        :param parameter_group_family: The family that is used as the basis of
the new
                                       parameter group.
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
```

```
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- Untuk detail API, lihat CreateDB ClusterParameterGroup di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
   pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
        self.engine_family = Some(engine.to_string());
        self.engine_version = Some(version.to_string());
        let create_db_cluster_parameter_group = self
            .rds
            .create_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
                engine,
            )
            .await;

        match create_db_cluster_parameter_group {
            Ok(CreateDbClusterParameterGroupOutput {
                db_cluster_parameter_group: None,
                ..
```

```
            }) => {
                return Err(ScenarioError::with(
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to
 do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
                    ));
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }

        Ok(())
    }

    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
    ) -> Result<CreateDbClusterParameterGroupOutput,
 SdkError<CreateDBClusterParameterGroupError>>
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;
```

```
    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(

 CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}
```

- Untuk detail API, lihat CreateDB ClusterParameterGroup di AWS SDK untuk referensi Rust API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Buat snapshot cluster Aurora DB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara membuat snapshot klaster DB Aurora.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }
```

- Untuk detail API, lihat CreateDB ClusterSnapshot di AWS SDK for .NET Referensi API.

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
            request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                    client.CreateDBClusterSnapshot(request);

            if (outcome.IsSuccess()) {
                std::cout << "Snapshot creation has started."
                          << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
```

- Untuk detail API, lihat CreateDB ClusterSnapshot di AWS SDK for C++ Referensi API.

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(clusterName string,
 snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(context.TODO(),
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
}
```

- Untuk detail API, lihat CreateDB ClusterSnapshot di AWS SDK for Go Referensi API.

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void createDBClusterSnapshot(RdsClient rdsClient, String
dbInstanceClusterIdentifier,
           String dbSnapshotIdentifier) {
       try {
           CreateDbClusterSnapshotRequest snapshotRequest =
CreateDbClusterSnapshotRequest.builder()
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                   .build();

           CreateDbClusterSnapshotResponse response =
rdsClient.createDBClusterSnapshot(snapshotRequest);
           System.out.println("The Snapshot ARN is " +
response.dbClusterSnapshot().dbClusterSnapshotArn());

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Untuk detail API, lihat CreateDB ClusterSnapshot di AWS SDK for Java 2.x Referensi API.

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun createDBClusterSnapshot(dbInstanceClusterIdentifier: String?,
 dbSnapshotIdentifier: String?) {
    val snapshotRequest = CreateDbClusterSnapshotRequest {
        dbClusterIdentifier = dbInstanceClusterIdentifier
        dbClusterSnapshotIdentifier = dbSnapshotIdentifier
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}
```

- Untuk detail API, lihat CreateDB ClusterSnapshot di AWS SDK untuk referensi API Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""
```

```
    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
DBClusterIdentifier=cluster_id
            )
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- Untuk detail API, lihat CreateDB ClusterSnapshot di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
```

```
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
```

```
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
```

```
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
SdkError<CreateDBClusterSnapshotError>> {
        self.inner
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
```

```
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
```

```
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
```

```
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
```

```
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
```

```
 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
```

```
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
```

```
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

 .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat CreateDB ClusterSnapshot di AWS SDK untuk referensi Rust API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Buat instans DB di cluster Aurora DB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara membuat instans DB di klaster DB Aurora.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
   /// with a particular set of properties. Use the action
DescribeDBInstancesAsync
   /// to determine when the DB instance is ready to use.
   /// </summary>
   /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
   /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
   /// <param name="dbEngine">The engine for the DB instance.</param>
   /// <param name="dbEngineVersion">Version for the DB instance.</param>
   /// <param name="instanceClass">Class for the DB instance.</param>
   /// <returns>DB instance object.</returns>
```

```
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }
```

- Lihat detail API di CreateDBInstance dalam Referensi API AWS SDK for .NET .

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::CreateDBInstanceRequest request;
```

```
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
 "",
                             client);
            return false;
        }
```

- Lihat detail API di CreateDBInstance dalam Referensi API AWS SDK for C++ .

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(clusterName string,
 instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(context.TODO(),
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}
```

- Lihat detail API di CreateDBInstance dalam Referensi API AWS SDK for Go .

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    public static String createDBInstanceCluster(RdsClient rdsClient,
            String dbInstanceIdentifier,
            String dbInstanceClusterIdentifier,
            String instanceClass) {
        try {
            CreateDbInstanceRequest instanceRequest =
CreateDbInstanceRequest.builder()
```

```
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .engine("aurora-mysql")
                    .dbInstanceClass(instanceClass)
                    .build();

            CreateDbInstanceResponse response =
 rdsClient.createDBInstance(instanceRequest);
            System.out.print("The status is " +
 response.dbInstance().dbInstanceStatus());
            return response.dbInstance().dbInstanceArn();

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }
```

- Lihat detail API di CreateDBInstance dalam Referensi API AWS SDK for Java 2.x .

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun createDBInstanceCluster(dbInstanceIdentifierVal: String?,
 dbInstanceClusterIdentifierVal: String?, instanceClassVal: String?): String? {
    val instanceRequest = CreateDbInstanceRequest {
        dbInstanceIdentifier = dbInstanceIdentifierVal
        dbClusterIdentifier = dbInstanceClusterIdentifierVal
        engine = "aurora-mysql"
        dbInstanceClass = instanceClassVal
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
```

```
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}
```

- Lihat detail API di CreateDBInstance dalam Referensi API AWS SDK for Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
```

```
        Creates a database instance in an existing DB cluster. The first database
that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
        :param cluster_id: The ID of the DB cluster where the DB instance is
created.
        :param db_engine: The database engine of a database to create in the DB
instance.
                          This must be compatible with the configured parameter
group
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- Untuk detail API, lihat CreateDBInstance di Referensi API AWS SDK for Python (Boto3).

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
```

```
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
```

```
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));
```

```
            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
            .engine(engine)
            .send()
```

```
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
```

```
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
```

```
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
```

```
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
```

```
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
```

```
.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
```

```
 .db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat CreateDBInstance di Referensi API AWS SDK for Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Menghapus klaster Aurora DB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara menghapus klaster Aurora DB.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Delete a particular DB cluster.
   /// </summary>
   /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
   /// <returns>DB cluster object.</returns>
   public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
   {
       var response = await _amazonRDS.DeleteDBClusterAsync(
           new DeleteDBClusterRequest()
           {
               DBClusterIdentifier = dbClusterIdentifier,
               SkipFinalSnapshot = true
           });
```

```
        return response.DBCluster;
    }
```

- Untuk detail API, lihat DeleteDBCluster di Referensi API AWS SDK for .NET .

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
parameter group."
                        << std::endl;
                std::cout << "This may take a while." << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
                          << outcome.GetError().GetMessage()
```

```
                        << std::endl;
                result = false;
            }
```

- Untuk detail API, lihat DeleteDBCluster di Referensi API AWS SDK for C++ .

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(clusterName string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(context.TODO(),
 &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   true,
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}
```

- Untuk detail API, lihat DeleteDBCluster di Referensi API AWS SDK for Go .

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void deleteCluster(RdsClient rdsClient, String
dbInstanceClusterIdentifier) {
       try {
           DeleteDbClusterRequest deleteDbClusterRequest =
DeleteDbClusterRequest.builder()
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .skipFinalSnapshot(true)
                   .build();

           rdsClient.deleteDBCluster(deleteDbClusterRequest);
           System.out.println(dbInstanceClusterIdentifier + " was deleted!");

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Untuk detail API, lihat DeleteDBCluster di Referensi API AWS SDK for Java 2.x .

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest = DeleteDbClusterRequest {
        dbClusterIdentifier = dbInstanceClusterIdentifier
        skipFinalSnapshot = true
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
    }
}
```

- Untuk detail API, lihat DeleteDBCluster di Referensi API AWS SDK for Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
```

```
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
            raise
```

- Untuk detail API, lihat DeleteDBCluster di Referensi API AWS SDK for Python (Boto3).

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
```

```
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
```

```
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
```

```
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
```

```
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
```

```
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
```

```
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
```

```
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat DeleteDBCluster di Referensi API AWS SDK for Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Menghapus grup parameter cluster Aurora DB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara menghapus grup parameter klaster Aurora DB.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await
_amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Untuk detail API, lihat DeleteDB ClusterParameterGroup di Referensi AWS SDK for .NET API.

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    Aws::Client::ClientConfiguration clientConfig;
    // Optional: Set to the AWS Region (overrides config file).
    // clientConfig.region = "us-east-1";

Aws::RDS::RDSClient client(clientConfig);

    Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
    request.SetDBClusterParameterGroupName(parameterGroupName);

    Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
            client.DeleteDBClusterParameterGroup(request);

    if (outcome.IsSuccess()) {
        std::cout << "The DB parameter group was successfully deleted."
                  << std::endl;
    }
    else {
        std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
        result = false;
    }
```

- Untuk detail API, lihat DeleteDB ClusterParameterGroup di Referensi AWS SDK for C++ API.

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(parameterGroupName string) error
 {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(context.TODO(),
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- Untuk detail API, lihat DeleteDB ClusterParameterGroup di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
           throws InterruptedException {
       try {
           boolean isDataDel = false;
           boolean didFind;
           String instanceARN;

           // Make sure that the database has been deleted.
           while (!isDataDel) {
               DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
               List<DBInstance> instanceList = response.dbInstances();
               int listSize = instanceList.size();
               didFind = false;
               int index = 1;
               for (DBInstance instance : instanceList) {
                   instanceARN = instance.dbInstanceArn();
                   if (instanceARN.compareTo(clusterDBARN) == 0) {
                       System.out.println(clusterDBARN + " still exists");
                       didFind = true;
                   }
                   if ((index == listSize) && (!didFind)) {
                       // Went through the entire list and did not find the
database ARN.
                       isDataDel = true;
                   }
                   Thread.sleep(sleepTime * 1000);
                   index++;
               }
           }
```

```
            DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
 DeleteDbClusterParameterGroupRequest
                    .builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .build();


 rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
            System.out.println(dbClusterGroupName + " was deleted.");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat DeleteDB ClusterParameterGroup di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
@Throws(InterruptedException::class)
suspend fun deleteDBClusterGroup(dbClusterGroupName: String, clusterDBARN:
 String) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
```

```
            val instanceList = response.dbInstances
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
                    index++
                }
            }
        }
        val clusterParameterGroupRequest = DeleteDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupName
        }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}
```

- Untuk detail API, lihat DeleteDB ClusterParameterGroup di AWS SDK untuk referensi API Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
```

```
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- Untuk detail API, lihat DeleteDB ClusterParameterGroup di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
```

```
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
```

```
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
```

```
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_cluster_parameter_group(
        &self,
```

```
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
 SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
```

```
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Cluster
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
```

```
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );
```

```
    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat DeleteDB ClusterParameterGroup di AWS SDK untuk referensi Rust API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Menghapus instans Aurora DB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara menghapus instans DB Aurora.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
```

- Lihat detail API di DeleteDBInstance dalam Referensi API AWS SDK for .NET .

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);

            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
                        << "Waiting for DB instance to delete before deleting the
parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
```

- Lihat detail API di DeleteDBInstance dalam Referensi API AWS SDK for C++ .

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(instanceName string) error {
 _, err := clusters.AuroraClient.DeleteDBInstance(context.TODO(),
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      true,
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

- Lihat detail API di DeleteDBInstance dalam Referensi API AWS SDK for Go .

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void deleteDatabaseInstance(RdsClient rdsClient, String
dbInstanceIdentifier) {
       try {
```

```
            DeleteDbInstanceRequest deleteDbInstanceRequest =
 DeleteDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .deleteAutomatedBackups(true)
                    .skipFinalSnapshot(true)
                    .build();

            DeleteDbInstanceResponse response =
 rdsClient.deleteDBInstance(deleteDbInstanceRequest);
            System.out.println("The status of the database is " +
 response.dbInstance().dbInstanceStatus());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Lihat detail API di DeleteDBInstance dalam Referensi API AWS SDK for Java 2.x .

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest = DeleteDbInstanceRequest {
        dbInstanceIdentifier = dbInstanceIdentifierVal
        deleteAutomatedBackups = true
        skipFinalSnapshot = true
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
```

```
    }
}
```

- Lihat detail API di DeleteDBInstance dalam Referensi API AWS SDK for Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
```

```
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- Untuk detail API, lihat DeleteDBInstance di Referensi API AWS SDK for Python (Boto3).

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
```

```
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
```

```
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
```

```
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
```

```
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));
```

```
    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });
```

```
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
```

```
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat DeleteDBInstance di Referensi API AWS SDK for Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Jelaskan grup parameter cluster Aurora DB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara menjelaskan grup parameter klaster Aurora DB.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }
```

- Untuk detail API, lihat DescribeDB ClusterParameterGroups di Referensi AWS SDK for .NET API.

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
                client.DescribeDBClusterParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }

        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- Untuk detail API, lihat DescribeDB ClusterParameterGroups di Referensi AWS SDK for C++ API.

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  context.TODO(), &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}
```

- Untuk detail API, lihat DescribeDB ClusterParameterGroups di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
       try {
           DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .maxRecords(20)
                   .build();

           List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                   .dbClusterParameterGroups();
           for (DBClusterParameterGroup group : groups) {
               System.out.println("The group name is " +
group.dbClusterParameterGroupName());
               System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
           }

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Untuk detail API, lihat DescribeDB ClusterParameterGroups di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest = DescribeDbClusterParameterGroupsRequest {
        dbClusterParameterGroupName = dbClusterGroupName
        maxRecords = 20
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}
```

- Untuk detail API, lihat DescribeDB ClusterParameterGroups di AWS SDK untuk referensi API Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
```

```
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
        Gets a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The requested parameter group.
        """
        try:
            response = self.rds_client.describe_db_cluster_parameter_groups(
                DBClusterParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBClusterParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return parameter_group
```

- Untuk detail API, lihat DescribeDB ClusterParameterGroups di AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Jelaskan snapshot cluster Aurora DB menggunakan SDK AWS

Contoh kode berikut menunjukkan bagaimana mendeskripsikan snapshot klaster DB Aurora.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Return a list of DB snapshots for a particular DB cluster.
   /// </summary>
   /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
   /// <returns>List of DB snapshots.</returns>
   public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
   {
       var results = new List<DBClusterSnapshot>();

       DescribeDBClusterSnapshotsResponse response;
```

```
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- Untuk detail API, lihat DescribeDB ClusterSnapshots di Referensi AWS SDK for .NET API.

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
```

```
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
 client);
                return false;
            }
```

- Untuk detail API, lihat DescribeDB ClusterSnapshots di Referensi AWS SDK for C++ API.

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(snapshotName string)
 (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(context.TODO(),
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
```

```
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}
```

- Untuk detail API, lihat DescribeDB ClusterSnapshots di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void waitForSnapshotReady(RdsClient rdsClient, String
dbSnapshotIdentifier,
           String dbInstanceClusterIdentifier) {
       try {
           boolean snapshotReady = false;
           String snapshotReadyStr;
           System.out.println("Waiting for the snapshot to become available.");

           DescribeDbClusterSnapshotsRequest snapshotsRequest =
DescribeDbClusterSnapshotsRequest.builder()
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .build();

           while (!snapshotReady) {
               DescribeDbClusterSnapshotsResponse response =
rdsClient.describeDBClusterSnapshots(snapshotsRequest);
               List<DBClusterSnapshot> snapshotList =
response.dbClusterSnapshots();
               for (DBClusterSnapshot snapshot : snapshotList) {
                   snapshotReadyStr = snapshot.status();
                   if (snapshotReadyStr.contains("available")) {
                       snapshotReady = true;
```

```
                    } else {
                        System.out.println(".");
                        Thread.sleep(sleepTime * 5000);
                    }
                }
            }

            System.out.println("The Snapshot is available!");

        } catch (RdsException | InterruptedException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat DescribeDB ClusterSnapshots di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun waitSnapshotReady(dbSnapshotIdentifier: String?,
 dbInstanceClusterIdentifier: String?) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest = DescribeDbClusterSnapshotsRequest {
        dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        dbClusterIdentifier = dbInstanceClusterIdentifier
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
```

```
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        println(".")
                        delay(slTime * 5000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}
```

- Untuk detail API, lihat DescribeDB ClusterSnapshots di AWS SDK untuk referensi API Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
```

```
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
            logger.error(
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot
```

- Untuk detail API, lihat DescribeDB ClusterSnapshots di AWS SDK for Python (Boto3) Referensi API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Jelaskan cluster Aurora DB menggunakan SDK AWS

Contoh kode berikut menunjukkan bagaimana mendeskripsikan klaster DB Aurora.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Returns a list of DB clusters.
   /// </summary>
   /// <param name="dbInstanceIdentifier">Optional name of a specific DB
cluster.</param>
   /// <returns>List of DB clusters.</returns>
   public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
   {
       var results = new List<DBCluster>();

       DescribeDBClustersResponse response;
       DescribeDBClustersRequest request = new DescribeDBClustersRequest
       {
           DBClusterIdentifier = dbClusterIdentifier
       };
       // Get the full list if there are multiple pages.
       do
       {
           response = await _amazonRDS.DescribeDBClustersAsync(request);
           results.AddRange(response.DBClusters);
           request.Marker = response.Marker;
```

```
        }
        while (response.Marker is not null);
        return results;
    }
```

- Untuk detail API, lihat DescribeDBClusters di Referensi AWS SDK for .NET API.

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
            client.DescribeDBClusters(request);

    bool result = true;
```

```
    if (outcome.IsSuccess()) {
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }

    return result;

}
```

- Untuk detail API, lihat DescribeDBClusters di Referensi API AWS SDK for C++ .

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetDbCluster gets data about an Aurora DB cluster.
```

```
func (clusters *DbClusters) GetDbCluster(clusterName string) (*types.DBCluster,
 error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(context.TODO(),
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}
```

- Untuk detail API, lihat DescribeDBClusters di Referensi API AWS SDK for Go .

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
 dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
 DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
```

```
                        .build();
            } else {
                dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                        .dbClusterParameterGroupName(dbCLusterGroupName)
                        .source("user")
                        .build();
            }

            DescribeDbClusterParametersResponse response = rdsClient
                    .describeDBClusterParameters(dbParameterGroupsRequest);
            List<Parameter> dbParameters = response.parameters();
            String paraName;
            for (Parameter para : dbParameters) {
                // Only print out information about either auto_increment_offset
or
                // auto_increment_increment.
                paraName = para.parameterName();
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat DescribeDBClusters di Referensi API AWS SDK for Java 2.x .

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun describeDbClusterParameters(dbCLusterGroupName: String?, flag: Int) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest = if (flag == 0) {
        DescribeDbClusterParametersRequest {
            dbClusterParameterGroupName = dbCLusterGroupName
        }
    } else {
        DescribeDbClusterParametersRequest {
            dbClusterParameterGroupName = dbCLusterGroupName
            source = "user"
        }
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
```

```
        }
    }
}
```

- Untuk detail API, lihat DescribeDBClusters di Referensi API AWS SDK for Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_db_cluster(self, cluster_name):
        """
        Gets data about an Aurora DB cluster.

        :param cluster_name: The name of the DB cluster to retrieve.
        :return: The retrieved DB cluster.
```

```
        """
        try:
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why:
%s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return cluster
```

- Untuk detail API, lihat DescribeDBClusters di Referensi API AWS SDK for Python (Boto3).

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
   // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
   // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
```

```
   // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

   // Create a database instance in the cluster.
   // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
   pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
       if self.password.is_none() {
           return Err(ScenarioError::with(
               "Must set Secret Password before starting a cluster",
           ));
       }
       let create_db_cluster = self
           .rds
           .create_db_cluster(
               DB_CLUSTER_IDENTIFIER,
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_ENGINE,
               self.engine_version.as_deref().expect("engine version"),
               self.username.as_deref().expect("username"),
               self.password
                   .replace(SecretString::new("".to_string()))
                   .expect("password"),
           )
           .await;
       if let Err(err) = create_db_cluster {
           return Err(ScenarioError::new(
               "Failed to create DB Cluster with cluster group",
               &err,
           ));
       }

       self.db_cluster_identifier = create_db_cluster
           .unwrap()
           .db_cluster
           .and_then(|c| c.db_cluster_identifier);

       if self.db_cluster_identifier.is_none() {
           return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
       }
```

```
        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
```

```
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));

            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
```

```
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }

    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }

#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });
```

```
    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });
```

```
    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
```

```
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}
```

```
#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
```

```
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });
```

```
    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

.db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });
```

```
    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
```

- Untuk detail API, lihat DescribeDBClusters di referensi AWS SDK for Rust API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Jelaskan instans Aurora DB menggunakan SDK AWS

Contoh kode berikut ini menunjukkan cara mendeskripsikan instans Aurora.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }
```

- Lihat detail API di DescribeDBInstances dalam Referensi API AWS SDK for .NET .

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                        Aws::RDS::Model::DBInstance
 &instanceResult,
                                        const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
```

```
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
    }

    return result;
}
```

- Lihat detail API di DescribeDBInstances dalam Referensi API AWS SDK for C++ .

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(instanceName string) (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(context.TODO(),
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
```

```
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}
```

- Lihat detail API di DescribeDBInstances dalam Referensi API AWS SDK for Go .

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Waits until the database instance is available.
   public static void waitForInstanceReady(RdsClient rdsClient, String
dbClusterIdentifier) {
       boolean instanceReady = false;
       String instanceReadyStr;
       System.out.println("Waiting for instance to become available.");
       try {
           DescribeDbClustersRequest instanceRequest =
DescribeDbClustersRequest.builder()
                   .dbClusterIdentifier(dbClusterIdentifier)
                   .build();

           while (!instanceReady) {
               DescribeDbClustersResponse response =
rdsClient.describeDBClusters(instanceRequest);
               List<DBCluster> clusterList = response.dbClusters();
```

```
                for (DBCluster cluster : clusterList) {
                    instanceReadyStr = cluster.status();
                    if (instanceReadyStr.contains("available")) {
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database cluster is available!");

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

- Lihat detail API di DescribeDBInstances dalam Referensi API AWS SDK for Java 2.x .

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest = DescribeDbInstancesRequest {
        dbInstanceIdentifier = dbInstanceIdentifierVal
    }

    var endpoint = ""
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
```

```
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database instance is available! The connection endpoint is
 $endpoint")
}
```

- Lihat detail API di DescribeDBInstances dalam Referensi API AWS SDK for Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
```

```
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return db_inst
```

- Untuk detail API, lihat DescribeDBInstances di Referensi API AWS SDK for Python (Boto3).

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
```

```
                let db_instances = describe_db_instances
                    .unwrap()
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
```

```
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
```

```
        let delete_db_cluster_parameter_group = self
            .rds
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }

    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }

#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
```

```
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));
```

```
    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
```

```
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
```

```
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
Describe Cluster
    tokio::time::resume();
```

```
    let _ = assertions.await;
}
```

- Untuk detail API, lihat DescribeDBInstances di Referensi API AWS SDK for Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Jelaskan versi mesin database Aurora menggunakan SDK AWS

Contoh-contoh kode berikut menunjukkan cara mendeskripsikan versi mesin basis data Aurora.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   /// <summary>
   /// Get a list of DB engine versions for a particular DB engine.
   /// </summary>
   /// <param name="engine">The name of the engine.</param>
   /// <param name="parameterGroupFamily">Optional parameter group family
name.</param>
   /// <returns>A list of DBEngineVersions.</returns>
   public async Task<List<DBEngineVersion>>
DescribeDBEngineVersionsForEngineAsync(string engine,
       string? parameterGroupFamily = null)
```

```
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }
```

- Untuk detail API, lihat DescribeDB EngineVersions di Referensi AWS SDK for .NET API.

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
```

```
                                        const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                        const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
            client.DescribeDBEngineVersions(request);

    if (outcome.IsSuccess()) {
        engineVersionsResult = outcome.GetResult().GetDBEngineVersions();
    }
    else {
        std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- Untuk detail API, lihat DescribeDB EngineVersions di Referensi AWS SDK for C++ API.

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(engine string, parameterGroupFamily
 string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(context.TODO(),
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}
```

- Untuk detail API, lihat DescribeDB EngineVersions di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
                    .defaultOnly(true)
```

```
                    .maxRecords(20)
                    .build();

            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
engineOb.engine());
                System.out.println("The version number of the database engine " +
engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat DescribeDB EngineVersions di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest = DescribeDbEngineVersionsRequest {
        dbParameterGroupFamily = dbParameterGroupFamilyVal
        engine = "aurora-mysql"
```

```
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}
```

- Untuk detail API, lihat DescribeDB EngineVersions di AWS SDK untuk referensi API Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)
```

```
    def get_engine_versions(self, engine, parameter_group_family=None):
        """
        Gets database engine versions that are available for the specified engine
        and parameter group family.

        :param engine: The database engine to look up.
        :param parameter_group_family: When specified, restricts the returned
list of
                                       engine versions to those that are
compatible with
                                       this parameter group family.
        :return: The list of database engine versions.
        """
        try:
            kwargs = {"Engine": engine}
            if parameter_group_family is not None:
                kwargs["DBParameterGroupFamily"] = parameter_group_family
            response = self.rds_client.describe_db_engine_versions(**kwargs)
            versions = response["DBEngineVersions"]
        except ClientError as err:
            logger.error(
                "Couldn't get engine versions for %s. Here's why: %s: %s",
                engine,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return versions
```

- Untuk detail API, lihat DescribeDB EngineVersions di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Get available engine families for Aurora MySql.
rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
   pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
ScenarioError> {
       let describe_db_engine_versions =
self.rds.describe_db_engine_versions(DB_ENGINE).await;
       trace!(versions=?describe_db_engine_versions, "full list of versions");

       if let Err(err) = describe_db_engine_versions {
           return Err(ScenarioError::new(
               "Failed to retrieve DB Engine Versions",
               &err,
           ));
       };

       let version_count = describe_db_engine_versions
           .as_ref()
           .map(|o| o.db_engine_versions().len())
           .unwrap_or_default();
       info!(version_count, "got list of versions");

       // Create a map of engine families to their available versions.
       let mut versions = HashMap::<String, Vec<String>>::new();
       describe_db_engine_versions
           .unwrap()
           .db_engine_versions()
           .iter()
           .filter_map(
               |v| match (&v.db_parameter_group_family, &v.engine_version) {
                   (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
```

```
                    _ => None,
                },
            )
            .for_each(|(family, version)|
 versions.entry(family).or_default().push(version));

        Ok(versions)
    }

    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
    ) -> Result<DescribeDbEngineVersionsOutput,
 SdkError<DescribeDBEngineVersionsError>> {
        self.inner
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
```

```
                        .db_parameter_group_family("f2")
                        .engine_version("f2a")
                        .build(),
                )
                .db_engine_versions(DbEngineVersion::builder().build())
                .build())
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

    assert_eq!(
        versions_map,
        Ok(HashMap::from([
            ("f1".into(), vec!["f1a".into(), "f1b".into()]),
            ("f2".into(), vec!["f2a".into()])
        ]))
    );
}

#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
```

```
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}
```

- Untuk detail API, lihat DescribeDB EngineVersions di AWS SDK untuk referensi API Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Jelaskan opsi untuk instans Aurora DB menggunakan SDK AWS

Contoh kode berikut ini menunjukkan cara mendeskripsikan opsi untuk instans DB Aurora.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
```

```
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }
```

- Untuk detail API, lihat DescribeOrderableDB InstanceOptions di Referensi AWS SDK for .NET API.

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);
```

```
//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
                                           Aws::String &dbInstanceClass,
                                           const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (std::find(instanceClasses.begin(), instanceClasses.end(),
 instanceClass) == instanceClasses.end()) {
                    instanceClasses.push_back(instanceClass);
                }
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
```

```
            std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
 "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available DB instance classes for your database engine
 are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}
```

- Untuk detail API, lihat DescribeOrderableDB InstanceOptions di Referensi AWS SDK for C++ + API.

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(engine string, engineVersion
 string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
 return instances, err
}
```

- Untuk detail API, lihat DescribeOrderableDB InstanceOptions di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void describeDBEngines(RdsClient rdsClient) {
       try {
           DescribeDbEngineVersionsRequest engineVersionsRequest =
DescribeDbEngineVersionsRequest.builder()
                   .engine("aurora-mysql")
                   .defaultOnly(true)
                   .maxRecords(20)
                   .build();

           DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(engineVersionsRequest);
           List<DBEngineVersion> engines = response.dbEngineVersions();

           // Get all DBEngineVersion objects.
           for (DBEngineVersion engineOb : engines) {
               System.out.println("The name of the DB parameter group family for
the database engine is "
                       + engineOb.dbParameterGroupFamily());
               System.out.println("The name of the database engine " +
engineOb.engine());
               System.out.println("The version number of the database engine " +
engineOb.engineVersion());
           }

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Untuk detail API, lihat DescribeOrderableDB InstanceOptions di Referensi AWS SDK for Java 2.x API.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_orderable_instances(self, db_engine, db_engine_version):
        """
        Gets DB instance options that can be used to create DB instances that are
        compatible with a set of specifications.

        :param db_engine: The database engine that must be supported by the DB
 instance.
        :param db_engine_version: The engine version that must be supported by
 the DB instance.
```

```
        :return: The list of DB instance options that can be used to create a
compatible DB instance.
        """
        try:
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts
```

- Untuk detail API, lihat DescribeOrderableDB InstanceOptions di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    pub async fn get_instance_classes(&self) -> Result<Vec<String>,
ScenarioError> {
        let describe_orderable_db_instance_options_items = self
            .rds
```

```
            .describe_orderable_db_instance_options(
                DB_ENGINE,
                self.engine_version
                    .as_ref()
                    .expect("engine version for db instance options")
                    .as_str(),
            )
            .await;

        describe_orderable_db_instance_options_items
            .map(|options| {
                options
                    .iter()
                    .map(|o|
 o.db_instance_class().unwrap_or_default().to_string())
                    .collect::<Vec<String>>()
            })
            .map_err(|err| ScenarioError::new("Could not get available instance
 classes", &err))
    }

    pub async fn describe_orderable_db_instance_options(
        &self,
        engine: &str,
        engine_version: &str,
    ) -> Result<Vec<OrderableDbInstanceOption>,
 SdkError<DescribeOrderableDBInstanceOptionsError>>
    {
        self.inner
            .describe_orderable_db_instance_options()
            .engine(engine)
            .engine_version(engine_version)
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
    }

#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
```

```
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .build(),
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
```

```
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(

 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}
```

- Untuk detail API, lihat referensi DescribeOrderableDB InstanceOptions di AWS SDK for Rust API.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Jelaskan parameter dari grup parameter cluster Aurora DB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara menjelaskan parameter dalam grup parameter klaster DB Aurora.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
_amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
```

```
        }
        while (response.Marker is not null);

        return paramList;
    }
```

- Untuk detail API, lihat DescribeDB ClusterParameters di Referensi AWS SDK for .NET API.

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);


//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBCLusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBCLusterParameters(const Aws::String
 &parameterGroupName,
                                            const Aws::String &namePrefix,
                                            const Aws::String &source,

 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
```

```
                                                const Aws::RDS::RDSClient &client) {
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}
```

- Untuk detail API, lihat DescribeDB ClusterParameters di Referensi AWS SDK for C++ API.

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(parameterGroupName string, source
 string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
```

```
}
```

- Untuk detail API, lihat DescribeDB ClusterParameters di Referensi AWS SDK for Go API.

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .source("user")
                       .build();
           }

           DescribeDbClusterParametersResponse response = rdsClient
                   .describeDBClusterParameters(dbParameterGroupsRequest);
           List<Parameter> dbParameters = response.parameters();
           String paraName;
           for (Parameter para : dbParameters) {
               // Only print out information about either auto_increment_offset
or
               // auto_increment_increment.
               paraName = para.parameterName();
```

```
                if ((paraName.compareTo("auto_increment_offset") == 0)
                        || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                    System.out.println("*** The parameter name is  " + paraName);
                    System.out.println("*** The parameter value is  " +
para.parameterValue());
                    System.out.println("*** The parameter data type is " +
para.dataType());
                    System.out.println("*** The parameter description is " +
para.description());
                    System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
                }
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

- Untuk detail API, lihat DescribeDB ClusterParameters di Referensi AWS SDK for Java 2.x API.

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
suspend fun describeDbClusterParameters(dbCLusterGroupName: String?, flag: Int) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest = if (flag == 0) {
        DescribeDbClusterParametersRequest {
            dbClusterParameterGroupName = dbCLusterGroupName
        }
```

```
    } else {
        DescribeDbClusterParametersRequest {
            dbClusterParameterGroupName = dbCLusterGroupName
            source = "user"
        }
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}
```

- Untuk detail API, lihat DescribeDB ClusterParameters di AWS SDK untuk referensi API Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
 filtered
                            to contain only parameters that start with this
 prefix.
        :param source: When specified, only parameters from this source are
 retrieved.
                       For example, a source of 'user' retrieves only parameters
 that
```

```
                        were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters
```

- Untuk detail API, lihat DescribeDB ClusterParameters di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Get the parameter group. rds.DescribeDbClusterParameterGroups
   // Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
   // Parse the ParameterName, Description, and AllowedValues values and display
them.
   pub async fn cluster_parameters(&self) ->
Result<Vec<AuroraScenarioParameter>, ScenarioError> {
       let parameters_output = self
           .rds
           .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
           .await;

       if let Err(err) = parameters_output {
           return Err(ScenarioError::new(
               format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
               &err,
           ));
       }

       let parameters = parameters_output
           .unwrap()
           .into_iter()
           .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
           .filter(|p|
FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
           .map(AuroraScenarioParameter::from)
           .collect::<Vec<_>>();

       Ok(parameters)
   }

   pub async fn describe_db_cluster_parameters(
       &self,
       name: &str,
   ) -> Result<Vec<DescribeDbClusterParametersOutput>,
SdkError<DescribeDBClusterParametersError>>
   {
       self.inner
           .describe_db_cluster_parameters()
           .db_cluster_parameter_group_name(name)
           .into_paginator()
```

```
            .send()
            .try_collect()
            .await
    }

#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
```

```
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}
```

- Untuk detail API, lihat DescribeDB ClusterParameters di AWS SDK untuk referensi API Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Perbarui parameter dalam grup parameter cluster Aurora DB menggunakan SDK AWS

Contoh kode berikut menunjukkan cara memperbarui parameter dalam grup parameter klaster DB Aurora.

Contoh-contoh tindakan adalah kutipan kode dari program yang lebih besar dan harus dijalankan di dalam konteks. Anda dapat melihat tindakan ini dalam konteks pada contoh kode berikut:

- Memulai dengan klaster DB

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string
 groupName, List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
 allowed values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
```

```
            DBClusterParameterGroupName = groupName,
        };

        var result = await
_amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }
```

- Untuk detail API, lihat ModifyDB ClusterParameterGroup di AWS SDK for .NET Referensi API.

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::RDS::RDSClient client(clientConfig);

        Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
                client.ModifyDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
modified."
                      << std::endl;
        }
        else {
```

```
            std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- Untuk detail API, lihat ModifyDB ClusterParameterGroup di AWS SDK for C++ Referensi API.

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(parameterGroupName string, params
 []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(context.TODO(),
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- Untuk detail API, lihat ModifyDB ClusterParameterGroup di AWS SDK for Go Referensi API.

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
       try {
           DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .maxRecords(20)
                   .build();

           List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                   .dbClusterParameterGroups();
           for (DBClusterParameterGroup group : groups) {
               System.out.println("The group name is " +
group.dbClusterParameterGroupName());
               System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
           }

       } catch (RdsException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }
```

- Untuk detail API, lihat ModifyDB ClusterParameterGroup di AWS SDK for Java 2.x Referensi API.

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 = Parameter {
        parameterName = "auto_increment_offset"
        applyMethod = ApplyMethod.fromValue("immediate")
        parameterValue = "5"
    }

    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest = ModifyDbClusterParameterGroupRequest {
        dbClusterParameterGroupName = dClusterGroupName
        parameters = paraList
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
    }
}
```

- Untuk detail API, lihat ModifyDB ClusterParameterGroup di AWS SDK untuk referensi API Kotlin.

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""

    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
 RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
```

```
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response
```

- Untuk detail API, lihat ModifyDB ClusterParameterGroup di AWS SDK for Python (Boto3) Referensi API.

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
   // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
   pub async fn update_auto_increment(
       &self,
       offset: u8,
       increment: u8,
   ) -> Result<(), ScenarioError> {
       let modify_db_cluster_parameter_group = self
           .rds
           .modify_db_cluster_parameter_group(
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               vec![
                   Parameter::builder()
                       .parameter_name("auto_increment_offset")
```

```
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
            )
            .await;

        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }

    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
 SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
            .send()
            .await
    }

#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
```

```
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}

#[tokio::test]
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(

 ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
```

```
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}
```

- Untuk detail API, lihat ModifyDB ClusterParameterGroup di AWS SDK untuk referensi API Rust.

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Skenario untuk Aurora menggunakan SDK AWS

Contoh kode berikut menunjukkan cara menerapkan skenario umum di Aurora dengan AWS SDK. Skenario ini menunjukkan cara menyelesaikan tugas tertentu dengan memanggil beberapa fungsi dalam Aurora. Setiap skenario menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan kode.

Contoh-contoh

- Memulai cluster Aurora DB menggunakan SDK AWS

Memulai cluster Aurora DB menggunakan SDK AWS

Contoh kode berikut ini menunjukkan cara:

- Membuat grup parameter klaster DB Aurora dan mengatur nilai parameter.
- Membuat klaster DB yang menggunakan grup parameter.
- Membuat instans DB yang berisi basis data.
- Mengambil snapshot klaster DB, lalu membersihkan sumber daya.

.NET

AWS SDK for .NET

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di penggugah/prompt perintah.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using AuroraActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace AuroraScenario;

/// <summary>
/// Scenario for Amazon Aurora examples.
/// </summary>
public class AuroraScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
    1.  Return a list of the available DB engine families for Aurora MySql using
 the DescribeDBEngineVersionsAsync method.
    2.  Select an engine family and create a custom DB cluster parameter group
 using the CreateDBClusterParameterGroupAsync method.
    3.  Get the parameter group using the DescribeDBClusterParameterGroupsAsync
 method.
    4.  Get some parameters in the group using the
 DescribeDBClusterParametersAsync method.
    5.  Parse and display some parameters in the group.
```

```
   6.  Modify the auto_increment_offset and auto_increment_increment parameters
       using the ModifyDBClusterParameterGroupAsync method.
   7.  Get and display the updated parameters using the
DescribeDBClusterParametersAsync method with a source of "user".
   8.  Get a list of allowed engine versions using the
DescribeDBEngineVersionsAsync method.
   9.  Create an Aurora DB cluster that contains a MySql database and uses the
parameter group.
       using the CreateDBClusterAsync method.
   10. Wait for the DB cluster to be ready using the DescribeDBClustersAsync
method.
   11. Display and select from a list of instance classes available for the
selected engine and version
       using the paginated DescribeOrderableDBInstanceOptions method.
   12. Create a database instance in the cluster using the CreateDBInstanceAsync
method.
   13. Wait for the DB instance to be ready using the DescribeDBInstances
method.
   14. Display the connection endpoint string for the new DB cluster.
   15. Create a snapshot of the DB cluster using the
CreateDBClusterSnapshotAsync method.
   16. Wait for DB snapshot to be ready using the
DescribeDBClusterSnapshotsAsync method.
   17. Delete the DB instance using the DeleteDBInstanceAsync method.
   18. Delete the DB cluster using the DeleteDBClusterAsync method.
   19. Wait for DB cluster to be deleted using the DescribeDBClustersAsync
methods.
   20. Delete the cluster parameter group using the
DeleteDBClusterParameterGroupAsync.
   */

   private static readonly string sepBar = new('-', 80);
   private static AuroraWrapper auroraWrapper = null!;
   private static ILogger logger = null!;
   private static readonly string engine = "aurora-mysql";
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon Relational Database Service
(Amazon RDS).
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
```

```
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
                    .AddTransient<AuroraWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        }).CreateLogger<AuroraScenario>();

        auroraWrapper = host.Services.GetRequiredService<AuroraWrapper>();

        Console.WriteLine(sepBar);
        Console.WriteLine(
            "Welcome to the Amazon Aurora: get started with DB clusters
example.");
        Console.WriteLine(sepBar);

        DBClusterParameterGroup parameterGroup = null!;
        DBCluster? newCluster = null;
        DBInstance? newInstance = null;

        try
        {
            var parameterGroupFamily = await ChooseParameterGroupFamilyAsync();

            parameterGroup = await
CreateDBParameterGroupAsync(parameterGroupFamily);

            var parameters = await
DescribeParametersInGroupAsync(parameterGroup.DBClusterParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await
ModifyParametersAsync(parameterGroup.DBClusterParameterGroupName, parameters);

            await
DescribeUserSourceParameters(parameterGroup.DBClusterParameterGroupName);
```

```
            var engineVersionChoice = await
ChooseDBEngineVersionAsync(parameterGroupFamily);

            var newClusterIdentifier = "Example-Cluster-" + DateTime.Now.Ticks;

            newCluster = await CreateNewCluster
            (
                parameterGroup,
                engine,
                engineVersionChoice.EngineVersion,
                newClusterIdentifier
            );

            var instanceClassChoice = await ChooseDBInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            newInstance = await CreateNewInstance(
                newClusterIdentifier,
                engine,
                engineVersionChoice.EngineVersion,
                instanceClassChoice.DBInstanceClass,
                newInstanceIdentifier
            );

            DisplayConnectionString(newCluster!);
            await CreateSnapshot(newCluster!);
            await CleanupResources(newInstance, newCluster, parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
        catch (Exception ex)

        {
            await CleanupResources(newInstance, newCluster, parameterGroup);
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// Choose the Aurora DB parameter group family from a list of available
options.
```

```
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamilyAsync()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await
auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine);

        Console.WriteLine($"1. The following is a list of available DB parameter
group families for engine {engine}:");

        var parameterGroupFamilies =
            engines.GroupBy(e => e.DBParameterGroupFamily).ToList();
        for (var i = 1; i <= parameterGroupFamilies.Count; i++)
        {
            var parameterGroupFamily = parameterGroupFamilies[i - 1];
            // List the available parameter group families.
            Console.WriteLine(
                $"\t{i}. Family: {parameterGroupFamily.Key}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("2. Select an available DB parameter group family
by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber -
1];
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
    }

    /// <summary>
    /// Create and get information on a DB parameter group.
    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the
new DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBClusterParameterGroup>
CreateDBParameterGroupAsync(string dbParameterGroupFamily)
```

```
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
 {dbParameterGroupFamily}:");

        var parameterGroup = await
 auroraWrapper.CreateCustomClusterParameterGroupAsync(
            dbParameterGroupFamily,
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            "New example parameter group");

        var groupInfo =
            await
 auroraWrapper.DescribeCustomDBClusterParameterGroupAsync(parameterGroup.DBClusterParamet

        Console.WriteLine(
            $"3. New DB parameter group created: \n\t{groupInfo?.Description}, \n
\tARN {groupInfo?.DBClusterParameterGroupName}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <param name="parameterNames">Optional specific names of parameters to
 describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>>
 DescribeParametersInGroupAsync(string parameterGroupName, List<string>?
 parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);

        var parameters =
            await
 auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName);

        var matchingParameters =
```

```
            parameters.Where(p => parameterNames == null ||
 parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }

    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParametersAsync(string parameterGroupName,
 List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        await
 auroraWrapper.ModifyIntegerParametersInGroupAsync(parameterGroupName,
 parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</
param>
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string
 parameterGroupName)
    {
```

```
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe updated user source parameters in the
group.");

        var parameters =
            await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName,
"user");

        parameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group
family {dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"\t{i}. Engine: {version.Engine} Version
{version.EngineVersion}.");
            i++;
        }
```

```
        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by
 entering a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Create a new RDS DB cluster.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
    /// <param name="engineName">Engine to use for the DB cluster.</param>
    /// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
    /// <param name="clusterIdentifier">Cluster identifier to use for the DB
 cluster.</param>
    /// <returns>The new DB cluster.</returns>
    public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string clusterIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"9. Create a new DB cluster with identifier
 {clusterIdentifier}.");

        DBCluster newCluster;
        var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
        var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
 clusterIdentifier);

        if (isClusterCreated)
        {
            Console.WriteLine("Cluster already created.");
            newCluster = clusters.First(i => i.DBClusterIdentifier ==
 clusterIdentifier);
```

```
        }
        else
        {
            Console.WriteLine("Enter an admin username:");
            var username = Console.ReadLine();

            Console.WriteLine("Enter an admin password:");
            var password = Console.ReadLine();

            newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
                "ExampleDatabase",
                clusterIdentifier,
                parameterGroup.DBClusterParameterGroupName,
                engineName,
                engineVersion,
                username!,
                password!
            );

            Console.WriteLine("10. Waiting for DB cluster to be ready...");
            while (newCluster.Status != "available")
            {
                Console.Write(".");
                Thread.Sleep(5000);
                clusters = await
 auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
                newCluster = clusters.First();
            }
        }

        Console.WriteLine(sepBar);
        return newCluster;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption>
 ChooseDBInstanceClass(string engine, string engineVersion)
    {
```

```
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await
 auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
 engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
 version {engineVersion}:");
        int i = 1;


        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("11. Select an available DB instance class by
 entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new DB instance.
    /// </summary>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
    /// <param name="instanceIdentifier">Instance identifier to use for the DB
 instance.</param>
```

```
    /// <returns>The new DB instance.</returns>
    public static async Task<DBInstance?> CreateNewInstance(
        string clusterIdentifier,
        string engineName,
        string engineVersion,
        string instanceClass,
        string instanceIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"12. Create a new DB instance with identifier
{instanceIdentifier}.");
        bool isInstanceReady = false;
        DBInstance newInstance;
        var instances = await auroraWrapper.DescribeDBInstancesPagedAsync();
        isInstanceReady = instances.FirstOrDefault(i =>
            i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
"available";

        if (isInstanceReady)
        {
            Console.WriteLine("Instance already created.");
            newInstance = instances.First(i => i.DBInstanceIdentifier ==
instanceIdentifier);
        }
        else
        {

            newInstance = await auroraWrapper.CreateDBInstanceInClusterAsync(
                clusterIdentifier,
                instanceIdentifier,
                engineName,
                engineVersion,
                instanceClass
            );

            Console.WriteLine("13. Waiting for DB instance to be ready...");
            while (!isInstanceReady)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                instances = await
auroraWrapper.DescribeDBInstancesPagedAsync(instanceIdentifier);
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
"available";
```

```
                newInstance = instances.First();
            }
        }

        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">The DB cluster to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("14. New DB cluster connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {cluster.Endpoint} -P {cluster.Port} "
            + $"-u {cluster.MasterUsername} -p [YOUR PASSWORD]\n");

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">DB cluster to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBClusterSnapshot> CreateSnapshot(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"15. Creating snapshot from DB cluster
 {cluster.DBClusterIdentifier}.");
        var snapshot = await
 auroraWrapper.CreateClusterSnapshotByIdentifierAsync(
            cluster.DBClusterIdentifier,
            "ExampleSnapshot-" + DateTime.Now.Ticks);

        // Wait for the snapshot to be available.
        bool isSnapshotReady = false;
```

```
        Console.WriteLine($"16. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            Console.Write(".");
            Thread.Sleep(5000);
            var snapshots =
                await
auroraWrapper.DescribeDBClusterSnapshotsByIdentifierAsync(cluster.DBClusterIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBClusterSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Clean up resources from the scenario.
    /// </summary>
    /// <param name="newInstance">The instance to clean up.</param>
    /// <param name="newCluster">The cluster to clean up.</param>
    /// <param name="parameterGroup">The parameter group to clean up.</param>
    /// <returns>Async Task.</returns>
    private static async Task CleanupResources(
        DBInstance? newInstance,
        DBCluster? newCluster,
        DBClusterParameterGroup? parameterGroup)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (newInstance is not null && GetYesNoResponse($"\tClean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }
```

```
        if (newCluster is not null && GetYesNoResponse($"\tClean up cluster
 {newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
            Console.WriteLine($"18. Deleting the DB cluster
 {newCluster.DBClusterIdentifier}.");
            await
 auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                var cluster = await auroraWrapper.DescribeDBClustersPagedAsync();
                isClusterDeleted = cluster.All(i => i.DBClusterIdentifier !=
 newCluster.DBClusterIdentifier);
            }

            Console.WriteLine("DB cluster deleted.");
        }

        if (parameterGroup is not null && GetYesNoResponse($"\tClean up parameter
 group? (y/n)"))
        {
            Console.WriteLine($"20. Deleting the DB parameter group
 {parameterGroup.DBClusterParameterGroupName}.");
            await
 auroraWrapper.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGr
            Console.WriteLine("Parameter group deleted.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</
param>
    /// <returns>True if the user responds with a yes.</returns>
```

```
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
```

Metode pembungkus yang dipanggil oleh skenario untuk mengelola tindakan Aurora.

```
using Amazon.RDS;
using Amazon.RDS.Model;

namespace AuroraActions;

/// <summary>
/// Wrapper for the Amazon Aurora cluster client operations.
/// </summary>
public class AuroraWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public AuroraWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family
 name.</param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
```

```
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }

    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</
param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await
 _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }

    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
 DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();
```

```
        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await
 _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }

    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</
param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
 DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }

    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
```

```
    /// <param name="parameters">The list of integer parameters to modify.</
param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string
 groupName, List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the
 allowed values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await
 _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }


    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
```

```
   public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string
engineVersion)
   {
       // Use a paginator to get a list of DB instance options.
       var results = new List<OrderableDBInstanceOption>();
       var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
           new DescribeOrderableDBInstanceOptionsRequest()
           {
               Engine = engine,
               EngineVersion = engineVersion,
           });
       // Get the entire list using the paginator.
       await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
       {
           results.Add(instanceOptions);
       }
       return results;
   }

   /// <summary>
   /// Delete a particular parameter group by name.
   /// </summary>
   /// <param name="groupName">The name of the parameter group.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> DeleteClusterParameterGroupByNameAsync(string
groupName)
   {
       var request = new DeleteDBClusterParameterGroupRequest
       {
           DBClusterParameterGroupName = groupName,
       };

       var response = await
_amazonRDS.DeleteDBClusterParameterGroupAsync(request);
       return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
   }

   /// <summary>
   /// Create a new cluster and database.
   /// </summary>
   /// <param name="dbName">The name of the new database.</param>
```

```
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
```

```
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }

    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
cluster.</param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            results.AddRange(response.DBClusters);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action
DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
```

```
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name
or size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });
```

```
        return response.DBClusterSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });
```

```
        return response.DBCluster;
    }

    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
 dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for .NET .
  - CreateDBCluster
  - dibuatB ClusterParameterGroup
  - dibuatB ClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DihapusB ClusterParameterGroup
  - DeleteDBInstance
  - DijelaskanB ClusterParameterGroups
  - DijelaskanB ClusterParameters
  - DijelaskanB ClusterSnapshots
  - DescribeDBClusters
  - DijelaskanB EngineVersions

- DescribeDBInstances
- DescribeOrderableDB InstanceOptions
- ModifyDB ClusterParameterGroup

C++

SDK for C++

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Routine which creates an Amazon Aurora DB cluster and demonstrates several
 operations
//! on that cluster.
/*!
 \sa gettingStartedWithDBClusters()
 \param clientConfiguration: AWS client configuration.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::gettingStartedWithDBClusters(
        const Aws::Client::ClientConfiguration &clientConfig) {
    Aws::RDS::RDSClient client(clientConfig);

    printAsterisksLine();
    std::cout << "Welcome to the Amazon Relational Database Service (Amazon
 Aurora)"
              << std::endl;
    std::cout << "get started with DB clusters demo." << std::endl;
    printAsterisksLine();

    std::cout << "Checking for an existing DB cluster parameter group named '" <<
              CLUSTER_PARAMETER_GROUP_NAME << "'." << std::endl;
    Aws::String dbParameterGroupFamily("Undefined");
    bool parameterGroupFound = true;
```

```
    {
        // 1. Check if the DB cluster parameter group already exists.
        Aws::RDS::Model::DescribeDBClusterParameterGroupsRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);

        Aws::RDS::Model::DescribeDBClusterParameterGroupsOutcome outcome =
                client.DescribeDBClusterParameterGroups(request);

        if (outcome.IsSuccess()) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' already exists." <<
 std::endl;
            dbParameterGroupFamily =
 outcome.GetResult().GetDBClusterParameterGroups()
[0].GetDBParameterGroupFamily();
        }
        else if (outcome.GetError().GetErrorType() ==
                 Aws::RDS::RDSErrors::D_B_PARAMETER_GROUP_NOT_FOUND_FAULT) {
            std::cout << "DB cluster parameter group named '" <<
                      CLUSTER_PARAMETER_GROUP_NAME << "' does not exist." <<
 std::endl;
            parameterGroupFound = false;
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameterGroups. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    if (!parameterGroupFound) {
        Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

        // 2. Get available parameter group families for the specified engine.
        if (!getDBEngineVersions(DB_ENGINE, NO_PARAMETER_GROUP_FAMILY,
                                 engineVersions, client)) {
            return false;
        }

        std::cout << "Getting available parameter group families for " <<
 DB_ENGINE
                  << "."
                  << std::endl;
```

```
        std::vector<Aws::String> families;
        for (const Aws::RDS::Model::DBEngineVersion &version: engineVersions) {
            Aws::String family = version.GetDBParameterGroupFamily();
            if (std::find(families.begin(), families.end(), family) ==
                families.end()) {
                families.push_back(family);
                std::cout << "  " << families.size() << ": " << family <<
std::endl;
            }
        }

        int choice = askQuestionForIntRange("Which family do you want to use? ",
1,
                                            static_cast<int>(families.size()));
        dbParameterGroupFamily = families[choice - 1];
    }
    if (!parameterGroupFound) {
        // 3.  Create a DB cluster parameter group.
        Aws::RDS::Model::CreateDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetDBParameterGroupFamily(dbParameterGroupFamily);
        request.SetDescription("Example cluster parameter group.");

        Aws::RDS::Model::CreateDBClusterParameterGroupOutcome outcome =
                client.CreateDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
created."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    printAsterisksLine();
    std::cout << "Let's set some parameter values in your cluster parameter
group."
              << std::endl;
```

```
    Aws::Vector<Aws::RDS::Model::Parameter> autoIncrementParameters;
    // 4.  Get the parameters in the DB cluster parameter group.
    if (!getDBClusterParameters(CLUSTER_PARAMETER_GROUP_NAME,
AUTO_INCREMENT_PREFIX,
                                NO_SOURCE,
                                autoIncrementParameters,
                                client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    Aws::Vector<Aws::RDS::Model::Parameter> updateParameters;

    for (Aws::RDS::Model::Parameter &autoIncParameter: autoIncrementParameters) {
        if (autoIncParameter.GetIsModifiable() &&
            (autoIncParameter.GetDataType() == "integer")) {
            std::cout << "The " << autoIncParameter.GetParameterName()
                      << " is described as: " <<
                      autoIncParameter.GetDescription() << "." << std::endl;
            if (autoIncParameter.ParameterValueHasBeenSet()) {
                std::cout << "The current value is "
                          << autoIncParameter.GetParameterValue()
                          << "." << std::endl;
            }
            std::vector<int> splitValues = splitToInts(
                    autoIncParameter.GetAllowedValues(), '-');
            if (splitValues.size() == 2) {
                int newValue = askQuestionForIntRange(
                        Aws::String("Enter a new value between ") +
                        autoIncParameter.GetAllowedValues() + ": ",
                        splitValues[0], splitValues[1]);
                autoIncParameter.SetParameterValue(std::to_string(newValue));
                updateParameters.push_back(autoIncParameter);

            }
            else {
                std::cerr << "Error parsing " <<
autoIncParameter.GetAllowedValues()
                          << std::endl;
            }
        }
    }

    {
```

```
        // 5.  Modify the auto increment parameters in the DB cluster parameter
group.
        Aws::RDS::Model::ModifyDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetParameters(updateParameters);

        Aws::RDS::Model::ModifyDBClusterParameterGroupOutcome outcome =
                client.ModifyDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster parameter group was successfully
modified."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::ModifyDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    }

    std::cout
            << "You can get a list of parameters you've set by specifying a
source of 'user'."
            << std::endl;

    Aws::Vector<Aws::RDS::Model::Parameter> userParameters;
    // 6.  Display the modified parameters in the DB cluster parameter group.
    if (!getDBClusterParameters(CLUSTER_PARAMETER_GROUP_NAME, NO_NAME_PREFIX,
"user",
                                userParameters,
                                client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
        return false;
    }

    for (const auto &userParameter: userParameters) {
        std::cout << "  " << userParameter.GetParameterName() << ", " <<
                  userParameter.GetDescription() << ", parameter value - "
                  << userParameter.GetParameterValue() << std::endl;
    }

    printAsterisksLine();
    std::cout << "Checking for an existing DB Cluster." << std::endl;
```

```
   Aws::RDS::Model::DBCluster dbCluster;
   // 7.  Check if the DB cluster already exists.
   if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
       cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
       return false;
   }

   Aws::String engineVersionName;
   Aws::String engineName;
   if (dbCluster.DBClusterIdentifierHasBeenSet()) {
       std::cout << "The DB cluster already exists." << std::endl;
       engineVersionName = dbCluster.GetEngineVersion();
       engineName = dbCluster.GetEngine();

   }
   else {
       std::cout << "Let's create a DB cluster." << std::endl;
       const Aws::String administratorName = askQuestion(
               "Enter an administrator username for the database: ");
       const Aws::String administratorPassword = askQuestion(
               "Enter a password for the administrator (at least 8 characters):
");
       Aws::Vector<Aws::RDS::Model::DBEngineVersion> engineVersions;

       // 8.  Get a list of engine versions for the parameter group family.
       if (!getDBEngineVersions(DB_ENGINE, dbParameterGroupFamily,
engineVersions,
                                client)) {
           cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
           return false;
       }

       std::cout << "The available engines for your parameter group family are:"
                 << std::endl;

       int index = 1;
       for (const Aws::RDS::Model::DBEngineVersion &engineVersion:
engineVersions) {
           std::cout << "   " << index << ": " <<
engineVersion.GetEngineVersion()
                     << std::endl;
           ++index;
       }
```

```
        int choice = askQuestionForIntRange("Which engine do you want to use? ",
1,

static_cast<int>(engineVersions.size()));
        const Aws::RDS::Model::DBEngineVersion engineVersion =
engineVersions[choice -
                                                                              1];

        engineName = engineVersion.GetEngine();
        engineVersionName = engineVersion.GetEngineVersion();
        std::cout << "Creating a DB cluster named '" << DB_CLUSTER_IDENTIFIER
                  << "' and database '" << DB_NAME << "'.\n"
                  << "The DB cluster is configured to use your custom cluster
parameter group '"
                  << CLUSTER_PARAMETER_GROUP_NAME << "', and \n"
                  << "selected engine version " <<
engineVersion.GetEngineVersion()
                  << ".\nThis typically takes several minutes." << std::endl;

        Aws::RDS::Model::CreateDBClusterRequest request;
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        request.SetEngine(engineName);
        request.SetEngineVersion(engineVersionName);
        request.SetMasterUsername(administratorName);
        request.SetMasterUserPassword(administratorPassword);

        Aws::RDS::Model::CreateDBClusterOutcome outcome =
                client.CreateDBCluster(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB cluster creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::CreateDBCluster. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, "", "", client);
            return false;
        }
    }

    std::cout << "Waiting for the DB cluster to become available." << std::endl;
```

```
    int counter = 0;
    // 11. Wait for the DB cluster to become available.
    do {
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 900) {
            std::cerr << "Wait for cluster to become available timed out ofter "
                      << counter
                      << " seconds." << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, "", client);
            return false;
        }

        dbCluster = Aws::RDS::Model::DBCluster();
        if (!describeDBCluster(DB_CLUSTER_IDENTIFIER, dbCluster, client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                             DB_CLUSTER_IDENTIFIER, "", client);
            return false;
        }

        if ((counter % 20) == 0) {
            std::cout << "Current DB cluster status is '"
                      << dbCluster.GetStatus()
                      << "' after " << counter << " seconds." << std::endl;
        }
    } while (dbCluster.GetStatus() != "available");

    if (dbCluster.GetStatus() == "available") {
        std::cout << "The DB cluster has been created." << std::endl;
    }

    printAsterisksLine();
    Aws::RDS::Model::DBInstance dbInstance;
    // 11.  Check if the DB instance already exists.
    if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
        cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER, "",
                         client);
        return false;
    }

    if (dbInstance.DbInstancePortHasBeenSet()) {
        std::cout << "The DB instance already exists." << std::endl;
```

```
    }
    else {
        std::cout << "Let's create a DB instance." << std::endl;

        Aws::String dbInstanceClass;
        // 12.  Get a list of instance classes.
        if (!chooseDBInstanceClass(engineName,
                                   engineVersionName,
                                   dbInstanceClass,
                                   client)) {
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
""",
                             client);
            return false;
        }

        std::cout << "Creating a DB instance named '" << DB_INSTANCE_IDENTIFIER
                  << "' with selected DB instance class '" << dbInstanceClass
                  << "'.\nThis typically takes several minutes." << std::endl;

        // 13. Create a DB instance.
        Aws::RDS::Model::CreateDBInstanceRequest request;
        request.SetDBInstanceIdentifier(DB_INSTANCE_IDENTIFIER);
        request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
        request.SetEngine(engineName);
        request.SetDBInstanceClass(dbInstanceClass);

        Aws::RDS::Model::CreateDBInstanceOutcome outcome =
                client.CreateDBInstance(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB instance creation has started."
                      << std::endl;
        }
        else {
            std::cerr << "Error with RDS::CreateDBInstance. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME, DB_CLUSTER_IDENTIFIER,
""",
                             client);
            return false;
        }
    }
```

```
   std::cout << "Waiting for the DB instance to become available." << std::endl;

   counter = 0;
   // 14. Wait for the DB instance to become available.
   do {
       std::this_thread::sleep_for(std::chrono::seconds(1));
       ++counter;
       if (counter > 900) {
           std::cerr << "Wait for instance to become available timed out ofter "
                     << counter
                     << " seconds." << std::endl;
           cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                            DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
           return false;
       }

       dbInstance = Aws::RDS::Model::DBInstance();
       if (!describeDBInstance(DB_INSTANCE_IDENTIFIER, dbInstance, client)) {
           cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                            DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
           return false;
       }

       if ((counter % 20) == 0) {
           std::cout << "Current DB instance status is '"
                     << dbInstance.GetDBInstanceStatus()
                     << "' after " << counter << " seconds." << std::endl;
       }
   } while (dbInstance.GetDBInstanceStatus() != "available");

   if (dbInstance.GetDBInstanceStatus() == "available") {
       std::cout << "The DB instance has been created." << std::endl;
   }

   // 15. Display the connection string that can be used to connect a 'mysql'
shell to the database.
   displayConnection(dbCluster);

   printAsterisksLine();

   if (askYesNoQuestion(
```

```
            "Do you want to create a snapshot of your DB cluster (y/n)? ")) {
        Aws::String snapshotID(DB_CLUSTER_IDENTIFIER + "-" +
                               Aws::String(Aws::Utils::UUID::RandomUUID()));
        {
            std::cout << "Creating a snapshot named " << snapshotID << "." <<
std::endl;
            std::cout << "This typically takes a few minutes." << std::endl;

            // 16. Create a snapshot of the DB cluster. (CreateDBClusterSnapshot)
            Aws::RDS::Model::CreateDBClusterSnapshotRequest request;
            request.SetDBClusterIdentifier(DB_CLUSTER_IDENTIFIER);
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::CreateDBClusterSnapshotOutcome outcome =
                    client.CreateDBClusterSnapshot(request);

            if (outcome.IsSuccess()) {
                std::cout << "Snapshot creation has started."
                          << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::CreateDBClusterSnapshot. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                 DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }
        }

        std::cout << "Waiting for the snapshot to become available." <<
std::endl;

        Aws::RDS::Model::DBClusterSnapshot snapshot;
        counter = 0;
        do {
            std::this_thread::sleep_for(std::chrono::seconds(1));
            ++counter;
            if (counter > 600) {
                std::cerr << "Wait for snapshot to be available timed out ofter "
                          << counter
                          << " seconds." << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
```

```
                                DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }

            // 17. Wait for the snapshot to become available.
            Aws::RDS::Model::DescribeDBClusterSnapshotsRequest request;
            request.SetDBClusterSnapshotIdentifier(snapshotID);

            Aws::RDS::Model::DescribeDBClusterSnapshotsOutcome outcome =
                    client.DescribeDBClusterSnapshots(request);

            if (outcome.IsSuccess()) {
                snapshot = outcome.GetResult().GetDBClusterSnapshots()[0];
            }
            else {
                std::cerr << "Error with Aurora::DescribeDBClusterSnapshots. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
                                DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
client);
                return false;
            }

            if ((counter % 20) == 0) {
                std::cout << "Current snapshot status is '"
                          << snapshot.GetStatus()
                          << "' after " << counter << " seconds." << std::endl;
            }
        } while (snapshot.GetStatus() != "available");

        if (snapshot.GetStatus() != "available") {
            std::cout << "A snapshot has been created." << std::endl;
        }
    }

    printAsterisksLine();

    bool result = true;
    if (askYesNoQuestion(
            "Do you want to delete the DB cluster, DB instance, and parameter
group (y/n)? ")) {
        result = cleanUpResources(CLUSTER_PARAMETER_GROUP_NAME,
```

```
                                    DB_CLUSTER_IDENTIFIER, DB_INSTANCE_IDENTIFIER,
                                    client);
    }

    return result;
}

//! Routine which gets a DB cluster description.
/*!
 \sa describeDBCluster()
 \param dbClusterIdentifier: A DB cluster identifier.
 \param clusterResult: The 'DBCluster' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBCluster(const Aws::String &dbClusterIdentifier,
                                       Aws::RDS::Model::DBCluster &clusterResult,
                                       const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBClustersRequest request;
    request.SetDBClusterIdentifier(dbClusterIdentifier);

    Aws::RDS::Model::DescribeDBClustersOutcome outcome =
            client.DescribeDBClusters(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        clusterResult = outcome.GetResult().GetDBClusters()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_CLUSTER_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::GDescribeDBClusters. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB cluster does not exist.
        // Instead, clusterResult is set to empty.
    else {
        clusterResult = Aws::RDS::Model::DBCluster();
    }

    return result;

}
```

```
//! Routine which gets DB parameters using the 'DescribeDBClusterParameters' api.
/*!
 \sa getDBCLusterParameters()
 \param parameterGroupName: The name of the cluster parameter group.
 \param namePrefix: Prefix string to filter results by parameter name.
 \param source: A source such as 'user', ignored if empty.
 \param parametersResult: Vector of 'Parameter' objects returned by the routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBCLusterParameters(const Aws::String
 &parameterGroupName,
                                             const Aws::String &namePrefix,
                                             const Aws::String &source,

 Aws::Vector<Aws::RDS::Model::Parameter> &parametersResult,
                                             const Aws::RDS::RDSClient &client) {
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeDBClusterParametersRequest request;
        request.SetDBClusterParameterGroupName(CLUSTER_PARAMETER_GROUP_NAME);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }
        if (!source.empty()) {
            request.SetSource(source);
        }

        Aws::RDS::Model::DescribeDBClusterParametersOutcome outcome =
                client.DescribeDBClusterParameters(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::Parameter> &parameters =
                    outcome.GetResult().GetParameters();
            for (const Aws::RDS::Model::Parameter &parameter: parameters) {
                if (!namePrefix.empty()) {
                    if (parameter.GetParameterName().find(namePrefix) == 0) {
                        parametersResult.push_back(parameter);
                    }
                }
                else {
                    parametersResult.push_back(parameter);
```

```
                }
            }

            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeDBClusterParameters. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    return true;
}


//! Routine which gets available DB engine versions for an engine name and
//! an optional parameter group family.
/*!
 \sa getDBEngineVersions()
 \param engineName: A DB engine name.
 \param parameterGroupFamily: A parameter group family name, ignored if empty.
 \param engineVersionsResult: Vector of 'DBEngineVersion' objects returned by the
 routine.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::getDBEngineVersions(const Aws::String &engineName,
                                         const Aws::String &parameterGroupFamily,

 Aws::Vector<Aws::RDS::Model::DBEngineVersion> &engineVersionsResult,
                                         const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBEngineVersionsRequest request;
    request.SetEngine(engineName);
    if (!parameterGroupFamily.empty()) {
        request.SetDBParameterGroupFamily(parameterGroupFamily);
    }

    Aws::RDS::Model::DescribeDBEngineVersionsOutcome outcome =
            client.DescribeDBEngineVersions(request);

    if (outcome.IsSuccess()) {
        engineVersionsResult = outcome.GetResult().GetDBEngineVersions();
```

```
    }
    else {
        std::cerr << "Error with Aurora::DescribeDBEngineVersionsRequest. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}


//! Routine which gets a DB instance description.
/*!
 \sa describeDBCluster()
 \param dbInstanceIdentifier: A DB instance identifier.
 \param instanceResult: The 'DBInstance' object containing the description.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::describeDBInstance(const Aws::String &dbInstanceIdentifier,
                                        Aws::RDS::Model::DBInstance
 &instanceResult,
                                        const Aws::RDS::RDSClient &client) {
    Aws::RDS::Model::DescribeDBInstancesRequest request;
    request.SetDBInstanceIdentifier(dbInstanceIdentifier);

    Aws::RDS::Model::DescribeDBInstancesOutcome outcome =
            client.DescribeDBInstances(request);

    bool result = true;
    if (outcome.IsSuccess()) {
        instanceResult = outcome.GetResult().GetDBInstances()[0];
    }
    else if (outcome.GetError().GetErrorType() !=
             Aws::RDS::RDSErrors::D_B_INSTANCE_NOT_FOUND_FAULT) {
        result = false;
        std::cerr << "Error with Aurora::DescribeDBInstances. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
        // This example does not log an error if the DB instance does not exist.
        // Instead, instanceResult is set to empty.
    else {
        instanceResult = Aws::RDS::Model::DBInstance();
```

```
    }

    return result;
}


//! Routine which gets available DB instance classes, displays the list
//! to the user, and returns the user selection.
/*!
 \sa chooseDBInstanceClass()
 \param engineName: The DB engine name.
 \param engineVersion: The DB engine version.
 \param dbInstanceClass: String for DB instance class chosen by the user.
 \param client: 'RDSClient' instance.
 \return bool: Successful completion.
 */
bool AwsDoc::Aurora::chooseDBInstanceClass(const Aws::String &engine,
                                           const Aws::String &engineVersion,
                                           Aws::String &dbInstanceClass,
                                           const Aws::RDS::RDSClient &client) {
    std::vector<Aws::String> instanceClasses;
    Aws::String marker; // The marker is used for pagination.
    do {
        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsRequest request;
        request.SetEngine(engine);
        request.SetEngineVersion(engineVersion);
        if (!marker.empty()) {
            request.SetMarker(marker);
        }

        Aws::RDS::Model::DescribeOrderableDBInstanceOptionsOutcome outcome =
                client.DescribeOrderableDBInstanceOptions(request);

        if (outcome.IsSuccess()) {
            const Aws::Vector<Aws::RDS::Model::OrderableDBInstanceOption>
 &options =
                    outcome.GetResult().GetOrderableDBInstanceOptions();
            for (const Aws::RDS::Model::OrderableDBInstanceOption &option:
 options) {
                const Aws::String &instanceClass = option.GetDBInstanceClass();
                if (std::find(instanceClasses.begin(), instanceClasses.end(),
 instanceClass) == instanceClasses.end()) {
                    instanceClasses.push_back(instanceClass);
                }
```

```
            }
            marker = outcome.GetResult().GetMarker();
        }
        else {
            std::cerr << "Error with Aurora::DescribeOrderableDBInstanceOptions.
 "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    } while (!marker.empty());

    std::cout << "The available DB instance classes for your database engine
 are:"
              << std::endl;
    for (int i = 0; i < instanceClasses.size(); ++i) {
        std::cout << "   " << i + 1 << ": " << instanceClasses[i] << std::endl;
    }

    int choice = askQuestionForIntRange(
            "Which DB instance class do you want to use? ",
            1, static_cast<int>(instanceClasses.size()));
    dbInstanceClass = instanceClasses[choice - 1];
    return true;
}

//! Routine which deletes resources created by the scenario.
/*!
\sa cleanUpResources()
\param parameterGroupName: A parameter group name, this may be empty.
\param dbInstanceIdentifier: A DB instance identifier, this may be empty.
\param client: 'RDSClient' instance.
\return bool: Successful completion.
*/
bool AwsDoc::Aurora::cleanUpResources(const Aws::String &parameterGroupName,
                                      const Aws::String &dbClusterIdentifier,
                                      const Aws::String &dbInstanceIdentifier,
                                      const Aws::RDS::RDSClient &client) {
    bool result = true;
    bool instanceDeleting = false;
    bool clusterDeleting = false;
    if (!dbInstanceIdentifier.empty()) {
        {
            // 18. Delete the DB instance.
```

```
            Aws::RDS::Model::DeleteDBInstanceRequest request;
            request.SetDBInstanceIdentifier(dbInstanceIdentifier);
            request.SetSkipFinalSnapshot(true);
            request.SetDeleteAutomatedBackups(true);

            Aws::RDS::Model::DeleteDBInstanceOutcome outcome =
                    client.DeleteDBInstance(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB instance deletion has started."
                          << std::endl;
                instanceDeleting = true;
                std::cout
                        << "Waiting for DB instance to delete before deleting the
parameter group."
                        << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBInstance. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }

    if (!dbClusterIdentifier.empty()) {
        {
            // 19. Delete the DB cluster.
            Aws::RDS::Model::DeleteDBClusterRequest request;
            request.SetDBClusterIdentifier(dbClusterIdentifier);
            request.SetSkipFinalSnapshot(true);

            Aws::RDS::Model::DeleteDBClusterOutcome outcome =
                    client.DeleteDBCluster(request);

            if (outcome.IsSuccess()) {
                std::cout << "DB cluster deletion has started."
                          << std::endl;
                clusterDeleting = true;
                std::cout
                        << "Waiting for DB cluster to delete before deleting the
parameter group."
                        << std::endl;
```

```
                std::cout << "This may take a while." << std::endl;
            }
            else {
                std::cerr << "Error with Aurora::DeleteDBCluster. "
                          << outcome.GetError().GetMessage()
                          << std::endl;
                result = false;
            }
        }
    }
    int counter = 0;

    while (clusterDeleting || instanceDeleting) {
        // 20. Wait for the DB cluster and instance to be deleted.
        std::this_thread::sleep_for(std::chrono::seconds(1));
        ++counter;
        if (counter > 800) {
            std::cerr << "Wait for instance to delete timed out ofter " <<
counter
                      << " seconds." << std::endl;
            return false;
        }

        Aws::RDS::Model::DBInstance dbInstance = Aws::RDS::Model::DBInstance();
        if (instanceDeleting) {
            if (!describeDBInstance(dbInstanceIdentifier, dbInstance, client)) {
                return false;
            }
            instanceDeleting = dbInstance.DBInstanceIdentifierHasBeenSet();
        }

        Aws::RDS::Model::DBCluster dbCluster = Aws::RDS::Model::DBCluster();
        if (clusterDeleting) {
            if (!describeDBCluster(dbClusterIdentifier, dbCluster, client)) {
                return false;
            }

            clusterDeleting = dbCluster.DBClusterIdentifierHasBeenSet();
        }

        if ((counter % 20) == 0) {
            if (instanceDeleting) {
                std::cout << "Current DB instance status is '"
```

```
                        << dbInstance.GetDBInstanceStatus() << "." <<
 std::endl;
            }

            if (clusterDeleting) {
                std::cout << "Current DB cluster status is '"
                          << dbCluster.GetStatus() << "." << std::endl;
            }
        }
    }

    if (!parameterGroupName.empty()) {
        // 21. Delete the DB cluster parameter group.
        Aws::RDS::Model::DeleteDBClusterParameterGroupRequest request;
        request.SetDBClusterParameterGroupName(parameterGroupName);

        Aws::RDS::Model::DeleteDBClusterParameterGroupOutcome outcome =
                client.DeleteDBClusterParameterGroup(request);

        if (outcome.IsSuccess()) {
            std::cout << "The DB parameter group was successfully deleted."
                      << std::endl;
        }
        else {
            std::cerr << "Error with Aurora::DeleteDBClusterParameterGroup. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            result = false;
        }
    }

    return result;
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for C++ .
  - CreateDBCluster
  - dibuatB ClusterParameterGroup
  - dibuatB ClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster

- DihapusB ClusterParameterGroup
- DeleteDBInstance
- DijelaskanB ClusterParameterGroups
- DijelaskanB ClusterParameters
- DijelaskanB ClusterSnapshots
- DescribeDBClusters
- DijelaskanB EngineVersions
- DescribeDBInstances
- DescribeOrderableDB InstanceOptions
- ModifyDB ClusterParameterGroup

Go

SDK for Go V2

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di prompt perintah.

```
// GetStartedClusters is an interactive example that shows you how to use the AWS
 SDK for Go
// with Amazon Aurora to do the following:
//
// 1. Create a custom DB cluster parameter group and set parameter values.
// 2. Create an Aurora DB cluster that is configured to use the parameter group.
// 3. Create a DB instance in the DB cluster that contains a database.
// 4. Take a snapshot of the DB cluster.
// 5. Delete the DB instance, DB cluster, and parameter group.
type GetStartedClusters struct {
 sdkConfig  aws.Config
 dbClusters actions.DbClusters
 questioner demotools.IQuestioner
 helper     IScenarioHelper
```

```
 isTestRun  bool
}

// NewGetStartedClusters constructs a GetStartedClusters instance from a
 configuration.
// It uses the specified config to get an Amazon Relational Database Service
 (Amazon RDS)
// client and create wrappers for the actions used in the scenario.
func NewGetStartedClusters(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
 helper IScenarioHelper) GetStartedClusters {
 auroraClient := rds.NewFromConfig(sdkConfig)
 return GetStartedClusters{
  sdkConfig:  sdkConfig,
  dbClusters: actions.DbClusters{AuroraClient: auroraClient},
  questioner: questioner,
  helper:     helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedClusters) Run(dbEngine string, parameterGroupName
 string,
 clusterName string, dbName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon Aurora DB Cluster demo.")
 log.Println(strings.Repeat("-", 88))

 parameterGroup := scenario.CreateParameterGroup(dbEngine, parameterGroupName)
 scenario.SetUserParameters(parameterGroupName)
 cluster := scenario.CreateCluster(clusterName, dbEngine, dbName, parameterGroup)
 scenario.helper.Pause(5)
 dbInstance := scenario.CreateInstance(cluster)
 scenario.DisplayConnection(cluster)
 scenario.CreateSnapshot(clusterName)
 scenario.Cleanup(dbInstance, cluster, parameterGroup)

 log.Println(strings.Repeat("-", 88))
```

```
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// CreateParameterGroup shows how to get available engine versions for a
 specified
// database engine and create a DB cluster parameter group that is compatible
 with a
// selected engine family.
func (scenario GetStartedClusters) CreateParameterGroup(dbEngine string,
 parameterGroupName string) *types.DBClusterParameterGroup {

 log.Printf("Checking for an existing DB cluster parameter group named %v.\n",
  parameterGroupName)
 parameterGroup, err := scenario.dbClusters.GetParameterGroup(parameterGroupName)
 if err != nil {
  panic(err)
 }
 if parameterGroup == nil {
  log.Printf("Getting available database engine versions for %v.\n", dbEngine)
  engineVersions, err := scenario.dbClusters.GetEngineVersions(dbEngine, "")
  if err != nil {
   panic(err)
  }

  familySet := map[string]struct{}{}
  for _, family := range engineVersions {
   familySet[*family.DBParameterGroupFamily] = struct{}{}
  }
  var families []string
  for family := range familySet {
   families = append(families, family)
  }
  sort.Strings(families)
  familyIndex := scenario.questioner.AskChoice("Which family do you want to use?
\n", families)
  log.Println("Creating a DB cluster parameter group.")
  _, err = scenario.dbClusters.CreateParameterGroup(
   parameterGroupName, families[familyIndex], "Example parameter group.")
  if err != nil {
   panic(err)
  }
  parameterGroup, err = scenario.dbClusters.GetParameterGroup(parameterGroupName)
  if err != nil {
```

```
   panic(err)
  }
 }
 log.Printf("Parameter group %v:\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tName: %v\n", *parameterGroup.DBClusterParameterGroupName)
 log.Printf("\tARN: %v\n", *parameterGroup.DBClusterParameterGroupArn)
 log.Printf("\tFamily: %v\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tDescription: %v\n", *parameterGroup.Description)
 log.Println(strings.Repeat("-", 88))
 return parameterGroup

}

// SetUserParameters shows how to get the parameters contained in a custom
 parameter
// group and update some of the parameter values in the group.
func (scenario GetStartedClusters) SetUserParameters(parameterGroupName string) {
 log.Println("Let's set some parameter values in your parameter group.")
 dbParameters, err := scenario.dbClusters.GetParameters(parameterGroupName, "")
 if err != nil {
  panic(err)
 }
 var updateParams []types.Parameter
 for _, dbParam := range dbParameters {
  if strings.HasPrefix(*dbParam.ParameterName, "auto_increment") &&
   dbParam.IsModifiable && *dbParam.DataType == "integer" {
   log.Printf("The %v parameter is described as:\n\t%v",
    *dbParam.ParameterName, *dbParam.Description)
   rangeSplit := strings.Split(*dbParam.AllowedValues, "-")
   lower, _ := strconv.Atoi(rangeSplit[0])
   upper, _ := strconv.Atoi(rangeSplit[1])
   newValue := scenario.questioner.AskInt(
    fmt.Sprintf("Enter a value between %v and %v:", lower, upper),
    demotools.InIntRange{Lower: lower, Upper: upper})
   dbParam.ParameterValue = aws.String(strconv.Itoa(newValue))
   updateParams = append(updateParams, dbParam)
  }
 }
 err = scenario.dbClusters.UpdateParameters(parameterGroupName, updateParams)
 if err != nil {
  panic(err)
 }
 log.Println("You can get a list of parameters you've set by specifying a source
 of 'user'.")
```

```
 userParameters, err := scenario.dbClusters.GetParameters(parameterGroupName,
 "user")
 if err != nil {
  panic(err)
 }
 log.Println("Here are the parameters you've set:")
 for _, param := range userParameters {
  log.Printf("\t%v: %v\n", *param.ParameterName, *param.ParameterValue)
 }
 log.Println(strings.Repeat("-", 88))
}

// CreateCluster shows how to create an Aurora DB cluster that contains a
 database
// of a specified type. The database is also configured to use a custom DB
 cluster
// parameter group.
func (scenario GetStartedClusters) CreateCluster(clusterName string, dbEngine
 string,
 dbName string, parameterGroup *types.DBClusterParameterGroup) *types.DBCluster {

 log.Println("Checking for an existing DB cluster.")
 cluster, err := scenario.dbClusters.GetDbCluster(clusterName)
 if err != nil {
  panic(err)
 }
 if cluster == nil {
  adminUsername := scenario.questioner.Ask(
   "Enter an administrator user name for the database: ", demotools.NotEmpty{})
  adminPassword := scenario.questioner.Ask(
   "Enter a password for the administrator (at least 8 characters): ",
 demotools.NotEmpty{})
  engineVersions, err := scenario.dbClusters.GetEngineVersions(dbEngine,
 *parameterGroup.DBParameterGroupFamily)
  if err != nil {
   panic(err)
  }
  var engineChoices []string
  for _, engine := range engineVersions {
   engineChoices = append(engineChoices, *engine.EngineVersion)
  }
  log.Println("The available engines for your parameter group are:")
  engineIndex := scenario.questioner.AskChoice("Which engine do you want to use?
\n", engineChoices)
```

```
  log.Printf("Creating DB cluster %v and database %v.\n", clusterName, dbName)
  log.Printf("The DB cluster is configured to use\nyour custom parameter group %v
\n",
   *parameterGroup.DBClusterParameterGroupName)
  log.Printf("and selected engine %v.\n", engineChoices[engineIndex])
  log.Println("This typically takes several minutes.")
  cluster, err = scenario.dbClusters.CreateDbCluster(
   clusterName, *parameterGroup.DBClusterParameterGroupName, dbName, dbEngine,
   engineChoices[engineIndex], adminUsername, adminPassword)
  if err != nil {
   panic(err)
  }
  for *cluster.Status != "available" {
   scenario.helper.Pause(30)
   cluster, err = scenario.dbClusters.GetDbCluster(clusterName)
   if err != nil {
    panic(err)
   }
   log.Println("Cluster created and available.")
  }
 }
 log.Println("Cluster data:")
 log.Printf("\tDBClusterIdentifier: %v\n", *cluster.DBClusterIdentifier)
 log.Printf("\tARN: %v\n", *cluster.DBClusterArn)
 log.Printf("\tStatus: %v\n", *cluster.Status)
 log.Printf("\tEngine: %v\n", *cluster.Engine)
 log.Printf("\tEngine version: %v\n", *cluster.EngineVersion)
 log.Printf("\tDBClusterParameterGroup: %v\n", *cluster.DBClusterParameterGroup)
 log.Printf("\tEngineMode: %v\n", *cluster.EngineMode)
 log.Println(strings.Repeat("-", 88))
 return cluster
}

// CreateInstance shows how to create a DB instance in an existing Aurora DB
 cluster.
// A new DB cluster contains no DB instances, so you must add one. The first DB
 instance
// that is added to a DB cluster defaults to a read-write DB instance.
func (scenario GetStartedClusters) CreateInstance(cluster *types.DBCluster)
 *types.DBInstance {
 log.Println("Checking for an existing database instance.")
 dbInstance, err := scenario.dbClusters.GetInstance(*cluster.DBClusterIdentifier)
 if err != nil {
  panic(err)
```

```
 }
 if dbInstance == nil {
  log.Println("Let's create a database instance in your DB cluster.")
  log.Println("First, choose a DB instance type:")
  instOpts, err := scenario.dbClusters.GetOrderableInstances(
   *cluster.Engine, *cluster.EngineVersion)
  if err != nil {
   panic(err)
  }
  var instChoices []string
  for _, opt := range instOpts {
   instChoices = append(instChoices, *opt.DBInstanceClass)
  }
  instIndex := scenario.questioner.AskChoice(
   "Which DB instance class do you want to use?\n", instChoices)
  log.Println("Creating a database instance. This typically takes several
 minutes.")
  dbInstance, err = scenario.dbClusters.CreateInstanceInCluster(
   *cluster.DBClusterIdentifier, *cluster.DBClusterIdentifier, *cluster.Engine,
   instChoices[instIndex])
  if err != nil {
   panic(err)
  }
  for *dbInstance.DBInstanceStatus != "available" {
   scenario.helper.Pause(30)
   dbInstance, err =
 scenario.dbClusters.GetInstance(*cluster.DBClusterIdentifier)
   if err != nil {
    panic(err)
   }
  }
 }
 log.Println("Instance data:")
 log.Printf("\tDBInstanceIdentifier: %v\n", *dbInstance.DBInstanceIdentifier)
 log.Printf("\tARN: %v\n", *dbInstance.DBInstanceArn)
 log.Printf("\tStatus: %v\n", *dbInstance.DBInstanceStatus)
 log.Printf("\tEngine: %v\n", *dbInstance.Engine)
 log.Printf("\tEngine version: %v\n", *dbInstance.EngineVersion)
 log.Println(strings.Repeat("-", 88))
 return dbInstance
}

// DisplayConnection displays connection information about an Aurora DB cluster
 and tips
```

```
// on how to connect to it.
func (scenario GetStartedClusters) DisplayConnection(cluster *types.DBCluster) {
 log.Println(
  "You can now connect to your database using your favorite MySql client.\n" +
   "One way to connect is by using the 'mysql' shell on an Amazon EC2 instance\n"
 +
   "that is running in the same VPC as your database cluster. Pass the endpoint,
\n" +
   "port, and administrator user name to 'mysql' and enter your password\n" +
   "when prompted:")
 log.Printf("\n\tmysql -h %v -P %v -u %v -p\n",
  *cluster.Endpoint, *cluster.Port, *cluster.MasterUsername)
 log.Println("For more information, see the User Guide for Aurora:\n" +
  "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
 log.Println(strings.Repeat("-", 88))
}

// CreateSnapshot shows how to create a DB cluster snapshot and wait until it's
 available.
func (scenario GetStartedClusters) CreateSnapshot(clusterName string) {
 if scenario.questioner.AskBool(
  "Do you want to create a snapshot of your DB cluster (y/n)? ", "y") {
  snapshotId := fmt.Sprintf("%v-%v", clusterName, scenario.helper.UniqueId())
  log.Printf("Creating a snapshot named %v. This typically takes a few minutes.
\n", snapshotId)
  snapshot, err := scenario.dbClusters.CreateClusterSnapshot(clusterName,
 snapshotId)
  if err != nil {
   panic(err)
  }
  for *snapshot.Status != "available" {
   scenario.helper.Pause(30)
   snapshot, err = scenario.dbClusters.GetClusterSnapshot(snapshotId)
   if err != nil {
    panic(err)
   }
  }
  log.Println("Snapshot data:")
  log.Printf("\tDBClusterSnapshotIdentifier: %v\n",
 *snapshot.DBClusterSnapshotIdentifier)
  log.Printf("\tARN: %v\n", *snapshot.DBClusterSnapshotArn)
  log.Printf("\tStatus: %v\n", *snapshot.Status)
  log.Printf("\tEngine: %v\n", *snapshot.Engine)
```

```
  log.Printf("\tEngine version: %v\n", *snapshot.EngineVersion)
  log.Printf("\tDBClusterIdentifier: %v\n", *snapshot.DBClusterIdentifier)
  log.Printf("\tSnapshotCreateTime: %v\n", *snapshot.SnapshotCreateTime)
  log.Println(strings.Repeat("-", 88))
 }
}

// Cleanup shows how to clean up a DB instance, DB cluster, and DB cluster
 parameter group.
// Before the DB cluster parameter group can be deleted, all associated DB
 instances and
// DB clusters must first be deleted.
func (scenario GetStartedClusters) Cleanup(dbInstance *types.DBInstance, cluster
 *types.DBCluster,
 parameterGroup *types.DBClusterParameterGroup) {

 if scenario.questioner.AskBool(
  "\nDo you want to delete the database instance, DB cluster, and parameter group
 (y/n)? ", "y") {
  log.Printf("Deleting database instance %v.\n",
 *dbInstance.DBInstanceIdentifier)
  err := scenario.dbClusters.DeleteInstance(*dbInstance.DBInstanceIdentifier)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleting database cluster %v.\n", *cluster.DBClusterIdentifier)
  err = scenario.dbClusters.DeleteDbCluster(*cluster.DBClusterIdentifier)
  if err != nil {
   panic(err)
  }
  log.Println(
   "Waiting for the DB instance and DB cluster to delete. This typically takes
 several minutes.")
  for dbInstance != nil || cluster != nil {
   scenario.helper.Pause(30)
   if dbInstance != nil {
    dbInstance, err =
 scenario.dbClusters.GetInstance(*dbInstance.DBInstanceIdentifier)
    if err != nil {
     panic(err)
    }
   }
   if cluster != nil {
    cluster, err = scenario.dbClusters.GetDbCluster(*cluster.DBClusterIdentifier)
```

```
    if err != nil {
     panic(err)
    }
   }
  }
  log.Printf("Deleting parameter group %v.",
 *parameterGroup.DBClusterParameterGroupName)
  err =
 scenario.dbClusters.DeleteParameterGroup(*parameterGroup.DBClusterParameterGroupName)
  if err != nil {
   panic(err)
  }
 }
}
```

Tentukan fungsi-fungsi yang dipanggil oleh skenario untuk mengelola tindakan Aurora.

```
type DbClusters struct {
 AuroraClient *rds.Client
}


// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  context.TODO(), &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
```

```
 }
}


// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 parameterGroupName string, parameterGroupFamily string, description string) (
 *types.DBClusterParameterGroup, error) {

 output, err :=
 clusters.AuroraClient.CreateDBClusterParameterGroup(context.TODO(),
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(parameterGroupName string) error
 {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(context.TODO(),
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

```
// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(parameterGroupName string, source
 string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(parameterGroupName string, params
 []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(context.TODO(),
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
```

```
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(clusterName string) (*types.DBCluster,
 error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(context.TODO(),
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified
 engine and
// engine version.
func (clusters *DbClusters) CreateDbCluster(clusterName string,
 parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(context.TODO(),
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
```

```
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(clusterName string) error {
 _, err := clusters.AuroraClient.DeleteDBCluster(context.TODO(),
 &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   true,
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(clusterName string,
 snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(context.TODO(),
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
```

```
  return output.DBClusterSnapshot, nil
 }
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(snapshotName string)
 (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(context.TODO(),
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(clusterName string,
 instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(context.TODO(),
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}
```

```
// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(instanceName string) (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(context.TODO(),
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(instanceName string) error {
 _, err := clusters.AuroraClient.DeleteDBInstance(context.TODO(),
 &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      true,
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

```
// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(engine string, parameterGroupFamily
 string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(context.TODO(),
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(engine string, engineVersion
 string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(context.TODO())
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
```

```
  }
 }
 return instances, err
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for Go .
  - CreateDBCluster
  - dibuatB ClusterParameterGroup
  - dibuatB ClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DihapusB ClusterParameterGroup
  - DeleteDBInstance
  - DijelaskanB ClusterParameterGroups
  - DijelaskanB ClusterParameters
  - DijelaskanB ClusterSnapshots
  - DescribeDBClusters
  - DijelaskanB EngineVersions
  - DescribeDBInstances
  - DescribeOrderableDB InstanceOptions
  - ModifyDB ClusterParameterGroup

Java

SDK for Java 2.x

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/**
```

```
 * Before running this Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * This example requires an AWS Secrets Manager secret that contains the
 * database credentials. If you do not create a
 * secret, this example will not work. For details, see:
 *
 * https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html
 *
 * This Java example performs the following tasks:
 *
 * 1. Gets available engine families for Amazon Aurora MySQL-Compatible Edition
 * by calling the DescribeDbEngineVersions(Engine='aurora-mysql') method.
 * 2. Selects an engine family and creates a custom DB cluster parameter group
 * by invoking the describeDBClusterParameters method.
 * 3. Gets the parameter groups by invoking the describeDBClusterParameterGroups
 * method.
 * 4. Gets parameters in the group by invoking the describeDBClusterParameters
 * method.
 * 5. Modifies the auto_increment_offset parameter by invoking the
 * modifyDbClusterParameterGroupRequest method.
 * 6. Gets and displays the updated parameters.
 * 7. Gets a list of allowed engine versions by invoking the
 * describeDbEngineVersions method.
 * 8. Creates an Aurora DB cluster database cluster that contains a MySQL
 * database.
 * 9. Waits for DB instance to be ready.
 * 10. Gets a list of instance classes available for the selected engine.
 * 11. Creates a database instance in the cluster.
 * 12. Waits for DB instance to be ready.
 * 13. Creates a snapshot.
 * 14. Waits for DB snapshot to be ready.
 * 15. Deletes the DB cluster.
 * 16. Deletes the DB cluster group.
 */
public class AuroraScenario {
    public static long sleepTime = 20;
```

```
   public static final String DASHES = new String(new char[80]).replace("\0",
"-");

   public static void main(String[] args) throws InterruptedException {
       final String usage = "\n" +
               "Usage:\n" +
               "    <dbClusterGroupName> <dbParameterGroupFamily>
<dbInstanceClusterIdentifier> <dbInstanceIdentifier> <dbName>
<dbSnapshotIdentifier><secretName>"
               +
               "Where:\n" +
               "    dbClusterGroupName - The name of the DB cluster parameter
group. \n" +
               "    dbParameterGroupFamily - The DB cluster parameter group
family name (for example, aurora-mysql5.7). \n"
               +
               "    dbInstanceClusterIdentifier - The instance cluster
identifier value.\n" +
               "    dbInstanceIdentifier - The database instance identifier.\n"
+
               "    dbName - The database name.\n" +
               "    dbSnapshotIdentifier - The snapshot identifier.\n" +
               "    secretName - The name of the AWS Secrets Manager secret that
contains the database credentials\"\n";
       ;

       if (args.length != 7) {
           System.out.println(usage);
           System.exit(1);
       }

       String dbClusterGroupName = args[0];
       String dbParameterGroupFamily = args[1];
       String dbInstanceClusterIdentifier = args[2];
       String dbInstanceIdentifier = args[3];
       String dbName = args[4];
       String dbSnapshotIdentifier = args[5];
       String secretName = args[6];

       // Retrieve the database credentials using AWS Secrets Manager.
       Gson gson = new Gson();
       User user = gson.fromJson(String.valueOf(getSecretValues(secretName)),
User.class);
       String username = user.getUsername();
```

```
        String userPassword = user.getPassword();

        Region region = Region.US_WEST_2;
        RdsClient rdsClient = RdsClient.builder()
                .region(region)
                .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon Aurora example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("1. Return a list of the available DB engines");
        describeDBEngines(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Create a custom parameter group");
        createDBClusterParameterGroup(rdsClient, dbClusterGroupName,
dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. Get the parameter group");
        describeDbClusterParameterGroups(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Get the parameters in the group");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, 0);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Modify the auto_increment_offset parameter");
        modifyDBClusterParas(rdsClient, dbClusterGroupName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Display the updated parameter value");
        describeDbClusterParameters(rdsClient, dbClusterGroupName, -1);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Get a list of allowed engine versions");
```

```
        getAllowedEngines(rdsClient, dbParameterGroupFamily);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Create an Aurora DB cluster database");
        String arnClusterVal = createDBCluster(rdsClient, dbClusterGroupName,
dbName, dbInstanceClusterIdentifier,
                username, userPassword);
        System.out.println("The ARN of the cluster is " + arnClusterVal);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Wait for DB instance to be ready");
        waitForInstanceReady(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Get a list of instance classes available for the
selected engine");
        String instanceClass = getListInstanceClasses(rdsClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Create a database instance in the cluster.");
        String clusterDBARN = createDBInstanceCluster(rdsClient,
dbInstanceIdentifier, dbInstanceClusterIdentifier,
                instanceClass);
        System.out.println("The ARN of the database is " + clusterDBARN);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Wait for DB instance to be ready");
        waitDBInstanceReady(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("13. Create a snapshot");
        createDBClusterSnapshot(rdsClient, dbInstanceClusterIdentifier,
dbSnapshotIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Wait for DB snapshot to be ready");
```

```
        waitForSnapshotReady(rdsClient, dbSnapshotIdentifier,
dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Delete the DB instance");
        deleteDatabaseInstance(rdsClient, dbInstanceIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("15. Delete the DB cluster");
        deleteCluster(rdsClient, dbInstanceClusterIdentifier);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("16. Delete the DB cluster group");
        deleteDBClusterGroup(rdsClient, dbClusterGroupName, clusterDBARN);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The Scenario has successfully completed.");
        System.out.println(DASHES);
        rdsClient.close();
    }

    private static SecretsManagerClient getSecretClient() {
        Region region = Region.US_WEST_2;
        return SecretsManagerClient.builder()
                .region(region)

.credentialsProvider(EnvironmentVariableCredentialsProvider.create())
                .build();
    }

    private static String getSecretValues(String secretName) {
        SecretsManagerClient secretClient = getSecretClient();
        GetSecretValueRequest valueRequest = GetSecretValueRequest.builder()
                .secretId(secretName)
                .build();

        GetSecretValueResponse valueResponse =
secretClient.getSecretValue(valueRequest);
        return valueResponse.secretString();
    }
```

```
   public static void deleteDBClusterGroup(RdsClient rdsClient, String
dbClusterGroupName, String clusterDBARN)
           throws InterruptedException {
       try {
           boolean isDataDel = false;
           boolean didFind;
           String instanceARN;

           // Make sure that the database has been deleted.
           while (!isDataDel) {
               DescribeDbInstancesResponse response =
rdsClient.describeDBInstances();
               List<DBInstance> instanceList = response.dbInstances();
               int listSize = instanceList.size();
               didFind = false;
               int index = 1;
               for (DBInstance instance : instanceList) {
                   instanceARN = instance.dbInstanceArn();
                   if (instanceARN.compareTo(clusterDBARN) == 0) {
                       System.out.println(clusterDBARN + " still exists");
                       didFind = true;
                   }
                   if ((index == listSize) && (!didFind)) {
                       // Went through the entire list and did not find the
database ARN.
                       isDataDel = true;
                   }
                   Thread.sleep(sleepTime * 1000);
                   index++;
               }
           }

           DeleteDbClusterParameterGroupRequest clusterParameterGroupRequest =
DeleteDbClusterParameterGroupRequest
                   .builder()
                   .dbClusterParameterGroupName(dbClusterGroupName)
                   .build();


rdsClient.deleteDBClusterParameterGroup(clusterParameterGroupRequest);
           System.out.println(dbClusterGroupName + " was deleted.");

       } catch (RdsException e) {
```

```
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void deleteCluster(RdsClient rdsClient, String
dbInstanceClusterIdentifier) {
        try {
            DeleteDbClusterRequest deleteDbClusterRequest =
DeleteDbClusterRequest.builder()
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .skipFinalSnapshot(true)
                    .build();

            rdsClient.deleteDBCluster(deleteDbClusterRequest);
            System.out.println(dbInstanceClusterIdentifier + " was deleted!");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void deleteDatabaseInstance(RdsClient rdsClient, String
dbInstanceIdentifier) {
        try {
            DeleteDbInstanceRequest deleteDbInstanceRequest =
DeleteDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .deleteAutomatedBackups(true)
                    .skipFinalSnapshot(true)
                    .build();

            DeleteDbInstanceResponse response =
rdsClient.deleteDBInstance(deleteDbInstanceRequest);
            System.out.println("The status of the database is " +
response.dbInstance().dbInstanceStatus());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
```

```
   public static void waitForSnapshotReady(RdsClient rdsClient, String
dbSnapshotIdentifier,
           String dbInstanceClusterIdentifier) {
       try {
           boolean snapshotReady = false;
           String snapshotReadyStr;
           System.out.println("Waiting for the snapshot to become available.");

           DescribeDbClusterSnapshotsRequest snapshotsRequest =
DescribeDbClusterSnapshotsRequest.builder()
                   .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                   .dbClusterIdentifier(dbInstanceClusterIdentifier)
                   .build();

           while (!snapshotReady) {
               DescribeDbClusterSnapshotsResponse response =
rdsClient.describeDBClusterSnapshots(snapshotsRequest);
               List<DBClusterSnapshot> snapshotList =
response.dbClusterSnapshots();
               for (DBClusterSnapshot snapshot : snapshotList) {
                   snapshotReadyStr = snapshot.status();
                   if (snapshotReadyStr.contains("available")) {
                       snapshotReady = true;
                   } else {
                       System.out.println(".");
                       Thread.sleep(sleepTime * 5000);
                   }
               }
           }

           System.out.println("The Snapshot is available!");

       } catch (RdsException | InterruptedException e) {
           System.out.println(e.getLocalizedMessage());
           System.exit(1);
       }
   }

   public static void createDBClusterSnapshot(RdsClient rdsClient, String
dbInstanceClusterIdentifier,
           String dbSnapshotIdentifier) {
       try {
           CreateDbClusterSnapshotRequest snapshotRequest =
CreateDbClusterSnapshotRequest.builder()
```

```
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .dbClusterSnapshotIdentifier(dbSnapshotIdentifier)
                    .build();

            CreateDbClusterSnapshotResponse response =
rdsClient.createDBClusterSnapshot(snapshotRequest);
            System.out.println("The Snapshot ARN is " +
response.dbClusterSnapshot().dbClusterSnapshotArn());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void waitDBInstanceReady(RdsClient rdsClient, String
dbInstanceIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbInstancesRequest instanceRequest =
DescribeDbInstancesRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .build();

            String endpoint = "";
            while (!instanceReady) {
                DescribeDbInstancesResponse response =
rdsClient.describeDBInstances(instanceRequest);
                List<DBInstance> instanceList = response.dbInstances();
                for (DBInstance instance : instanceList) {
                    instanceReadyStr = instance.dbInstanceStatus();
                    if (instanceReadyStr.contains("available")) {
                        endpoint = instance.endpoint().address();
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
            }
            System.out.println("Database instance is available! The connection
endpoint is " + endpoint);
```

```
        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String createDBInstanceCluster(RdsClient rdsClient,
            String dbInstanceIdentifier,
            String dbInstanceClusterIdentifier,
            String instanceClass) {
        try {
            CreateDbInstanceRequest instanceRequest =
CreateDbInstanceRequest.builder()
                    .dbInstanceIdentifier(dbInstanceIdentifier)
                    .dbClusterIdentifier(dbInstanceClusterIdentifier)
                    .engine("aurora-mysql")
                    .dbInstanceClass(instanceClass)
                    .build();

            CreateDbInstanceResponse response =
rdsClient.createDBInstance(instanceRequest);
            System.out.print("The status is " +
response.dbInstance().dbInstanceStatus());
            return response.dbInstance().dbInstanceArn();

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    public static String getListInstanceClasses(RdsClient rdsClient) {
        try {
            DescribeOrderableDbInstanceOptionsRequest optionsRequest =
DescribeOrderableDbInstanceOptionsRequest
                    .builder()
                    .engine("aurora-mysql")
                    .maxRecords(20)
                    .build();

            DescribeOrderableDbInstanceOptionsResponse response = rdsClient
                    .describeOrderableDBInstanceOptions(optionsRequest);
```

```
            List<OrderableDBInstanceOption> instanceOptions =
response.orderableDBInstanceOptions();
            String instanceClass = "";
            for (OrderableDBInstanceOption instanceOption : instanceOptions) {
                instanceClass = instanceOption.dbInstanceClass();
                System.out.println("The instance class is " +
instanceOption.dbInstanceClass());
                System.out.println("The engine version is " +
instanceOption.engineVersion());
            }
            return instanceClass;

        } catch (RdsException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        return "";
    }

    // Waits until the database instance is available.
    public static void waitForInstanceReady(RdsClient rdsClient, String
dbClusterIdentifier) {
        boolean instanceReady = false;
        String instanceReadyStr;
        System.out.println("Waiting for instance to become available.");
        try {
            DescribeDbClustersRequest instanceRequest =
DescribeDbClustersRequest.builder()
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .build();

            while (!instanceReady) {
                DescribeDbClustersResponse response =
rdsClient.describeDBClusters(instanceRequest);
                List<DBCluster> clusterList = response.dbClusters();
                for (DBCluster cluster : clusterList) {
                    instanceReadyStr = cluster.status();
                    if (instanceReadyStr.contains("available")) {
                        instanceReady = true;
                    } else {
                        System.out.print(".");
                        Thread.sleep(sleepTime * 1000);
                    }
                }
```

```
            }
            System.out.println("Database cluster is available!");

        } catch (RdsException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }

    public static String createDBCluster(RdsClient rdsClient, String
dbParameterGroupFamily, String dbName,
            String dbClusterIdentifier, String userName, String password) {
        try {
            CreateDbClusterRequest clusterRequest =
CreateDbClusterRequest.builder()
                    .databaseName(dbName)
                    .dbClusterIdentifier(dbClusterIdentifier)
                    .dbClusterParameterGroupName(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .masterUsername(userName)
                    .masterUserPassword(password)
                    .build();

            CreateDbClusterResponse response =
rdsClient.createDBCluster(clusterRequest);
            return response.dbCluster().dbClusterArn();

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
        return "";
    }

    // Get a list of allowed engine versions.
    public static void getAllowedEngines(RdsClient rdsClient, String
dbParameterGroupFamily) {
        try {
            DescribeDbEngineVersionsRequest versionsRequest =
DescribeDbEngineVersionsRequest.builder()
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .engine("aurora-mysql")
                    .build();
```

```
            DescribeDbEngineVersionsResponse response =
rdsClient.describeDBEngineVersions(versionsRequest);
            List<DBEngineVersion> dbEngines = response.dbEngineVersions();
            for (DBEngineVersion dbEngine : dbEngines) {
                System.out.println("The engine version is " +
dbEngine.engineVersion());
                System.out.println("The engine description is " +
dbEngine.dbEngineDescription());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    // Modify the auto_increment_offset parameter.
    public static void modifyDBClusterParas(RdsClient rdsClient, String
dClusterGroupName) {
        try {
            Parameter parameter1 = Parameter.builder()
                    .parameterName("auto_increment_offset")
                    .applyMethod("immediate")
                    .parameterValue("5")
                    .build();

            List<Parameter> paraList = new ArrayList<>();
            paraList.add(parameter1);
            ModifyDbClusterParameterGroupRequest groupRequest =
ModifyDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dClusterGroupName)
                    .parameters(paraList)
                    .build();

            ModifyDbClusterParameterGroupResponse response =
rdsClient.modifyDBClusterParameterGroup(groupRequest);
            System.out.println(
                    "The parameter group " +
response.dbClusterParameterGroupName() + " was successfully modified");

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
```

```
   }

   public static void describeDbClusterParameters(RdsClient rdsClient, String
dbCLusterGroupName, int flag) {
       try {
           DescribeDbClusterParametersRequest dbParameterGroupsRequest;
           if (flag == 0) {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .build();
           } else {
               dbParameterGroupsRequest =
DescribeDbClusterParametersRequest.builder()
                       .dbClusterParameterGroupName(dbCLusterGroupName)
                       .source("user")
                       .build();
           }

           DescribeDbClusterParametersResponse response = rdsClient
                   .describeDBClusterParameters(dbParameterGroupsRequest);
           List<Parameter> dbParameters = response.parameters();
           String paraName;
           for (Parameter para : dbParameters) {
               // Only print out information about either auto_increment_offset
or
               // auto_increment_increment.
               paraName = para.parameterName();
               if ((paraName.compareTo("auto_increment_offset") == 0)
                       || (paraName.compareTo("auto_increment_increment ") ==
0)) {
                   System.out.println("*** The parameter name is  " + paraName);
                   System.out.println("*** The parameter value is  " +
para.parameterValue());
                   System.out.println("*** The parameter data type is " +
para.dataType());
                   System.out.println("*** The parameter description is " +
para.description());
                   System.out.println("*** The parameter allowed values  is " +
para.allowedValues());
               }
           }

       } catch (RdsException e) {
```

```
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDbClusterParameterGroups(RdsClient rdsClient,
String dbClusterGroupName) {
        try {
            DescribeDbClusterParameterGroupsRequest groupsRequest =
DescribeDbClusterParameterGroupsRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .maxRecords(20)
                    .build();

            List<DBClusterParameterGroup> groups =
rdsClient.describeDBClusterParameterGroups(groupsRequest)
                    .dbClusterParameterGroups();
            for (DBClusterParameterGroup group : groups) {
                System.out.println("The group name is " +
group.dbClusterParameterGroupName());
                System.out.println("The group ARN is " +
group.dbClusterParameterGroupArn());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void createDBClusterParameterGroup(RdsClient rdsClient, String
dbClusterGroupName,
            String dbParameterGroupFamily) {
        try {
            CreateDbClusterParameterGroupRequest groupRequest =
CreateDbClusterParameterGroupRequest.builder()
                    .dbClusterParameterGroupName(dbClusterGroupName)
                    .dbParameterGroupFamily(dbParameterGroupFamily)
                    .description("Created by using the AWS SDK for Java")
                    .build();

            CreateDbClusterParameterGroupResponse response =
rdsClient.createDBClusterParameterGroup(groupRequest);
```

```
            System.out.println("The group name is " +
 response.dbClusterParameterGroup().dbClusterParameterGroupName());

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }

    public static void describeDBEngines(RdsClient rdsClient) {
        try {
            DescribeDbEngineVersionsRequest engineVersionsRequest =
 DescribeDbEngineVersionsRequest.builder()
                    .engine("aurora-mysql")
                    .defaultOnly(true)
                    .maxRecords(20)
                    .build();

            DescribeDbEngineVersionsResponse response =
 rdsClient.describeDBEngineVersions(engineVersionsRequest);
            List<DBEngineVersion> engines = response.dbEngineVersions();

            // Get all DBEngineVersion objects.
            for (DBEngineVersion engineOb : engines) {
                System.out.println("The name of the DB parameter group family for
 the database engine is "
                        + engineOb.dbParameterGroupFamily());
                System.out.println("The name of the database engine " +
 engineOb.engine());
                System.out.println("The version number of the database engine " +
 engineOb.engineVersion());
            }

        } catch (RdsException e) {
            System.out.println(e.getLocalizedMessage());
            System.exit(1);
        }
    }
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for Java 2.x .
  - CreateDBCluster

- dibuatB ClusterParameterGroup
- dibuatB ClusterSnapshot
- CreateDBInstance
- DeleteDBCluster
- DihapusB ClusterParameterGroup
- DeleteDBInstance
- DijelaskanB ClusterParameterGroups
- DijelaskanB ClusterParameters
- DijelaskanB ClusterSnapshots
- DescribeDBClusters
- DijelaskanB EngineVersions
- DescribeDBInstances
- DescribeOrderableDB InstanceOptions
- ModifyDB ClusterParameterGroup

Kotlin

SDK for Kotlin

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

```
/**
Before running this Kotlin code example, set up your development environment,
 including your credentials.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

This example requires an AWS Secrets Manager secret that contains the database
 credentials. If you do not create a
secret, this example will not work. For more details, see:
```

```
https://docs.aws.amazon.com/secretsmanager/latest/userguide/integrating_how-
services-use-secrets_RS.html

This Kotlin example performs the following tasks:

1. Returns a list of the available DB engines.
2. Creates a custom DB parameter group.
3. Gets the parameter groups.
4. Gets the parameters in the group.
5. Modifies the auto_increment_increment parameter.
6. Displays the updated parameter value.
7. Gets a list of allowed engine versions.
8. Creates an Aurora DB cluster database.
9. Waits for DB instance to be ready.
10. Gets a list of instance classes available for the selected engine.
11. Creates a database instance in the cluster.
12. Waits for the database instance in the cluster to be ready.
13. Creates a snapshot.
14. Waits for DB snapshot to be ready.
15. Deletes the DB instance.
16. Deletes the DB cluster.
17. Deletes the DB cluster group.
 */

var slTime: Long = 20
suspend fun main(args: Array<String>) {
    val usage = """
        Usage:
            <dbClusterGroupName> <dbParameterGroupFamily>
 <dbInstanceClusterIdentifier> <dbName> <dbSnapshotIdentifier> <secretName>
        Where:
            dbClusterGroupName - The database group name.
            dbParameterGroupFamily - The database parameter group name.
            dbInstanceClusterIdentifier - The database instance identifier.
            dbName -  The database name.
            dbSnapshotIdentifier - The snapshot identifier.
            secretName - The name of the AWS Secrets Manager secret that contains
 the database credentials.
    """

    if (args.size != 7) {
        println(usage)
        exitProcess(1)
```

```
    }

    val dbClusterGroupName = args[0]
    val dbParameterGroupFamily = args[1]
    val dbInstanceClusterIdentifier = args[2]
    val dbInstanceIdentifier = args[3]
    val dbName = args[4]
    val dbSnapshotIdentifier = args[5]
    val secretName = args[6]

    val gson = Gson()
    val user = gson.fromJson(getSecretValues(secretName).toString(),
User::class.java)
    val username = user.username
    val userPassword = user.password

    println("1. Return a list of the available DB engines")
    describeAuroraDBEngines()

    println("2. Create a custom parameter group")
    createDBClusterParameterGroup(dbClusterGroupName, dbParameterGroupFamily)

    println("3. Get the parameter group")
    describeDbClusterParameterGroups(dbClusterGroupName)

    println("4. Get the parameters in the group")
    describeDbClusterParameters(dbClusterGroupName, 0)

    println("5. Modify the auto_increment_offset parameter")
    modifyDBClusterParas(dbClusterGroupName)

    println("6. Display the updated parameter value")
    describeDbClusterParameters(dbClusterGroupName, -1)

    println("7. Get a list of allowed engine versions")
    getAllowedClusterEngines(dbParameterGroupFamily)

    println("8. Create an Aurora DB cluster database")
    val arnClusterVal = createDBCluster(dbClusterGroupName, dbName,
dbInstanceClusterIdentifier, username, userPassword)
    println("The ARN of the cluster is $arnClusterVal")

    println("9. Wait for DB instance to be ready")
    waitForClusterInstanceReady(dbInstanceClusterIdentifier)
```

```
    println("10. Get a list of instance classes available for the selected
 engine")
    val instanceClass = getListInstanceClasses()

    println("11. Create a database instance in the cluster.")
    val clusterDBARN = createDBInstanceCluster(dbInstanceIdentifier,
 dbInstanceClusterIdentifier, instanceClass)
    println("The ARN of the database is $clusterDBARN")

    println("12. Wait for DB instance to be ready")
    waitDBAuroraInstanceReady(dbInstanceIdentifier)

    println("13. Create a snapshot")
    createDBClusterSnapshot(dbInstanceClusterIdentifier, dbSnapshotIdentifier)

    println("14. Wait for DB snapshot to be ready")
    waitSnapshotReady(dbSnapshotIdentifier, dbInstanceClusterIdentifier)

    println("15. Delete the DB instance")
    deleteDBInstance(dbInstanceIdentifier)

    println("16. Delete the DB cluster")
    deleteCluster(dbInstanceClusterIdentifier)

    println("17. Delete the DB cluster group")
    if (clusterDBARN != null) {
        deleteDBClusterGroup(dbClusterGroupName, clusterDBARN)
    }
    println("The Scenario has successfully completed.")
}

@Throws(InterruptedException::class)
suspend fun deleteDBClusterGroup(dbClusterGroupName: String, clusterDBARN:
 String) {
    var isDataDel = false
    var didFind: Boolean
    var instanceARN: String

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        // Make sure that the database has been deleted.
        while (!isDataDel) {
            val response = rdsClient.describeDbInstances()
            val instanceList = response.dbInstances
```

```
            val listSize = instanceList?.size
            isDataDel = false
            didFind = false
            var index = 1
            if (instanceList != null) {
                for (instance in instanceList) {
                    instanceARN = instance.dbInstanceArn.toString()
                    if (instanceARN.compareTo(clusterDBARN) == 0) {
                        println("$clusterDBARN still exists")
                        didFind = true
                    }
                    if (index == listSize && !didFind) {
                        // Went through the entire list and did not find the
 database ARN.
                        isDataDel = true
                    }
                    delay(slTime * 1000)
                    index++
                }
            }
        }
        val clusterParameterGroupRequest = DeleteDbClusterParameterGroupRequest {
            dbClusterParameterGroupName = dbClusterGroupName
        }

        rdsClient.deleteDbClusterParameterGroup(clusterParameterGroupRequest)
        println("$dbClusterGroupName was deleted.")
    }
}

suspend fun deleteCluster(dbInstanceClusterIdentifier: String) {
    val deleteDbClusterRequest = DeleteDbClusterRequest {
        dbClusterIdentifier = dbInstanceClusterIdentifier
        skipFinalSnapshot = true
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        rdsClient.deleteDbCluster(deleteDbClusterRequest)
        println("$dbInstanceClusterIdentifier was deleted!")
    }
}

suspend fun deleteDBInstance(dbInstanceIdentifierVal: String) {
    val deleteDbInstanceRequest = DeleteDbInstanceRequest {
```

```
        dbInstanceIdentifier = dbInstanceIdentifierVal
        deleteAutomatedBackups = true
        skipFinalSnapshot = true
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.deleteDbInstance(deleteDbInstanceRequest)
        print("The status of the database is
 ${response.dbInstance?.dbInstanceStatus}")
    }
}

suspend fun waitSnapshotReady(dbSnapshotIdentifier: String?,
 dbInstanceClusterIdentifier: String?) {
    var snapshotReady = false
    var snapshotReadyStr: String
    println("Waiting for the snapshot to become available.")

    val snapshotsRequest = DescribeDbClusterSnapshotsRequest {
        dbClusterSnapshotIdentifier = dbSnapshotIdentifier
        dbClusterIdentifier = dbInstanceClusterIdentifier
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!snapshotReady) {
            val response = rdsClient.describeDbClusterSnapshots(snapshotsRequest)
            val snapshotList = response.dbClusterSnapshots
            if (snapshotList != null) {
                for (snapshot in snapshotList) {
                    snapshotReadyStr = snapshot.status.toString()
                    if (snapshotReadyStr.contains("available")) {
                        snapshotReady = true
                    } else {
                        println(".")
                        delay(slTime * 5000)
                    }
                }
            }
        }
    }
    println("The Snapshot is available!")
}
```

```
suspend fun createDBClusterSnapshot(dbInstanceClusterIdentifier: String?,
 dbSnapshotIdentifier: String?) {
    val snapshotRequest = CreateDbClusterSnapshotRequest {
        dbClusterIdentifier = dbInstanceClusterIdentifier
        dbClusterSnapshotIdentifier = dbSnapshotIdentifier
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterSnapshot(snapshotRequest)
        println("The Snapshot ARN is
 ${response.dbClusterSnapshot?.dbClusterSnapshotArn}")
    }
}

suspend fun waitDBAuroraInstanceReady(dbInstanceIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")
    val instanceRequest = DescribeDbInstancesRequest {
        dbInstanceIdentifier = dbInstanceIdentifierVal
    }

    var endpoint = ""
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
            val response = rdsClient.describeDbInstances(instanceRequest)
            response.dbInstances?.forEach { instance ->
                instanceReadyStr = instance.dbInstanceStatus.toString()
                if (instanceReadyStr.contains("available")) {
                    endpoint = instance.endpoint?.address.toString()
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database instance is available! The connection endpoint is
 $endpoint")
}

suspend fun createDBInstanceCluster(dbInstanceIdentifierVal: String?,
 dbInstanceClusterIdentifierVal: String?, instanceClassVal: String?): String? {
```

```
    val instanceRequest = CreateDbInstanceRequest {
        dbInstanceIdentifier = dbInstanceIdentifierVal
        dbClusterIdentifier = dbInstanceClusterIdentifierVal
        engine = "aurora-mysql"
        dbInstanceClass = instanceClassVal
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbInstance(instanceRequest)
        print("The status is ${response.dbInstance?.dbInstanceStatus}")
        return response.dbInstance?.dbInstanceArn
    }
}

suspend fun getListInstanceClasses(): String {
    val optionsRequest = DescribeOrderableDbInstanceOptionsRequest {
        engine = "aurora-mysql"
        maxRecords = 20
    }
    var instanceClass = ""
    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeOrderableDbInstanceOptions(optionsRequest)
        response.orderableDbInstanceOptions?.forEach { instanceOption ->
            instanceClass = instanceOption.dbInstanceClass.toString()
            println("The instance class is ${instanceOption.dbInstanceClass}")
            println("The engine version is ${instanceOption.engineVersion}")
        }
    }
    return instanceClass
}

// Waits until the database instance is available.
suspend fun waitForClusterInstanceReady(dbClusterIdentifierVal: String?) {
    var instanceReady = false
    var instanceReadyStr: String
    println("Waiting for instance to become available.")

    val instanceRequest = DescribeDbClustersRequest {
        dbClusterIdentifier = dbClusterIdentifierVal
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        while (!instanceReady) {
```

```
            val response = rdsClient.describeDbClusters(instanceRequest)
            response.dbClusters?.forEach { cluster ->
                instanceReadyStr = cluster.status.toString()
                if (instanceReadyStr.contains("available")) {
                    instanceReady = true
                } else {
                    print(".")
                    delay(sleepTime * 1000)
                }
            }
        }
    }
    println("Database cluster is available!")
}

suspend fun createDBCluster(dbParameterGroupFamilyVal: String?, dbName: String?,
 dbClusterIdentifierVal: String?, userName: String?, password: String?): String?
 {
    val clusterRequest = CreateDbClusterRequest {
        databaseName = dbName
        dbClusterIdentifier = dbClusterIdentifierVal
        dbClusterParameterGroupName = dbParameterGroupFamilyVal
        engine = "aurora-mysql"
        masterUsername = userName
        masterUserPassword = password
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbCluster(clusterRequest)
        return response.dbCluster?.dbClusterArn
    }
}

// Get a list of allowed engine versions.
suspend fun getAllowedClusterEngines(dbParameterGroupFamilyVal: String?) {
    val versionsRequest = DescribeDbEngineVersionsRequest {
        dbParameterGroupFamily = dbParameterGroupFamilyVal
        engine = "aurora-mysql"
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(versionsRequest)
        response.dbEngineVersions?.forEach { dbEngine ->
            println("The engine version is ${dbEngine.engineVersion}")
```

```
            println("The engine description is ${dbEngine.dbEngineDescription}")
        }
    }
}

// Modify the auto_increment_offset parameter.
suspend fun modifyDBClusterParas(dClusterGroupName: String?) {
    val parameter1 = Parameter {
        parameterName = "auto_increment_offset"
        applyMethod = ApplyMethod.fromValue("immediate")
        parameterValue = "5"
    }

    val paraList = ArrayList<Parameter>()
    paraList.add(parameter1)
    val groupRequest = ModifyDbClusterParameterGroupRequest {
        dbClusterParameterGroupName = dClusterGroupName
        parameters = paraList
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.modifyDbClusterParameterGroup(groupRequest)
        println("The parameter group ${response.dbClusterParameterGroupName} was
 successfully modified")
    }
}

suspend fun describeDbClusterParameters(dbCLusterGroupName: String?, flag: Int) {
    val dbParameterGroupsRequest: DescribeDbClusterParametersRequest
    dbParameterGroupsRequest = if (flag == 0) {
        DescribeDbClusterParametersRequest {
            dbClusterParameterGroupName = dbCLusterGroupName
        }
    } else {
        DescribeDbClusterParametersRequest {
            dbClusterParameterGroupName = dbCLusterGroupName
            source = "user"
        }
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response =
 rdsClient.describeDbClusterParameters(dbParameterGroupsRequest)
        response.parameters?.forEach { para ->
```

```
            // Only print out information about either auto_increment_offset or
 auto_increment_increment.
            val paraName = para.parameterName
            if (paraName != null) {
                if (paraName.compareTo("auto_increment_offset") == 0 ||
 paraName.compareTo("auto_increment_increment ") == 0) {
                    println("*** The parameter name is  $paraName")
                    println("*** The parameter value is  ${para.parameterValue}")
                    println("*** The parameter data type is ${para.dataType}")
                    println("*** The parameter description is
 ${para.description}")
                    println("*** The parameter allowed values  is
 ${para.allowedValues}")
                }
            }
        }
    }
}

suspend fun describeDbClusterParameterGroups(dbClusterGroupName: String?) {
    val groupsRequest = DescribeDbClusterParameterGroupsRequest {
        dbClusterParameterGroupName = dbClusterGroupName
        maxRecords = 20
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbClusterParameterGroups(groupsRequest)
        response.dbClusterParameterGroups?.forEach { group ->
            println("The group name is ${group.dbClusterParameterGroupName}")
            println("The group ARN is ${group.dbClusterParameterGroupArn}")
        }
    }
}

suspend fun createDBClusterParameterGroup(dbClusterGroupNameVal: String?,
 dbParameterGroupFamilyVal: String?) {
    val groupRequest = CreateDbClusterParameterGroupRequest {
        dbClusterParameterGroupName = dbClusterGroupNameVal
        dbParameterGroupFamily = dbParameterGroupFamilyVal
        description = "Created by using the AWS SDK for Kotlin"
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.createDbClusterParameterGroup(groupRequest)
```

```
        println("The group name is
 ${response.dbClusterParameterGroup?.dbClusterParameterGroupName}")
    }
}

suspend fun describeAuroraDBEngines() {
    val engineVersionsRequest = DescribeDbEngineVersionsRequest {
        engine = "aurora-mysql"
        defaultOnly = true
        maxRecords = 20
    }

    RdsClient { region = "us-west-2" }.use { rdsClient ->
        val response = rdsClient.describeDbEngineVersions(engineVersionsRequest)
        response.dbEngineVersions?.forEach { engineOb ->
            println("The name of the DB parameter group family for the database
 engine is ${engineOb.dbParameterGroupFamily}")
            println("The name of the database engine ${engineOb.engine}")
            println("The version number of the database engine
 ${engineOb.engineVersion}")
        }
    }
}
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK For Kotlin.
  - CreateDBCluster
  - dibuatB ClusterParameterGroup
  - dibuatB ClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DihapusB ClusterParameterGroup
  - DeleteDBInstance
  - DijelaskanB ClusterParameterGroups
  - DijelaskanB ClusterParameters
  - DijelaskanB ClusterSnapshots
  - DescribeDBClusters

- DijelaskanB EngineVersions
- DescribeDBInstances
- DescribeOrderableDB InstanceOptions
- ModifyDB ClusterParameterGroup

Python

SDK for Python (Boto3)

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara pengaturan dan menjalankannya di Repositori Contoh Kode AWS.

Jalankan skenario interaktif di prompt perintah.

```
class AuroraClusterScenario:
    """Runs a scenario that shows how to get started using Aurora DB clusters."""

    def __init__(self, aurora_wrapper):
        """
        :param aurora_wrapper: An object that wraps Aurora DB cluster actions.
        """
        self.aurora_wrapper = aurora_wrapper

    def create_parameter_group(self, db_engine, parameter_group_name):
        """
        Shows how to get available engine versions for a specified database
 engine and
        create a DB cluster parameter group that is compatible with a selected
 engine family.

        :param db_engine: The database engine to use as a basis.
        :param parameter_group_name: The name given to the newly created
 parameter group.
        :return: The newly created parameter group.
        """
        print(
```

```
            f"Checking for an existing DB cluster parameter group named
{parameter_group_name}."
        )
        parameter_group =
self.aurora_wrapper.get_parameter_group(parameter_group_name)
        if parameter_group is None:
            print(f"Getting available database engine versions for {db_engine}.")
            engine_versions = self.aurora_wrapper.get_engine_versions(db_engine)
            families = list({ver["DBParameterGroupFamily"] for ver in
engine_versions})
            family_index = q.choose("Which family do you want to use? ",
families)
            print(f"Creating a DB cluster parameter group.")
            self.aurora_wrapper.create_parameter_group(
                parameter_group_name, families[family_index], "Example parameter
group."
            )
            parameter_group = self.aurora_wrapper.get_parameter_group(
                parameter_group_name
            )
        print(f"Parameter group
{parameter_group['DBClusterParameterGroupName']}:")
        pp(parameter_group)
        print("-" * 88)
        return parameter_group

    def set_user_parameters(self, parameter_group_name):
        """
        Shows how to get the parameters contained in a custom parameter group and
        update some of the parameter values in the group.

        :param parameter_group_name: The name of the parameter group to query and
modify.
        """
        print("Let's set some parameter values in your parameter group.")
        auto_inc_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, name_prefix="auto_increment"
        )
        update_params = []
        for auto_inc in auto_inc_parameters:
            if auto_inc["IsModifiable"] and auto_inc["DataType"] == "integer":
                print(f"The {auto_inc['ParameterName']} parameter is described
as:")
                print(f"\t{auto_inc['Description']}")
```

```
                param_range = auto_inc["AllowedValues"].split("-")
                auto_inc["ParameterValue"] = str(
                    q.ask(
                        f"Enter a value between {param_range[0]} and
{param_range[1]}: ",
                        q.is_int,
                        q.in_range(int(param_range[0]), int(param_range[1])),
                    )
                )
                update_params.append(auto_inc)
        self.aurora_wrapper.update_parameters(parameter_group_name,
update_params)
        print(
            "You can get a list of parameters you've set by specifying a source
of 'user'."
        )
        user_parameters = self.aurora_wrapper.get_parameters(
            parameter_group_name, source="user"
        )
        pp(user_parameters)
        print("-" * 88)

    def create_cluster(self, cluster_name, db_engine, db_name, parameter_group):
        """
        Shows how to create an Aurora DB cluster that contains a database of a
specified
        type. The database is also configured to use a custom DB cluster
parameter group.

        :param cluster_name: The name given to the newly created DB cluster.
        :param db_engine: The engine of the created database.
        :param db_name: The name given to the created database.
        :param parameter_group: The parameter group that is associated with the
DB cluster.
        :return: The newly created DB cluster.
        """
        print("Checking for an existing DB cluster.")
        cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
        if cluster is None:
            admin_username = q.ask(
                "Enter an administrator user name for the database: ",
q.non_empty
            )
            admin_password = q.ask(
```

```
                "Enter a password for the administrator (at least 8 characters):
",
                q.non_empty,
            )
            engine_versions = self.aurora_wrapper.get_engine_versions(
                db_engine, parameter_group["DBParameterGroupFamily"]
            )
            engine_choices = [ver["EngineVersion"] for ver in engine_versions]
            print("The available engines for your parameter group are:")
            engine_index = q.choose("Which engine do you want to use? ",
engine_choices)
            print(
                f"Creating DB cluster {cluster_name} and database {db_name}.\n"
                f"The DB cluster is configured to use\n"
                f"your custom parameter group
{parameter_group['DBClusterParameterGroupName']}\n"
                f"and selected engine {engine_choices[engine_index]}.\n"
                f"This typically takes several minutes."
            )
            cluster = self.aurora_wrapper.create_db_cluster(
                cluster_name,
                parameter_group["DBClusterParameterGroupName"],
                db_name,
                db_engine,
                engine_choices[engine_index],
                admin_username,
                admin_password,
            )
            while cluster.get("Status") != "available":
                wait(30)
                cluster = self.aurora_wrapper.get_db_cluster(cluster_name)
            print("Cluster created and available.\n")
        print("Cluster data:")
        pp(cluster)
        print("-" * 88)
        return cluster

    def create_instance(self, cluster):
        """
        Shows how to create a DB instance in an existing Aurora DB cluster. A new
DB cluster
        contains no DB instances, so you must add one. The first DB instance that
is added
        to a DB cluster defaults to a read-write DB instance.
```

```
        :param cluster: The DB cluster where the DB instance is added.
        :return: The newly created DB instance.
        """
        print("Checking for an existing database instance.")
        cluster_name = cluster["DBClusterIdentifier"]
        db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        if db_inst is None:
            print("Let's create a database instance in your DB cluster.")
            print("First, choose a DB instance type:")
            inst_opts = self.aurora_wrapper.get_orderable_instances(
                cluster["Engine"], cluster["EngineVersion"]
            )
            inst_choices = list({opt["DBInstanceClass"] for opt in inst_opts})
            inst_index = q.choose(
                "Which DB instance class do you want to use? ", inst_choices
            )
            print(
                f"Creating a database instance. This typically takes several
minutes."
            )
            db_inst = self.aurora_wrapper.create_instance_in_cluster(
                cluster_name, cluster_name, cluster["Engine"],
inst_choices[inst_index]
            )
            while db_inst.get("DBInstanceStatus") != "available":
                wait(30)
                db_inst = self.aurora_wrapper.get_db_instance(cluster_name)
        print("Instance data:")
        pp(db_inst)
        print("-" * 88)
        return db_inst

    @staticmethod
    def display_connection(cluster):
        """
        Displays connection information about an Aurora DB cluster and tips on
how to
        connect to it.

        :param cluster: The DB cluster to display.
        """
        print(
```

```
            "You can now connect to your database using your favorite MySql
 client.\n"
            "One way to connect is by using the 'mysql' shell on an Amazon EC2
 instance\n"
            "that is running in the same VPC as your database cluster. Pass the
 endpoint,\n"
            "port, and administrator user name to 'mysql' and enter your password
\n"
            "when prompted:\n"
        )
        print(
            f"\n\tmysql -h {cluster['Endpoint']} -P {cluster['Port']} -u
 {cluster['MasterUsername']} -p\n"
        )
        print(
            "For more information, see the User Guide for Aurora:\n"
            "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora
        )
        print("-" * 88)

    def create_snapshot(self, cluster_name):
        """
        Shows how to create a DB cluster snapshot and wait until it's available.

        :param cluster_name: The name of a DB cluster to snapshot.
        """
        if q.ask(
            "Do you want to create a snapshot of your DB cluster (y/n)? ",
 q.is_yesno
        ):
            snapshot_id = f"{cluster_name}-{uuid.uuid4()}"
            print(
                f"Creating a snapshot named {snapshot_id}. This typically takes a
 few minutes."
            )
            snapshot = self.aurora_wrapper.create_cluster_snapshot(
                snapshot_id, cluster_name
            )
            while snapshot.get("Status") != "available":
                wait(30)
                snapshot = self.aurora_wrapper.get_cluster_snapshot(snapshot_id)
            pp(snapshot)
            print("-" * 88)
```

```
   def cleanup(self, db_inst, cluster, parameter_group):
       """
       Shows how to clean up a DB instance, DB cluster, and DB cluster parameter
group.
       Before the DB cluster parameter group can be deleted, all associated DB
instances and
       DB clusters must first be deleted.

       :param db_inst: The DB instance to delete.
       :param cluster: The DB cluster to delete.
       :param parameter_group: The DB cluster parameter group to delete.
       """
       cluster_name = cluster["DBClusterIdentifier"]
       parameter_group_name = parameter_group["DBClusterParameterGroupName"]
       if q.ask(
           "\nDo you want to delete the database instance, DB cluster, and
parameter "
           "group (y/n)? ",
           q.is_yesno,
       ):
           print(f"Deleting database instance
{db_inst['DBInstanceIdentifier']}.")

self.aurora_wrapper.delete_db_instance(db_inst["DBInstanceIdentifier"])
           print(f"Deleting database cluster {cluster_name}.")
           self.aurora_wrapper.delete_db_cluster(cluster_name)
           print(
               "Waiting for the DB instance and DB cluster to delete.\n"
               "This typically takes several minutes."
           )
           while db_inst is not None or cluster is not None:
               wait(30)
               if db_inst is not None:
                   db_inst = self.aurora_wrapper.get_db_instance(
                       db_inst["DBInstanceIdentifier"]
                   )
               if cluster is not None:
                   cluster = self.aurora_wrapper.get_db_cluster(
                       cluster["DBClusterIdentifier"]
                   )
           print(f"Deleting parameter group {parameter_group_name}.")
           self.aurora_wrapper.delete_parameter_group(parameter_group_name)
```

```
    def run_scenario(self, db_engine, parameter_group_name, cluster_name,
 db_name):
        print("-" * 88)
        print(
            "Welcome to the Amazon Relational Database Service (Amazon RDS) get
 started\n"
            "with Aurora DB clusters demo."
        )
        print("-" * 88)

        parameter_group = self.create_parameter_group(db_engine,
 parameter_group_name)
        self.set_user_parameters(parameter_group_name)
        cluster = self.create_cluster(cluster_name, db_engine, db_name,
 parameter_group)
        wait(5)
        db_inst = self.create_instance(cluster)
        self.display_connection(cluster)
        self.create_snapshot(cluster_name)
        self.cleanup(db_inst, cluster, parameter_group)

        print("\nThanks for watching!")
        print("-" * 88)


if __name__ == "__main__":
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    try:
        scenario = AuroraClusterScenario(AuroraWrapper.from_client())
        scenario.run_scenario(
            "aurora-mysql",
            "doc-example-cluster-parameter-group",
            "doc-example-aurora",
            "docexampledb",
        )
    except Exception:
        logging.exception("Something went wrong with the demo.")
```

Tentukan fungsi-fungsi yang dipanggil oleh skenario untuk mengelola tindakan Aurora.

```
class AuroraWrapper:
    """Encapsulates Aurora DB cluster actions."""
```

```
    def __init__(self, rds_client):
        """
        :param rds_client: A Boto3 Amazon Relational Database Service (Amazon
RDS) client.
        """
        self.rds_client = rds_client

    @classmethod
    def from_client(cls):
        """
        Instantiates this class from a Boto3 client.
        """
        rds_client = boto3.client("rds")
        return cls(rds_client)


    def get_parameter_group(self, parameter_group_name):
        """
        Gets a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to retrieve.
        :return: The requested parameter group.
        """
        try:
            response = self.rds_client.describe_db_cluster_parameter_groups(
                DBClusterParameterGroupName=parameter_group_name
            )
            parameter_group = response["DBClusterParameterGroups"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBParameterGroupNotFound":
                logger.info("Parameter group %s does not exist.",
parameter_group_name)
            else:
                logger.error(
                    "Couldn't get parameter group %s. Here's why: %s: %s",
                    parameter_group_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return parameter_group
```

```
    def create_parameter_group(
        self, parameter_group_name, parameter_group_family, description
    ):
        """
        Creates a DB cluster parameter group that is based on the specified
parameter group
        family.

        :param parameter_group_name: The name of the newly created parameter
group.
        :param parameter_group_family: The family that is used as the basis of
the new
                                       parameter group.
        :param description: A description given to the parameter group.
        :return: Data about the newly created parameter group.
        """
        try:
            response = self.rds_client.create_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                DBParameterGroupFamily=parameter_group_family,
                Description=description,
            )
        except ClientError as err:
            logger.error(
                "Couldn't create parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def delete_parameter_group(self, parameter_group_name):
        """
        Deletes a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to delete.
        :return: Data about the parameter group.
        """
        try:
            response = self.rds_client.delete_db_cluster_parameter_group(
```

```
                DBClusterParameterGroupName=parameter_group_name
            )
        except ClientError as err:
            logger.error(
                "Couldn't delete parameter group %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_parameters(self, parameter_group_name, name_prefix="", source=None):
        """
        Gets the parameters that are contained in a DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to query.
        :param name_prefix: When specified, the retrieved list of parameters is
filtered
                            to contain only parameters that start with this
prefix.
        :param source: When specified, only parameters from this source are
retrieved.
                       For example, a source of 'user' retrieves only parameters
that
                       were set by a user.
        :return: The list of requested parameters.
        """
        try:
            kwargs = {"DBClusterParameterGroupName": parameter_group_name}
            if source is not None:
                kwargs["Source"] = source
            parameters = []
            paginator =
self.rds_client.get_paginator("describe_db_cluster_parameters")
            for page in paginator.paginate(**kwargs):
                parameters += [
                    p
                    for p in page["Parameters"]
                    if p["ParameterName"].startswith(name_prefix)
                ]
        except ClientError as err:
```

```
            logger.error(
                "Couldn't get parameters for %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return parameters


    def update_parameters(self, parameter_group_name, update_parameters):
        """
        Updates parameters in a custom DB cluster parameter group.

        :param parameter_group_name: The name of the parameter group to update.
        :param update_parameters: The parameters to update in the group.
        :return: Data about the modified parameter group.
        """
        try:
            response = self.rds_client.modify_db_cluster_parameter_group(
                DBClusterParameterGroupName=parameter_group_name,
                Parameters=update_parameters,
            )
        except ClientError as err:
            logger.error(
                "Couldn't update parameters in %s. Here's why: %s: %s",
                parameter_group_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response


    def get_db_cluster(self, cluster_name):
        """
        Gets data about an Aurora DB cluster.

        :param cluster_name: The name of the DB cluster to retrieve.
        :return: The retrieved DB cluster.
        """
        try:
```

```
            response = self.rds_client.describe_db_clusters(
                DBClusterIdentifier=cluster_name
            )
            cluster = response["DBClusters"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBClusterNotFoundFault":
                logger.info("Cluster %s does not exist.", cluster_name)
            else:
                logger.error(
                    "Couldn't verify the existence of DB cluster %s. Here's why:
%s: %s",
                    cluster_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return cluster


    def create_db_cluster(
        self,
        cluster_name,
        parameter_group_name,
        db_name,
        db_engine,
        db_engine_version,
        admin_name,
        admin_password,
    ):
        """
        Creates a DB cluster that is configured to use the specified parameter
group.
        The newly created DB cluster contains a database that uses the specified
engine and
        engine version.

        :param cluster_name: The name of the DB cluster to create.
        :param parameter_group_name: The name of the parameter group to associate
with
                                     the DB cluster.
        :param db_name: The name of the database to create.
        :param db_engine: The database engine of the database that is created,
such as MySql.
```

```
        :param db_engine_version: The version of the database engine.
        :param admin_name: The user name of the database administrator.
        :param admin_password: The password of the database administrator.
        :return: The newly created DB cluster.
        """
        try:
            response = self.rds_client.create_db_cluster(
                DatabaseName=db_name,
                DBClusterIdentifier=cluster_name,
                DBClusterParameterGroupName=parameter_group_name,
                Engine=db_engine,
                EngineVersion=db_engine_version,
                MasterUsername=admin_name,
                MasterUserPassword=admin_password,
            )
            cluster = response["DBCluster"]
        except ClientError as err:
            logger.error(
                "Couldn't create database %s. Here's why: %s: %s",
                db_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return cluster


    def delete_db_cluster(self, cluster_name):
        """
        Deletes a DB cluster.

        :param cluster_name: The name of the DB cluster to delete.
        """
        try:
            self.rds_client.delete_db_cluster(
                DBClusterIdentifier=cluster_name, SkipFinalSnapshot=True
            )
            logger.info("Deleted DB cluster %s.", cluster_name)
        except ClientError:
            logger.exception("Couldn't delete DB cluster %s.", cluster_name)
            raise
```

```
    def create_cluster_snapshot(self, snapshot_id, cluster_id):
        """
        Creates a snapshot of a DB cluster.

        :param snapshot_id: The ID to give the created snapshot.
        :param cluster_id: The DB cluster to snapshot.
        :return: Data about the newly created snapshot.
        """
        try:
            response = self.rds_client.create_db_cluster_snapshot(
                DBClusterSnapshotIdentifier=snapshot_id,
DBClusterIdentifier=cluster_id
            )
            snapshot = response["DBClusterSnapshot"]
        except ClientError as err:
            logger.error(
                "Couldn't create snapshot of %s. Here's why: %s: %s",
                cluster_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return snapshot


    def get_cluster_snapshot(self, snapshot_id):
        """
        Gets a DB cluster snapshot.

        :param snapshot_id: The ID of the snapshot to retrieve.
        :return: The retrieved snapshot.
        """
        try:
            response = self.rds_client.describe_db_cluster_snapshots(
                DBClusterSnapshotIdentifier=snapshot_id
            )
            snapshot = response["DBClusterSnapshots"][0]
        except ClientError as err:
            logger.error(
                "Couldn't get DB cluster snapshot %s. Here's why: %s: %s",
                snapshot_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
```

```
            )
            raise
        else:
            return snapshot


    def create_instance_in_cluster(
        self, instance_id, cluster_id, db_engine, instance_class
    ):
        """
        Creates a database instance in an existing DB cluster. The first database
that is
        created defaults to a read-write DB instance.

        :param instance_id: The ID to give the newly created DB instance.
        :param cluster_id: The ID of the DB cluster where the DB instance is
created.
        :param db_engine: The database engine of a database to create in the DB
instance.
                          This must be compatible with the configured parameter
group
                          of the DB cluster.
        :param instance_class: The DB instance class for the newly created DB
instance.
        :return: Data about the newly created DB instance.
        """
        try:
            response = self.rds_client.create_db_instance(
                DBInstanceIdentifier=instance_id,
                DBClusterIdentifier=cluster_id,
                Engine=db_engine,
                DBInstanceClass=instance_class,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't create DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

```
   def get_engine_versions(self, engine, parameter_group_family=None):
       """
       Gets database engine versions that are available for the specified engine
       and parameter group family.

       :param engine: The database engine to look up.
       :param parameter_group_family: When specified, restricts the returned
list of
                                      engine versions to those that are
compatible with
                                      this parameter group family.
       :return: The list of database engine versions.
       """
       try:
           kwargs = {"Engine": engine}
           if parameter_group_family is not None:
               kwargs["DBParameterGroupFamily"] = parameter_group_family
           response = self.rds_client.describe_db_engine_versions(**kwargs)
           versions = response["DBEngineVersions"]
       except ClientError as err:
           logger.error(
               "Couldn't get engine versions for %s. Here's why: %s: %s",
               engine,
               err.response["Error"]["Code"],
               err.response["Error"]["Message"],
           )
           raise
       else:
           return versions


   def get_orderable_instances(self, db_engine, db_engine_version):
       """
       Gets DB instance options that can be used to create DB instances that are
       compatible with a set of specifications.

       :param db_engine: The database engine that must be supported by the DB
instance.
       :param db_engine_version: The engine version that must be supported by
the DB instance.
       :return: The list of DB instance options that can be used to create a
compatible DB instance.
```

```
        """
        try:
            inst_opts = []
            paginator = self.rds_client.get_paginator(
                "describe_orderable_db_instance_options"
            )
            for page in paginator.paginate(
                Engine=db_engine, EngineVersion=db_engine_version
            ):
                inst_opts += page["OrderableDBInstanceOptions"]
        except ClientError as err:
            logger.error(
                "Couldn't get orderable DB instances. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return inst_opts


    def get_db_instance(self, instance_id):
        """
        Gets data about a DB instance.

        :param instance_id: The ID of the DB instance to retrieve.
        :return: The retrieved DB instance.
        """
        try:
            response = self.rds_client.describe_db_instances(
                DBInstanceIdentifier=instance_id
            )
            db_inst = response["DBInstances"][0]
        except ClientError as err:
            if err.response["Error"]["Code"] == "DBInstanceNotFound":
                logger.info("Instance %s does not exist.", instance_id)
            else:
                logger.error(
                    "Couldn't get DB instance %s. Here's why: %s: %s",
                    instance_id,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
```

```
        else:
            return db_inst


    def delete_db_instance(self, instance_id):
        """
        Deletes a DB instance.

        :param instance_id: The ID of the DB instance to delete.
        :return: Data about the deleted DB instance.
        """
        try:
            response = self.rds_client.delete_db_instance(
                DBInstanceIdentifier=instance_id,
                SkipFinalSnapshot=True,
                DeleteAutomatedBackups=True,
            )
            db_inst = response["DBInstance"]
        except ClientError as err:
            logger.error(
                "Couldn't delete DB instance %s. Here's why: %s: %s",
                instance_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return db_inst
```

- Lihat detail API di topik-topik berikut dalam Referensi API AWS SDK for Python (Boto3).
  - CreateDBCluster
  - dibuatB ClusterParameterGroup
  - dibuatB ClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DihapusB ClusterParameterGroup

- DeleteDBInstance
- DijelaskanB ClusterParameterGroups
- DijelaskanB ClusterParameters
- DijelaskanB ClusterSnapshots
- DescribeDBClusters
- DijelaskanB EngineVersions
- DescribeDBInstances
- DescribeOrderableDB InstanceOptions
- ModifyDB ClusterParameterGroup

Rust

SDK for Rust

> **Note**
>
> Ada lebih banyak tentang GitHub. Temukan contoh lengkapnya dan pelajari cara mengatur dan menjalankannya di Repositori Contoh Kode AWS.

Pustaka yang berisi fungsi khusus skenario untuk skenario Aurora.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use phf::{phf_set, Set};
use secrecy::SecretString;
use std::{collections::HashMap, fmt::Display, time::Duration};

use aws_sdk_rds::{
    error::ProvideErrorMetadata,

 operation::create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
    types::{DbCluster, DbClusterParameterGroup, DbClusterSnapshot, DbInstance,
 Parameter},
};
use sdk_examples_test_utils::waiter::Waiter;
use tracing::{info, trace, warn};
```

```
const DB_ENGINE: &str = "aurora-mysql";
const DB_CLUSTER_PARAMETER_GROUP_NAME: &str =
 "RustSDKCodeExamplesDBParameterGroup";
const DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION: &str =
    "Parameter Group created by Rust SDK Code Example";
const DB_CLUSTER_IDENTIFIER: &str = "RustSDKCodeExamplesDBCluster";
const DB_INSTANCE_IDENTIFIER: &str = "RustSDKCodeExamplesDBInstance";

static FILTER_PARAMETER_NAMES: Set<&'static str> = phf_set! {
    "auto_increment_offset",
    "auto_increment_increment",
};

#[derive(Debug, PartialEq, Eq)]
struct MetadataError {
    message: Option<String>,
    code: Option<String>,
}

impl MetadataError {
    fn from(err: &dyn ProvideErrorMetadata) -> Self {
        MetadataError {
            message: err.message().map(String::from),
            code: err.code().map(String::from),
        }
    }
}

impl Display for MetadataError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        let display = match (&self.message, &self.code) {
            (None, None) => "Unknown".to_string(),
            (None, Some(code)) => format!("({code})"),
            (Some(message), None) => message.to_string(),
            (Some(message), Some(code)) => format!("{message} ({code})"),
        };
        write!(f, "{display}")
    }
}

#[derive(Debug, PartialEq, Eq)]
pub struct ScenarioError {
    message: String,
```

```
    context: Option<MetadataError>,
}

impl ScenarioError {
    pub fn with(message: impl Into<String>) -> Self {
        ScenarioError {
            message: message.into(),
            context: None,
        }
    }

    pub fn new(message: impl Into<String>, err: &dyn ProvideErrorMetadata) ->
 Self {
        ScenarioError {
            message: message.into(),
            context: Some(MetadataError::from(err)),
        }
    }
}

impl std::error::Error for ScenarioError {}
impl Display for ScenarioError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        match &self.context {
            Some(c) => write!(f, "{}: {}", self.message, c),
            None => write!(f, "{}", self.message),
        }
    }
}

// Parse the ParameterName, Description, and AllowedValues values and display
 them.
#[derive(Debug)]
pub struct AuroraScenarioParameter {
    name: String,
    allowed_values: String,
    current_value: String,
}

impl Display for AuroraScenarioParameter {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(
            f,
            "{}: {} (allowed: {})",
```

```
            self.name, self.current_value, self.allowed_values
        )
    }
}

impl From<aws_sdk_rds::types::Parameter> for AuroraScenarioParameter {
    fn from(value: aws_sdk_rds::types::Parameter) -> Self {
        AuroraScenarioParameter {
            name: value.parameter_name.unwrap_or_default(),
            allowed_values: value.allowed_values.unwrap_or_default(),
            current_value: value.parameter_value.unwrap_or_default(),
        }
    }
}

pub struct AuroraScenario {
    rds: crate::rds::Rds,
    engine_family: Option<String>,
    engine_version: Option<String>,
    instance_class: Option<String>,
    db_cluster_parameter_group: Option<DbClusterParameterGroup>,
    db_cluster_identifier: Option<String>,
    db_instance_identifier: Option<String>,
    username: Option<String>,
    password: Option<SecretString>,
}

impl AuroraScenario {
    pub fn new(client: crate::rds::Rds) -> Self {
        AuroraScenario {
            rds: client,
            engine_family: None,
            engine_version: None,
            instance_class: None,
            db_cluster_parameter_group: None,
            db_cluster_identifier: None,
            db_instance_identifier: None,
            username: None,
            password: None,
        }
    }

    // snippet-start:[rust.aurora.get_engines.usage]
```

```
   // Get available engine families for Aurora MySql.
rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
   pub async fn get_engines(&self) -> Result<HashMap<String, Vec<String>>,
ScenarioError> {
       let describe_db_engine_versions =
self.rds.describe_db_engine_versions(DB_ENGINE).await;
       trace!(versions=?describe_db_engine_versions, "full list of versions");

       if let Err(err) = describe_db_engine_versions {
           return Err(ScenarioError::new(
               "Failed to retrieve DB Engine Versions",
               &err,
           ));
       };

       let version_count = describe_db_engine_versions
           .as_ref()
           .map(|o| o.db_engine_versions().len())
           .unwrap_or_default();
       info!(version_count, "got list of versions");

       // Create a map of engine families to their available versions.
       let mut versions = HashMap::<String, Vec<String>>::new();
       describe_db_engine_versions
           .unwrap()
           .db_engine_versions()
           .iter()
           .filter_map(
               |v| match (&v.db_parameter_group_family, &v.engine_version) {
                   (Some(family), Some(version)) => Some((family.clone(),
version.clone())),
                   _ => None,
               },
           )
           .for_each(|(family, version)|
versions.entry(family).or_default().push(version));

       Ok(versions)
   }
   // snippet-end:[rust.aurora.get_engines.usage]

   // snippet-start:[rust.aurora.get_instance_classes.usage]
```

```
   pub async fn get_instance_classes(&self) -> Result<Vec<String>,
ScenarioError> {
       let describe_orderable_db_instance_options_items = self
           .rds
           .describe_orderable_db_instance_options(
               DB_ENGINE,
               self.engine_version
                   .as_ref()
                   .expect("engine version for db instance options")
                   .as_str(),
           )
           .await;

       describe_orderable_db_instance_options_items
           .map(|options| {
               options
                   .iter()
                   .map(|o|
o.db_instance_class().unwrap_or_default().to_string())
                   .collect::<Vec<String>>()
           })
           .map_err(|err| ScenarioError::new("Could not get available instance
classes", &err))
   }
   // snippet-end:[rust.aurora.get_instance_classes.usage]

   // snippet-start:[rust.aurora.set_engine.usage]
   // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
   pub async fn set_engine(&mut self, engine: &str, version: &str) -> Result<(),
ScenarioError> {
       self.engine_family = Some(engine.to_string());
       self.engine_version = Some(version.to_string());
       let create_db_cluster_parameter_group = self
           .rds
           .create_db_cluster_parameter_group(
               DB_CLUSTER_PARAMETER_GROUP_NAME,
               DB_CLUSTER_PARAMETER_GROUP_DESCRIPTION,
               engine,
           )
           .await;

       match create_db_cluster_parameter_group {
           Ok(CreateDbClusterParameterGroupOutput {
```

```
                db_cluster_parameter_group: None,
                ..
            }) => {
                return Err(ScenarioError::with(
                    "CreateDBClusterParameterGroup had empty response",
                ));
            }
            Err(error) => {
                if error.code() == Some("DBParameterGroupAlreadyExists") {
                    info!("Cluster Parameter Group already exists, nothing to
do");
                } else {
                    return Err(ScenarioError::new(
                        "Could not create Cluster Parameter Group",
                        &error,
                    ));
                }
            }
            _ => {
                info!("Created Cluster Parameter Group");
            }
        }

        Ok(())
    }
    // snippet-end:[rust.aurora.set_engine.usage]

    pub fn set_instance_class(&mut self, instance_class: Option<String>) {
        self.instance_class = instance_class;
    }

    pub fn set_login(&mut self, username: Option<String>, password:
Option<SecretString>) {
        self.username = username;
        self.password = password;
    }

    pub async fn connection_string(&self) -> Result<String, ScenarioError> {
        let cluster = self.get_cluster().await?;
        let endpoint = cluster.endpoint().unwrap_or_default();
        let port = cluster.port().unwrap_or_default();
        let username = cluster.master_username().unwrap_or_default();
        Ok(format!("mysql -h {endpoint} -P {port} -u {username} -p"))
    }
```

```
   // snippet-start:[rust.aurora.get_cluster.usage]
   pub async fn get_cluster(&self) -> Result<DbCluster, ScenarioError> {
       let describe_db_clusters_output = self
           .rds
           .describe_db_clusters(
               self.db_cluster_identifier
                   .as_ref()
                   .expect("cluster identifier")
                   .as_str(),
           )
           .await;
       if let Err(err) = describe_db_clusters_output {
           return Err(ScenarioError::new("Failed to get cluster", &err));
       }

       let db_cluster = describe_db_clusters_output
           .unwrap()
           .db_clusters
           .and_then(|output| output.first().cloned());

       db_cluster.ok_or_else(|| ScenarioError::with("Did not find the cluster"))
   }
   // snippet-end:[rust.aurora.get_cluster.usage]

   // snippet-start:[rust.aurora.cluster_parameters.usage]
   // Get the parameter group. rds.DescribeDbClusterParameterGroups
   // Get parameters in the group. This is a long list so you will have to
paginate. Find the auto_increment_offset and auto_increment_increment parameters
(by ParameterName). rds.DescribeDbClusterParameters
   // Parse the ParameterName, Description, and AllowedValues values and display
them.
   pub async fn cluster_parameters(&self) ->
Result<Vec<AuroraScenarioParameter>, ScenarioError> {
       let parameters_output = self
           .rds
           .describe_db_cluster_parameters(DB_CLUSTER_PARAMETER_GROUP_NAME)
           .await;

       if let Err(err) = parameters_output {
           return Err(ScenarioError::new(
               format!("Failed to retrieve parameters for
{DB_CLUSTER_PARAMETER_GROUP_NAME}"),
               &err,
```

```
            ));
        }

        let parameters = parameters_output
            .unwrap()
            .into_iter()
            .flat_map(|p| p.parameters.unwrap_or_default().into_iter())
            .filter(|p|
FILTER_PARAMETER_NAMES.contains(p.parameter_name().unwrap_or_default()))
            .map(AuroraScenarioParameter::from)
            .collect::<Vec<_>>();

        Ok(parameters)
    }
    // snippet-end:[rust.aurora.cluster_parameters.usage]

    // snippet-start:[rust.aurora.update_auto_increment.usage]
    // Modify both the auto_increment_offset and auto_increment_increment
parameters in one call in the custom parameter group. Set their ParameterValue
fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
    pub async fn update_auto_increment(
        &self,
        offset: u8,
        increment: u8,
    ) -> Result<(), ScenarioError> {
        let modify_db_cluster_parameter_group = self
            .rds
            .modify_db_cluster_parameter_group(
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value(format!("{offset}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value(format!("{increment}"))
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ],
            )
            .await;
```

```
        if let Err(error) = modify_db_cluster_parameter_group {
            return Err(ScenarioError::new(
                "Failed to modify cluster parameter group",
                &error,
            ));
        }

        Ok(())
    }
    // snippet-end:[rust.aurora.update_auto_increment.usage]

    // snippet-start:[rust.aurora.start_cluster_and_instance.usage]
    // Get a list of allowed engine versions.
rds.DescribeDbEngineVersions(Engine='aurora-mysql', DBParameterGroupFamily=<the
family used to create your parameter group in step 2>)
    // Create an Aurora DB cluster database cluster that contains a MySql
database and uses the parameter group you created.
    // Wait for DB cluster to be ready. Call rds.DescribeDBClusters and check for
Status == 'available'.
    // Get a list of instance classes available for the selected engine
and engine version. rds.DescribeOrderableDbInstanceOptions(Engine='mysql',
EngineVersion=).

    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
for DBInstanceStatus == 'available'.
    pub async fn start_cluster_and_instance(&mut self) -> Result<(),
ScenarioError> {
        if self.password.is_none() {
            return Err(ScenarioError::with(
                "Must set Secret Password before starting a cluster",
            ));
        }
        let create_db_cluster = self
            .rds
            .create_db_cluster(
                DB_CLUSTER_IDENTIFIER,
                DB_CLUSTER_PARAMETER_GROUP_NAME,
                DB_ENGINE,
                self.engine_version.as_deref().expect("engine version"),
                self.username.as_deref().expect("username"),
                self.password
                    .replace(SecretString::new("".to_string()))
                    .expect("password"),
```

```
            )
            .await;
        if let Err(err) = create_db_cluster {
            return Err(ScenarioError::new(
                "Failed to create DB Cluster with cluster group",
                &err,
            ));
        }

        self.db_cluster_identifier = create_db_cluster
            .unwrap()
            .db_cluster
            .and_then(|c| c.db_cluster_identifier);

        if self.db_cluster_identifier.is_none() {
            return Err(ScenarioError::with("Created DB Cluster missing
Identifier"));
        }

        info!(
            "Started a db cluster: {}",
            self.db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing ARN")
        );

        let create_db_instance = self
            .rds
            .create_db_instance(
                self.db_cluster_identifier.as_deref().expect("cluster name"),
                DB_INSTANCE_IDENTIFIER,
                self.instance_class.as_deref().expect("instance class"),
                DB_ENGINE,
            )
            .await;
        if let Err(err) = create_db_instance {
            return Err(ScenarioError::new(
                "Failed to create Instance in DB Cluster",
                &err,
            ));
        }

        self.db_instance_identifier = create_db_instance
            .unwrap()
```

```
            .db_instance
            .and_then(|i| i.db_instance_identifier);

        // Cluster creation can take up to 20 minutes to become available
        let cluster_max_wait = Duration::from_secs(20 * 60);
        let waiter = Waiter::builder().max(cluster_max_wait).build();
        while waiter.sleep().await.is_ok() {
            let cluster = self
                .rds
                .describe_db_clusters(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = cluster {
                warn!(?err, "Failed to describe cluster while waiting for
ready");
                continue;
            }

            let instance = self
                .rds
                .describe_db_instance(
                    self.db_instance_identifier
                        .as_deref()
                        .expect("instance identifier"),
                )
                .await;
            if let Err(err) = instance {
                return Err(ScenarioError::new(
                    "Failed to find instance for cluster",
                    &err,
                ));
            }

            let instances_available = instance
                .unwrap()
                .db_instances()
                .iter()
                .all(|instance| instance.db_instance_status() ==
Some("Available"));
```

```
            let endpoints = self
                .rds
                .describe_db_cluster_endpoints(
                    self.db_cluster_identifier
                        .as_deref()
                        .expect("cluster identifier"),
                )
                .await;

            if let Err(err) = endpoints {
                return Err(ScenarioError::new(
                    "Failed to find endpoint for cluster",
                    &err,
                ));
            }

            let endpoints_available = endpoints
                .unwrap()
                .db_cluster_endpoints()
                .iter()
                .all(|endpoint| endpoint.status() == Some("available"));

            if instances_available && endpoints_available {
                return Ok(());
            }
        }

        Err(ScenarioError::with("timed out waiting for cluster"))
    }
    // snippet-end:[rust.aurora.start_cluster_and_instance.usage]

    // snippet-start:[rust.aurora.snapshot.usage]
    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
    // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
Status == 'available'.
    pub async fn snapshot(&self, name: &str) -> Result<DbClusterSnapshot,
ScenarioError> {
        let id = self.db_cluster_identifier.as_deref().unwrap_or_default();
        let snapshot = self
            .rds
            .snapshot_cluster(id, format!("{id}_{name}").as_str())
            .await;
        match snapshot {
            Ok(output) => match output.db_cluster_snapshot {
```

```
                Some(snapshot) => Ok(snapshot),
                None => Err(ScenarioError::with("Missing Snapshot")),
            },
            Err(err) => Err(ScenarioError::new("Failed to create snapshot",
&err)),
        }
    }
    // snippet-end:[rust.aurora.snapshot.usage]

    // snippet-start:[rust.aurora.clean_up.usage]
    pub async fn clean_up(self) -> Result<(), Vec<ScenarioError>> {
        let mut clean_up_errors: Vec<ScenarioError> = vec![];

        // Delete the instance. rds.DeleteDbInstance.
        let delete_db_instance = self
            .rds
            .delete_db_instance(
                self.db_instance_identifier
                    .as_deref()
                    .expect("instance identifier"),
            )
            .await;
        if let Err(err) = delete_db_instance {
            let identifier = self
                .db_instance_identifier
                .as_deref()
                .unwrap_or("Missing Instance Identifier");
            let message = format!("failed to delete db instance {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
            // Wait for the instance to delete
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_instances =
self.rds.describe_db_instances().await;
                if let Err(err) = describe_db_instances {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check instance state during deletion",
                        &err,
                    ));
                    break;
                }
                let db_instances = describe_db_instances
                    .unwrap()
```

```
                    .db_instances()
                    .iter()
                    .filter(|instance| instance.db_cluster_identifier ==
self.db_cluster_identifier)
                    .cloned()
                    .collect::<Vec<DbInstance>>();

                if db_instances.is_empty() {
                    trace!("Delete Instance waited and no instances were found");
                    break;
                }
                match db_instances.first().unwrap().db_instance_status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but instances is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB instance");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster. rds.DeleteDbCluster.
        let delete_db_cluster = self
            .rds
            .delete_db_cluster(
                self.db_cluster_identifier
                    .as_deref()
                    .expect("cluster identifier"),
            )
            .await;

        if let Err(err) = delete_db_cluster {
            let identifier = self
                .db_cluster_identifier
                .as_deref()
                .unwrap_or("Missing DB Cluster Identifier");
            let message = format!("failed to delete db cluster {identifier}");
            clean_up_errors.push(ScenarioError::new(message, &err));
        } else {
```

```
            // Wait for the instance and cluster to fully delete.
rds.DescribeDbInstances and rds.DescribeDbClusters until both are not found.
            let waiter = Waiter::default();
            while waiter.sleep().await.is_ok() {
                let describe_db_clusters = self
                    .rds
                    .describe_db_clusters(
                        self.db_cluster_identifier
                            .as_deref()
                            .expect("cluster identifier"),
                    )
                    .await;
                if let Err(err) = describe_db_clusters {
                    clean_up_errors.push(ScenarioError::new(
                        "Failed to check cluster state during deletion",
                        &err,
                    ));
                    break;
                }
                let describe_db_clusters = describe_db_clusters.unwrap();
                let db_clusters = describe_db_clusters.db_clusters();
                if db_clusters.is_empty() {
                    trace!("Delete cluster waited and no clusters were found");
                    break;
                }
                match db_clusters.first().unwrap().status() {
                    Some("Deleting") => continue,
                    Some(status) => {
                        info!("Attempting to delete but clusters is in
{status}");
                        continue;
                    }
                    None => {
                        warn!("No status for DB cluster");
                        break;
                    }
                }
            }
        }

        // Delete the DB cluster parameter group.
rds.DeleteDbClusterParameterGroup.
        let delete_db_cluster_parameter_group = self
            .rds
```

```
            .delete_db_cluster_parameter_group(
                self.db_cluster_parameter_group
                    .map(|g| {
                        g.db_cluster_parameter_group_name
                            .unwrap_or_else(||
 DB_CLUSTER_PARAMETER_GROUP_NAME.to_string())
                    })
                    .as_deref()
                    .expect("cluster parameter group name"),
            )
            .await;
        if let Err(error) = delete_db_cluster_parameter_group {
            clean_up_errors.push(ScenarioError::new(
                "Failed to delete the db cluster parameter group",
                &error,
            ))
        }

        if clean_up_errors.is_empty() {
            Ok(())
        } else {
            Err(clean_up_errors)
        }
    }
    // snippet-end:[rust.aurora.clean_up.usage]
}

#[cfg(test)]
pub mod tests;
```

Pengujian untuk pustaka menggunakan automock di sekitar pembungkus Klien RDS.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use crate::rds::MockRdsImpl;

use super::*;

use std::io::{Error, ErrorKind};

use assert_matches::assert_matches;
```

```
use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::DeleteDbClusterOutput,
        delete_db_cluster_parameter_group::DeleteDbClusterParameterGroupOutput,
        delete_db_instance::DeleteDbInstanceOutput,
        describe_db_cluster_endpoints::DescribeDbClusterEndpointsOutput,
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
        },
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{
        error::DbParameterGroupAlreadyExistsFault, DbClusterEndpoint,
 DbEngineVersion,
        OrderableDbInstanceOption,
    },
};
use aws_smithy_runtime_api::http::{Response, StatusCode};
use aws_smithy_types::body::SdkBody;
use mockall::predicate::eq;
use secrecy::ExposeSecret;

// snippet-start:[rust.aurora.set_engine.test]
#[tokio::test]
async fn test_scenario_set_engine() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert_eq!(set_engine, Ok(()));
    assert_eq!(Some("aurora-mysql"), scenario.engine_family.as_deref());
    assert_eq!(Some("aurora-mysql8.0"), scenario.engine_version.as_deref());
}

#[tokio::test]
async fn test_scenario_set_engine_not_create() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .with(
            eq("RustSDKCodeExamplesDBParameterGroup"),
            eq("Parameter Group created by Rust SDK Code Example"),
            eq("aurora-mysql"),
        )
        .return_once(|_, _, _|
 Ok(CreateDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;
```

```
    assert!(set_engine.is_err());
}

#[tokio::test]
async fn test_scenario_set_engine_param_group_exists() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .withf(|_, _, _| true)
        .return_once(|_, _, _| {
            Err(SdkError::service_error(

 CreateDBClusterParameterGroupError::DbParameterGroupAlreadyExistsFault(
                    DbParameterGroupAlreadyExistsFault::builder().build(),
                ),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);

    let set_engine = scenario.set_engine("aurora-mysql", "aurora-
mysql8.0").await;

    assert!(set_engine.is_err());
}
// snippet-end:[rust.aurora.set_engine.test]

// snippet-start:[rust.aurora.get_engines.test]
#[tokio::test]
async fn test_scenario_get_engines() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Ok(DescribeDbEngineVersionsOutput::builder()
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1a")
```

```
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f1")
                        .engine_version("f1b")
                        .build(),
                )
                .db_engine_versions(
                    DbEngineVersion::builder()
                        .db_parameter_group_family("f2")
                        .engine_version("f2a")
                        .build(),
                )
                .db_engine_versions(DbEngineVersion::builder().build())
                .build())
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;

    assert_eq!(
        versions_map,
        Ok(HashMap::from([
            ("f1".into(), vec!["f1a".into(), "f1b".into()]),
            ("f2".into(), vec!["f2a".into()])
        ]))
    );
}

#[tokio::test]
async fn test_scenario_get_engines_failed() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_engine_versions()
        .with(eq("aurora-mysql"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBEngineVersionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_engine_versions error",
                ))),
```

```
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let versions_map = scenario.get_engines().await;
    assert_matches!(
        versions_map,
        Err(ScenarioError { message, context: _ }) if message == "Failed to
 retrieve DB Engine Versions"
    );
}
// snippet-end:[rust.aurora.get_engines.test]

// snippet-start:[rust.aurora.get_instance_classes.test]
#[tokio::test]
async fn test_scenario_get_instance_classes() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Ok(vec![
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t1")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t2")
                    .build(),
                OrderableDbInstanceOption::builder()
                    .db_instance_class("t3")
                    .build(),
```

```
            ])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario
        .set_engine("aurora-mysql", "aurora-mysql8.0")
        .await
        .expect("set engine");

    let instance_classes = scenario.get_instance_classes().await;

    assert_eq!(
        instance_classes,
        Ok(vec!["t1".into(), "t2".into(), "t3".into()])
    );
}

#[tokio::test]
async fn test_scenario_get_instance_classes_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_orderable_db_instance_options()
        .with(eq("aurora-mysql"), eq("aurora-mysql8.0"))
        .return_once(|_, _| {
            Err(SdkError::service_error(

 DescribeOrderableDBInstanceOptionsError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_orderable_db_instance_options_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_family = Some("aurora-mysql".into());
    scenario.engine_version = Some("aurora-mysql8.0".into());

    let instance_classes = scenario.get_instance_classes().await;

    assert_matches!(
        instance_classes,
```

```
        Err(ScenarioError {message, context: _}) if message == "Could not get
 available instance classes"
    );
}
// snippet-end:[rust.aurora.get_instance_classes.test]

// snippet-start:[rust.aurora.get_cluster.test]
#[tokio::test]
async fn test_scenario_get_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert!(cluster.is_ok());
}

#[tokio::test]
async fn test_scenario_get_cluster_missing_cluster() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| Ok(DescribeDbClustersOutput::builder().build()));
```

```
    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Did not find the cluster");
}

#[tokio::test]
async fn test_scenario_get_cluster_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster_parameter_group()
        .return_once(|_, _, _| {
            Ok(CreateDbClusterParameterGroupOutput::builder()

 .db_cluster_parameter_group(DbClusterParameterGroup::builder().build())
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_clusters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let cluster = scenario.get_cluster().await;

    assert_matches!(cluster, Err(ScenarioError { message, context: _ }) if
 message == "Failed to get cluster");
}
// snippet-end:[rust.aurora.get_cluster.test]
```

```
#[tokio::test]
async fn test_scenario_connection_string() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .endpoint("test_endpoint")
                        .port(3306)
                        .master_username("test_username")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
    let connection_string = scenario.connection_string().await;

    assert_eq!(
        connection_string,
        Ok("mysql -h test_endpoint -P 3306 -u test_username -p".into())
    );
}

// snippet-start:[rust.aurora.cluster_parameters.test]
#[tokio::test]
async fn test_scenario_cluster_parameters() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Ok(vec![DescribeDbClusterParametersOutput::builder()
                .parameters(Parameter::builder().parameter_name("a").build())
                .parameters(Parameter::builder().parameter_name("b").build())
                .parameters(
                    Parameter::builder()
```

```
                        .parameter_name("auto_increment_offset")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("c").build())
                .parameters(
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .build(),
                )
                .parameters(Parameter::builder().parameter_name("d").build())
                .build()])
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());

    let params = scenario.cluster_parameters().await.expect("cluster params");
    let names: Vec<String> = params.into_iter().map(|p| p.name).collect();
    assert_eq!(
        names,
        vec!["auto_increment_offset", "auto_increment_increment"]
    );
}

#[tokio::test]
async fn test_scenario_cluster_parameters_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_describe_db_cluster_parameters()
        .with(eq("RustSDKCodeExamplesDBParameterGroup"))
        .return_once(|_| {
            Err(SdkError::service_error(
                DescribeDBClusterParametersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe_db_cluster_parameters_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("RustSDKCodeExamplesDBCluster".into());
```

```
    let params = scenario.cluster_parameters().await;
    assert_matches!(params, Err(ScenarioError { message, context: _ }) if message
 == "Failed to retrieve parameters for RustSDKCodeExamplesDBParameterGroup");
}
// snippet-end:[rust.aurora.cluster_parameters.test]

// snippet-start:[rust.aurora.update_auto_increment.test]
#[tokio::test]
async fn test_scenario_update_auto_increment() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .withf(|name, params| {
            assert_eq!(name, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(
                params,
                &vec![
                    Parameter::builder()
                        .parameter_name("auto_increment_offset")
                        .parameter_value("10")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                    Parameter::builder()
                        .parameter_name("auto_increment_increment")
                        .parameter_value("20")
                        .apply_method(aws_sdk_rds::types::ApplyMethod::Immediate)
                        .build(),
                ]
            );
            true
        })
        .return_once(|_, _|
 Ok(ModifyDbClusterParameterGroupOutput::builder().build()));

    let scenario = AuroraScenario::new(mock_rds);

    scenario
        .update_auto_increment(10, 20)
        .await
        .expect("update auto increment");
}

#[tokio::test]
```

```
async fn test_scenario_update_auto_increment_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_modify_db_cluster_parameter_group()
        .return_once(|_, _| {
            Err(SdkError::service_error(

 ModifyDBClusterParameterGroupError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "modify_db_cluster_parameter_group_error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let scenario = AuroraScenario::new(mock_rds);

    let update = scenario.update_auto_increment(10, 20).await;
    assert_matches!(update, Err(ScenarioError { message, context: _}) if message
 == "Failed to modify cluster parameter group");
}
// snippet-end:[rust.aurora.update_auto_increment.test]

// snippet-start:[rust.aurora.start_cluster_and_instance.test]
#[tokio::test]
async fn test_start_cluster_and_instance() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
```

```
.db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_describe_db_instance()
        .with(eq("RustSDKCodeExamplesDBInstance"))
        .return_once(|name| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_instance_identifier(name)
```

```
                        .db_instance_status("Available")
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()

 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
        assert!(scenario
            .password
            .replace(SecretString::new("BAD SECRET".into()))
            .unwrap()
            .expose_secret()
            .is_empty());
        assert_eq!(
            scenario.db_cluster_identifier,
            Some("RustSDKCodeExamplesDBCluster".into())
        );
    });
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_error() {
```

```
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Err(SdkError::service_error(
                CreateDBClusterError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _}) if message
 == "Failed to create DB Cluster with cluster group")
}

#[tokio::test]
async fn test_start_cluster_and_instance_cluster_create_missing_id() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .return_once(|_, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()
                .db_cluster(DbCluster::builder().build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));
```

```
    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context:_ }) if message
 == "Created DB Cluster missing Identifier");
}

#[tokio::test]
async fn test_start_cluster_and_instance_instance_create_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .return_once(|_, _, _, _| {
            Err(SdkError::service_error(
                CreateDBInstanceError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create db instance error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
```

```
    scenario.password = Some(SecretString::new("test password".into()));

    let create = scenario.start_cluster_and_instance().await;
    assert_matches!(create, Err(ScenarioError { message, context: _ }) if message
 == "Failed to create Instance in DB Cluster")
}

#[tokio::test]
async fn test_start_cluster_and_instance_wait_hiccup() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_create_db_cluster()
        .withf(|id, params, engine, version, username, password| {
            assert_eq!(id, "RustSDKCodeExamplesDBCluster");
            assert_eq!(params, "RustSDKCodeExamplesDBParameterGroup");
            assert_eq!(engine, "aurora-mysql");
            assert_eq!(version, "aurora-mysql8.0");
            assert_eq!(username, "test username");
            assert_eq!(password.expose_secret(), "test password");
            true
        })
        .return_once(|id, _, _, _, _, _| {
            Ok(CreateDbClusterOutput::builder()

 .db_cluster(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds
        .expect_create_db_instance()
        .withf(|cluster, name, class, engine| {
            assert_eq!(cluster, "RustSDKCodeExamplesDBCluster");
            assert_eq!(name, "RustSDKCodeExamplesDBInstance");
            assert_eq!(class, "m5.large");
            assert_eq!(engine, "aurora-mysql");
            true
        })
        .return_once(|cluster, name, class, _| {
            Ok(CreateDbInstanceOutput::builder()
                .db_instance(
                    DbInstance::builder()
                        .db_cluster_identifier(cluster)
                        .db_instance_identifier(name)
```

```
                        .db_instance_class(class)
                        .build(),
                )
                .build())
        });

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe cluster error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        })
        .with(eq("RustSDKCodeExamplesDBCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()

.db_clusters(DbCluster::builder().db_cluster_identifier(id).build())
                .build())
        });

    mock_rds.expect_describe_db_instance().return_once(|name| {
        Ok(DescribeDbInstancesOutput::builder()
            .db_instances(
                DbInstance::builder()
                    .db_instance_identifier(name)
                    .db_instance_status("Available")
                    .build(),
            )
            .build())
    });

    mock_rds
        .expect_describe_db_cluster_endpoints()
        .return_once(|_| {
            Ok(DescribeDbClusterEndpointsOutput::builder()
```

```
 .db_cluster_endpoints(DbClusterEndpoint::builder().status("available").build())
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.engine_version = Some("aurora-mysql8.0".into());
    scenario.instance_class = Some("m5.large".into());
    scenario.username = Some("test username".into());
    scenario.password = Some(SecretString::new("test password".into()));

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let create = scenario.start_cluster_and_instance().await;
        assert!(create.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::advance(Duration::from_secs(1)).await;
    tokio::time::resume();
    let _ = assertions.await;
}
// snippet-end:[rust.aurora.start_cluster_and_instance.test]

// snippet-start:[rust.aurora.clean_up.test]
#[tokio::test]
async fn test_scenario_clean_up() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
```

```
                        .build(),
                )
                .build())
        })
        .with()
        .times(1)
        .returning(|| Ok(DescribeDbInstancesOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| Ok(DescribeDbClustersOutput::builder().build()));

    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
```

```
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_ok());
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}

#[tokio::test]
async fn test_scenario_clean_up_errors() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_delete_db_instance()
        .with(eq("MockInstance"))
        .return_once(|_| Ok(DeleteDbInstanceOutput::builder().build()));

    mock_rds
        .expect_describe_db_instances()
        .with()
        .times(1)
        .returning(|| {
            Ok(DescribeDbInstancesOutput::builder()
                .db_instances(
                    DbInstance::builder()
                        .db_cluster_identifier("MockCluster")
                        .db_instance_status("Deleting")
                        .build(),
                )
                .build())
        })
        .with()
```

```
        .times(1)
        .returning(|| {
            Err(SdkError::service_error(
                DescribeDBInstancesError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db instances error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });

    mock_rds
        .expect_delete_db_cluster()
        .with(eq("MockCluster"))
        .return_once(|_| Ok(DeleteDbClusterOutput::builder().build()));

    mock_rds
        .expect_describe_db_clusters()
        .with(eq("MockCluster"))
        .times(1)
        .returning(|id| {
            Ok(DescribeDbClustersOutput::builder()
                .db_clusters(
                    DbCluster::builder()
                        .db_cluster_identifier(id)
                        .status("Deleting")
                        .build(),
                )
                .build())
        })
        .with(eq("MockCluster"))
        .times(1)
        .returning(|_| {
            Err(SdkError::service_error(
                DescribeDBClustersError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "describe db clusters error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
SdkBody::empty()),
            ))
        });
```

```
    mock_rds
        .expect_delete_db_cluster_parameter_group()
        .with(eq("MockParamGroup"))
        .return_once(|_|
 Ok(DeleteDbClusterParameterGroupOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some(String::from("MockCluster"));
    scenario.db_instance_identifier = Some(String::from("MockInstance"));
    scenario.db_cluster_parameter_group = Some(
        DbClusterParameterGroup::builder()
            .db_cluster_parameter_group_name("MockParamGroup")
            .build(),
    );

    tokio::time::pause();
    let assertions = tokio::spawn(async move {
        let clean_up = scenario.clean_up().await;
        assert!(clean_up.is_err());
        let errs = clean_up.unwrap_err();
        assert_eq!(errs.len(), 2);
        assert_matches!(errs.get(0), Some(ScenarioError {message, context: _}) if
 message == "Failed to check instance state during deletion");
        assert_matches!(errs.get(1), Some(ScenarioError {message, context: _}) if
 message == "Failed to check cluster state during deletion");
    });

    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Instances
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for first
 Describe Cluster
    tokio::time::advance(Duration::from_secs(1)).await; // Wait for second
 Describe Cluster
    tokio::time::resume();
    let _ = assertions.await;
}
// snippet-end:[rust.aurora.clean_up.test]

// snippet-start:[rust.aurora.snapshot.test]
#[tokio::test]
async fn test_scenario_snapshot() {
    let mut mock_rds = MockRdsImpl::default();
```

```
    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| {
            Ok(CreateDbClusterSnapshotOutput::builder()
                .db_cluster_snapshot(
                    DbClusterSnapshot::builder()
                        .db_cluster_identifier("MockCluster")

 .db_cluster_snapshot_identifier("MockCluster_MockSnapshot")
                        .build(),
                )
                .build())
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert!(create_snapshot.is_ok());
}

#[tokio::test]
async fn test_scenario_snapshot_error() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _| {
            Err(SdkError::service_error(
                CreateDBClusterSnapshotError::unhandled(Box::new(Error::new(
                    ErrorKind::Other,
                    "create snapshot error",
                ))),
                Response::new(StatusCode::try_from(400).unwrap(),
 SdkBody::empty()),
            ))
        });

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
```

```
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Failed to create snapshot");
}

#[tokio::test]
async fn test_scenario_snapshot_invalid() {
    let mut mock_rds = MockRdsImpl::default();

    mock_rds
        .expect_snapshot_cluster()
        .with(eq("MockCluster"), eq("MockCluster_MockSnapshot"))
        .times(1)
        .return_once(|_, _|
 Ok(CreateDbClusterSnapshotOutput::builder().build()));

    let mut scenario = AuroraScenario::new(mock_rds);
    scenario.db_cluster_identifier = Some("MockCluster".into());
    let create_snapshot = scenario.snapshot("MockSnapshot").await;
    assert_matches!(create_snapshot, Err(ScenarioError { message, context: _}) if
 message == "Missing Snapshot");
}
// snippet-end:[rust.aurora.snapshot.test]
```

Biner untuk menjalankan skenario dari depan ke ujung, menggunakan inquirer sehingga pengguna dapat membuat beberapa keputusan.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use std::fmt::Display;

use anyhow::anyhow;
use aurora_code_examples::{
    aurora_scenario::{AuroraScenario, ScenarioError},
    rds::Rds as RdsClient,
};
use aws_sdk_rds::Client;
use inquire::{validator::StringValidator, CustomUserError};
use secrecy::SecretString;
use tracing::warn;
```

```
#[derive(Default, Debug)]
struct Warnings(Vec<String>);

impl Warnings {
    fn new() -> Self {
        Warnings(Vec::with_capacity(5))
    }

    fn push(&mut self, warning: &str, error: ScenarioError) {
        let formatted = format!("{warning}: {error}");
        warn!("{formatted}");
        self.0.push(formatted);
    }

    fn is_empty(&self) -> bool {
        self.0.is_empty()
    }
}

impl Display for Warnings {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        writeln!(f, "Warnings:")?;
        for warning in &self.0 {
            writeln!(f, "{: >4}- {warning}", "")?;
        }
        Ok(())
    }
}

fn select(
    prompt: &str,
    choices: Vec<String>,
    error_message: &str,
) -> Result<String, anyhow::Error> {
    inquire::Select::new(prompt, choices)
        .prompt()
        .map_err(|error| anyhow!("{error_message}: {error}"))
}

// Prepare the Aurora Scenario. Prompt for several settings that are optional to
 the Scenario, but that the user should choose for the demo.
// This includes the engine, engine version, and instance class.
async fn prepare_scenario(rds: RdsClient) -> Result<AuroraScenario,
 anyhow::Error> {
```

```
    let mut scenario = AuroraScenario::new(rds);

    // Get available engine families for Aurora MySql.
rds.DescribeDbEngineVersions(Engine='aurora-mysql') and build a set of the
'DBParameterGroupFamily' field values. I get {aurora-mysql8.0, aurora-mysql5.7}.
    let available_engines = scenario.get_engines().await;
    if let Err(error) = available_engines {
        return Err(anyhow!("Failed to get available engines: {}", error));
    }
    let available_engines = available_engines.unwrap();

    // Select an engine family and create a custom DB cluster parameter group.
rds.CreateDbClusterParameterGroup(DBParameterGroupFamily='aurora-mysql8.0')
    let engine = select(
        "Select an Aurora engine family",
        available_engines.keys().cloned().collect::<Vec<String>>(),
        "Invalid engine selection",
    )?;

    let version = select(
        format!("Select an Aurora engine version for {engine}").as_str(),
        available_engines.get(&engine).cloned().unwrap_or_default(),
        "Invalid engine version selection",
    )?;

    let set_engine = scenario.set_engine(engine.as_str(),
version.as_str()).await;
    if let Err(error) = set_engine {
        return Err(anyhow!("Could not set engine: {}", error));
    }

    let instance_classes = scenario.get_instance_classes().await;
    match instance_classes {
        Ok(classes) => {
            let instance_class = select(
                format!("Select an Aurora instance class for {engine}").as_str(),
                classes,
                "Invalid instance class selection",
            )?;
            scenario.set_instance_class(Some(instance_class))
        }
        Err(err) => return Err(anyhow!("Failed to get instance classes for
engine: {err}")),
    }
```

```
    Ok(scenario)
}

// Prepare the cluster, creating a custom parameter group overriding some group
 parameters based on user input.
async fn prepare_cluster(scenario: &mut AuroraScenario, warnings: &mut Warnings)
 -> Result<(), ()> {
    show_parameters(scenario, warnings).await;

    let offset = prompt_number_or_default(warnings, "auto_increment_offset", 5);
    let increment = prompt_number_or_default(warnings,
 "auto_increment_increment", 3);

    // Modify both the auto_increment_offset and auto_increment_increment
 parameters in one call in the custom parameter group. Set their ParameterValue
 fields to a new allowable value. rds.ModifyDbClusterParameterGroup.
    let update_auto_increment = scenario.update_auto_increment(offset,
 increment).await;

    if let Err(error) = update_auto_increment {
        warnings.push("Failed to update auto increment", error);
        return Err(());
    }

    // Get and display the updated parameters. Specify Source of 'user' to get
 just the modified parameters. rds.DescribeDbClusterParameters(Source='user')
    show_parameters(scenario, warnings).await;

    let username = inquire::Text::new("Username for the database (default
 'testuser')")
        .with_default("testuser")
        .with_initial_value("testuser")
        .prompt();

    if let Err(error) = username {
        warnings.push(
            "Failed to get username, using default",
            ScenarioError::with(format!("Error from inquirer: {error}")),
        );
        return Err(());
    }
    let username = username.unwrap();
```

```
    let password = inquire::Text::new("Password for the database (minimum 8
 characters)")
        .with_validator(|i: &str| {
            if i.len() >= 8 {
                Ok(inquire::validator::Validation::Valid)
            } else {
                Ok(inquire::validator::Validation::Invalid(
                    "Password must be at least 8 characters".into(),
                ))
            }
        })
        .prompt();

    let password: Option<SecretString> = match password {
        Ok(password) => Some(SecretString::from(password)),
        Err(error) => {
            warnings.push(
                "Failed to get password, using none (and not starting a DB)",
                ScenarioError::with(format!("Error from inquirer: {error}")),
            );
            return Err(());
        }
    };

    scenario.set_login(Some(username), password);

    Ok(())
}

// Start a single instance in the cluster,
async fn run_instance(scenario: &mut AuroraScenario) -> Result<(), ScenarioError>
 {
    // Create an Aurora DB cluster database cluster that contains a MySql
 database and uses the parameter group you created.
    // Create a database instance in the cluster.
    // Wait for DB instance to be ready. Call rds.DescribeDbInstances and check
 for DBInstanceStatus == 'available'.
    scenario.start_cluster_and_instance().await?;

    let connection_string = scenario.connection_string().await?;

    println!("Database ready: {connection_string}",);
```

```
    let _ = inquire::Text::new("Use the database with the connection string. When
 you're finished, press enter key to continue.").prompt();

    // Create a snapshot of the DB cluster. rds.CreateDbClusterSnapshot.
    // Wait for the snapshot to create. rds.DescribeDbClusterSnapshots until
 Status == 'available'.
    let snapshot_name = inquire::Text::new("Provide a name for the snapshot")
        .prompt()
        .unwrap_or(String::from("ScenarioRun"));
    let snapshot = scenario.snapshot(snapshot_name.as_str()).await?;
    println!(
        "Snapshot is available: {}",
        snapshot.db_cluster_snapshot_arn().unwrap_or("Missing ARN")
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), anyhow::Error> {
    tracing_subscriber::fmt::init();
    let sdk_config = aws_config::from_env().load().await;
    let client = Client::new(&sdk_config);
    let rds = RdsClient::new(client);
    let mut scenario = prepare_scenario(rds).await?;

    // At this point, the scenario has things in AWS and needs to get cleaned up.
    let mut warnings = Warnings::new();

    if prepare_cluster(&mut scenario, &mut warnings).await.is_ok() {
        println!("Configured database cluster, starting an instance.");
        if let Err(err) = run_instance(&mut scenario).await {
            warnings.push("Problem running instance", err);
        }
    }

    // Clean up the instance, cluster, and parameter group, waiting for the
 instance and cluster to delete before moving on.
    let clean_up = scenario.clean_up().await;
    if let Err(errors) = clean_up {
        for error in errors {
            warnings.push("Problem cleaning up scenario", error);
        }
    }
```

```
    if warnings.is_empty() {
        Ok(())
    } else {
        println!("There were problems running the scenario:");
        println!("{warnings}");
        Err(anyhow!("There were problems running the scenario"))
    }
}

#[derive(Clone)]
struct U8Validator {}
impl StringValidator for U8Validator {
    fn validate(&self, input: &str) -> Result<inquire::validator::Validation,
 CustomUserError> {
        if input.parse::<u8>().is_err() {
            Ok(inquire::validator::Validation::Invalid(
                "Can't parse input as number".into(),
            ))
        } else {
            Ok(inquire::validator::Validation::Valid)
        }
    }
}

async fn show_parameters(scenario: &AuroraScenario, warnings: &mut Warnings) {
    let parameters = scenario.cluster_parameters().await;

    match parameters {
        Ok(parameters) => {
            println!("Current parameters");
            for parameter in parameters {
                println!("\t{parameter}");
            }
        }
        Err(error) => warnings.push("Could not find cluster parameters", error),
    }
}

fn prompt_number_or_default(warnings: &mut Warnings, name: &str, default: u8) ->
 u8 {
    let input = inquire::Text::new(format!("Updated {name}:").as_str())
        .with_validator(U8Validator {})
        .prompt();
```

```
    match input {
        Ok(increment) => match increment.parse::<u8>() {
            Ok(increment) => increment,
            Err(error) => {
                warnings.push(
                    format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                    ScenarioError::with(format!("{error}")),
                );
                default
            }
        },
        Err(error) => {
            warnings.push(
                format!("Invalid updated {name} (using {default}
 instead)").as_str(),
                ScenarioError::with(format!("{error}")),
            );
            default
        }
    }
}
```

Pembungkus di sekitar layanan Amazon RDS yang memungkinkan automocking untuk pengujian.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

use aws_sdk_rds::{
    error::SdkError,
    operation::{
        create_db_cluster::{CreateDBClusterError, CreateDbClusterOutput},
        create_db_cluster_parameter_group::CreateDBClusterParameterGroupError,
        create_db_cluster_parameter_group::CreateDbClusterParameterGroupOutput,
        create_db_cluster_snapshot::{CreateDBClusterSnapshotError,
 CreateDbClusterSnapshotOutput},
        create_db_instance::{CreateDBInstanceError, CreateDbInstanceOutput},
        delete_db_cluster::{DeleteDBClusterError, DeleteDbClusterOutput},
        delete_db_cluster_parameter_group::{
```

```
            DeleteDBClusterParameterGroupError,
 DeleteDbClusterParameterGroupOutput,
        },
        delete_db_instance::{DeleteDBInstanceError, DeleteDbInstanceOutput},
        describe_db_cluster_endpoints::{
            DescribeDBClusterEndpointsError, DescribeDbClusterEndpointsOutput,
        },
        describe_db_cluster_parameters::{
            DescribeDBClusterParametersError, DescribeDbClusterParametersOutput,
        },
        describe_db_clusters::{DescribeDBClustersError,
 DescribeDbClustersOutput},
        describe_db_engine_versions::{
            DescribeDBEngineVersionsError, DescribeDbEngineVersionsOutput,
        },
        describe_db_instances::{DescribeDBInstancesError,
 DescribeDbInstancesOutput},

 describe_orderable_db_instance_options::DescribeOrderableDBInstanceOptionsError,
        modify_db_cluster_parameter_group::{
            ModifyDBClusterParameterGroupError,
 ModifyDbClusterParameterGroupOutput,
        },
    },
    types::{OrderableDbInstanceOption, Parameter},
    Client as RdsClient,
};
use secrecy::{ExposeSecret, SecretString};

#[cfg(test)]
use mockall::automock;

#[cfg(test)]
pub use MockRdsImpl as Rds;
#[cfg(not(test))]
pub use RdsImpl as Rds;

pub struct RdsImpl {
    pub inner: RdsClient,
}

#[cfg_attr(test, automock)]
impl RdsImpl {
    pub fn new(inner: RdsClient) -> Self {
```

```
        RdsImpl { inner }
    }

    // snippet-start:[rust.aurora.describe_db_engine_versions.wrapper]
    pub async fn describe_db_engine_versions(
        &self,
        engine: &str,
    ) -> Result<DescribeDbEngineVersionsOutput,
SdkError<DescribeDBEngineVersionsError>> {
        self.inner
            .describe_db_engine_versions()
            .engine(engine)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_engine_versions.wrapper]

    // snippet-start:[rust.aurora.describe_orderable_db_instance_options.wrapper]
    pub async fn describe_orderable_db_instance_options(
        &self,
        engine: &str,
        engine_version: &str,
    ) -> Result<Vec<OrderableDbInstanceOption>,
SdkError<DescribeOrderableDBInstanceOptionsError>>
    {
        self.inner
            .describe_orderable_db_instance_options()
            .engine(engine)
            .engine_version(engine_version)
            .into_paginator()
            .items()
            .send()
            .try_collect()
            .await
    }
    // snippet-end:[rust.aurora.describe_orderable_db_instance_options.wrapper]

    // snippet-start:[rust.aurora.create_db_cluster_parameter_group.wrapper]
    pub async fn create_db_cluster_parameter_group(
        &self,
        name: &str,
        description: &str,
        family: &str,
```

```
    ) -> Result<CreateDbClusterParameterGroupOutput,
SdkError<CreateDBClusterParameterGroupError>>
    {
        self.inner
            .create_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .description(description)
            .db_parameter_group_family(family)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_cluster_parameter_group.wrapper]

    // snippet-start:[rust.aurora.describe_db_clusters.wrapper]
    pub async fn describe_db_clusters(
        &self,
        id: &str,
    ) -> Result<DescribeDbClustersOutput, SdkError<DescribeDBClustersError>> {
        self.inner
            .describe_db_clusters()
            .db_cluster_identifier(id)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_clusters.wrapper]

    // snippet-start:[rust.aurora.describe_db_cluster_parameters.wrapper]
    pub async fn describe_db_cluster_parameters(
        &self,
        name: &str,
    ) -> Result<Vec<DescribeDbClusterParametersOutput>,
SdkError<DescribeDBClusterParametersError>>
    {
        self.inner
            .describe_db_cluster_parameters()
            .db_cluster_parameter_group_name(name)
            .into_paginator()
            .send()
            .try_collect()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_cluster_parameters.wrapper]

    // snippet-start:[rust.aurora.modify_db_cluster_parameter_group.wrapper]
```

```
    pub async fn modify_db_cluster_parameter_group(
        &self,
        name: &str,
        parameters: Vec<Parameter>,
    ) -> Result<ModifyDbClusterParameterGroupOutput,
SdkError<ModifyDBClusterParameterGroupError>>
    {
        self.inner
            .modify_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .set_parameters(Some(parameters))
            .send()
            .await
    }
    // snippet-end:[rust.aurora.modify_db_cluster_parameter_group.wrapper]

    // snippet-start:[rust.aurora.create_db_cluster.wrapper]
    pub async fn create_db_cluster(
        &self,
        name: &str,
        parameter_group: &str,
        engine: &str,
        version: &str,
        username: &str,
        password: SecretString,
    ) -> Result<CreateDbClusterOutput, SdkError<CreateDBClusterError>> {
        self.inner
            .create_db_cluster()
            .db_cluster_identifier(name)
            .db_cluster_parameter_group_name(parameter_group)
            .engine(engine)
            .engine_version(version)
            .master_username(username)
            .master_user_password(password.expose_secret())
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_cluster.wrapper]

    // snippet-start:[rust.aurora.create_db_instance.wrapper]
    pub async fn create_db_instance(
        &self,
        cluster_name: &str,
        instance_name: &str,
```

```
        instance_class: &str,
        engine: &str,
    ) -> Result<CreateDbInstanceOutput, SdkError<CreateDBInstanceError>> {
        self.inner
            .create_db_instance()
            .db_cluster_identifier(cluster_name)
            .db_instance_identifier(instance_name)
            .db_instance_class(instance_class)
            .engine(engine)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_instance.wrapper]

    // snippet-start:[rust.aurora.describe_db_instance.wrapper]
    pub async fn describe_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner
            .describe_db_instances()
            .db_instance_identifier(instance_identifier)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_instance.wrapper]

    // snippet-start:[rust.aurora.create_db_cluster_snapshot.wrapper]
    pub async fn snapshot_cluster(
        &self,
        db_cluster_identifier: &str,
        snapshot_name: &str,
    ) -> Result<CreateDbClusterSnapshotOutput,
SdkError<CreateDBClusterSnapshotError>> {
        self.inner
            .create_db_cluster_snapshot()
            .db_cluster_identifier(db_cluster_identifier)
            .db_cluster_snapshot_identifier(snapshot_name)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.create_db_cluster_snapshot.wrapper]

    // snippet-start:[rust.aurora.describe_db_instances.wrapper]
```

```
    pub async fn describe_db_instances(
        &self,
    ) -> Result<DescribeDbInstancesOutput, SdkError<DescribeDBInstancesError>> {
        self.inner.describe_db_instances().send().await
    }
    // snippet-end:[rust.aurora.describe_db_instances.wrapper]

    // snippet-start:[rust.aurora.describe_db_cluster_endpoints.wrapper]
    pub async fn describe_db_cluster_endpoints(
        &self,
        cluster_identifier: &str,
    ) -> Result<DescribeDbClusterEndpointsOutput,
SdkError<DescribeDBClusterEndpointsError>> {
        self.inner
            .describe_db_cluster_endpoints()
            .db_cluster_identifier(cluster_identifier)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.describe_db_cluster_endpoints.wrapper]

    // snippet-start:[rust.aurora.delete_db_instance.wrapper]
    pub async fn delete_db_instance(
        &self,
        instance_identifier: &str,
    ) -> Result<DeleteDbInstanceOutput, SdkError<DeleteDBInstanceError>> {
        self.inner
            .delete_db_instance()
            .db_instance_identifier(instance_identifier)
            .skip_final_snapshot(true)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.delete_db_instance.wrapper]

    // snippet-start:[rust.aurora.delete_db_cluster.wrapper]
    pub async fn delete_db_cluster(
        &self,
        cluster_identifier: &str,
    ) -> Result<DeleteDbClusterOutput, SdkError<DeleteDBClusterError>> {
        self.inner
            .delete_db_cluster()
            .db_cluster_identifier(cluster_identifier)
            .skip_final_snapshot(true)
```

```
            .send()
            .await
    }
    // snippet-end:[rust.aurora.delete_db_cluster.wrapper]

    // snippet-start:[rust.aurora.delete_db_cluster_parameter_group.wrapper]
    pub async fn delete_db_cluster_parameter_group(
        &self,
        name: &str,
    ) -> Result<DeleteDbClusterParameterGroupOutput,
 SdkError<DeleteDBClusterParameterGroupError>>
    {
        self.inner
            .delete_db_cluster_parameter_group()
            .db_cluster_parameter_group_name(name)
            .send()
            .await
    }
    // snippet-end:[rust.aurora.delete_db_cluster_parameter_group.wrapper]
}
```

Cargo.toml dengan dependensi yang digunakan dalam skenario ini.

```
[package]
name = "aurora-code-examples"
authors = [
  "David Souther <dpsouth@amazon.com>",
]
edition = "2021"
version = "0.1.0"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/
reference/manifest.html

[dependencies]
anyhow = "1.0.75"
assert_matches = "1.5.0"
aws-config = { version = "1.0.1", features = ["behavior-version-latest"] }
aws-smithy-types = { version = "1.0.1" }
aws-smithy-runtime-api = { version = "1.0.1" }
aws-sdk-rds = { version = "1.3.0" }
inquire = "0.6.2"
```

```
mockall = "0.11.4"
phf = { version = "0.11.2", features = ["std", "macros"] }
sdk-examples-test-utils = { path = "../../test-utils" }
secrecy = "0.8.0"
tokio = { version = "1.20.1", features = ["full", "test-util"] }
tracing = "0.1.37"
tracing-subscriber = { version = "0.3.15", features = ["env-filter"] }
```

- Untuk detail API, lihat topik berikut di Referensi API AWS SDK for Rust.
  - CreateDBCluster
  - dibuatB ClusterParameterGroup
  - dibuatB ClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DihapusB ClusterParameterGroup
  - DeleteDBInstance
  - DijelaskanB ClusterParameterGroups
  - DijelaskanB ClusterParameters
  - DijelaskanB ClusterSnapshots
  - DescribeDBClusters
  - DijelaskanB EngineVersions
  - DescribeDBInstances
  - DescribeOrderableDB InstanceOptions
  - ModifyDB ClusterParameterGroup

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Contoh lintas layanan untuk AWS Aurora menggunakan SDK

Contoh aplikasi berikut menggunakan AWS SDK untuk menggabungkan Aurora dengan yang lain. Layanan AWS Setiap contoh menyertakan tautan ke GitHub, di mana Anda dapat menemukan petunjuk tentang cara mengatur dan menjalankan aplikasi.

Contoh-contoh

- Membuat API REST pustaka peminjaman
- Buat pelacak butir kerja Aurora Nirserver

Membuat API REST pustaka peminjaman

Contoh kode berikut menunjukkan cara membuat pustaka peminjaman tempat pelanggan dapat meminjam dan mengembalikan buku dengan menggunakan API REST yang didukung oleh basis data Amazon Aurora.

Python

SDK untuk Python (Boto3)

Menunjukkan cara menggunakan API Amazon Relational Database Service (Amazon RDS) dan AWS Chalice untuk membuat REST API yang didukung oleh database Amazon Aurora. AWS SDK for Python (Boto3) Layanan web sepenuhnya nirserver dan mewakili pustaka peminjaman sederhana tempat pelanggan dapat meminjam dan mengembalikan buku. Pelajari cara:

- Membuat dan mengelola klaster basis data Aurora nirserver.
- Gunakan AWS Secrets Manager untuk mengelola kredensi database.
- Menerapkan lapisan penyimpanan data yang menggunakan Amazon RDS untuk memindahkan data masuk dan keluar dari basis data.
- Gunakan AWS Chalice untuk menerapkan REST API tanpa server ke Amazon API Gateway dan. AWS Lambda
- Menggunakan paket Permintaan untuk mengirim permintaan ke layanan web.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- API Gateway
- Aurora
- Lambda
- Secrets Manager

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Buat pelacak butir kerja Aurora Nirserver

Contoh-contoh kode berikut menunjukkan cara membuat aplikasi web yang melacak butir kerja dalam basis data Amazon Aurora Nirserver dan menggunakan Amazon Simple Email Service (Amazon SES) untuk mengirim laporan.

.NET

AWS SDK for .NET

Menunjukkan cara menggunakan AWS SDK for .NET untuk membuat aplikasi web yang melacak item pekerjaan dalam database Amazon Aurora dan laporan email dengan menggunakan Amazon Simple Email Service (Amazon SES). Contoh ini menggunakan sisi depan yang dibangun dengan React.js untuk berinteraksi dengan backend RESTful .NET.

- Integrasikan aplikasi web React dengan AWS layanan.
- Cantumkan, tambahkan, perbarui, dan hapus butir di tabel Aurora.
- Kirim laporan email tentang butir kerja terfilter dengan menggunakan Amazon SES.
- Menyebarkan dan mengelola sumber daya contoh dengan AWS CloudFormation skrip yang disertakan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS
- Layanan Data Amazon RDS
- Amazon SES

C++

SDK for C++

Menunjukkan cara membuat aplikasi web yang melacak dan melaporkan butir kerja yang tersimpan dalam basis data Amazon Aurora Nirserver.

Untuk kode sumber lengkap dan instruksi tentang cara menyiapkan C++ REST API yang menanyakan data Amazon Aurora Tanpa Server dan untuk digunakan oleh aplikasi React, lihat contoh lengkapnya di. GitHub

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS
- Layanan Data Amazon RDS
- Amazon SES

Java

SDK for Java 2.x

Menunjukkan cara membuat aplikasi web yang melacak dan melaporkan butir kerja yang tersimpan dalam basis data Amazon RDS.

Untuk kode sumber lengkap dan petunjuk tentang cara menyiapkan Spring REST API yang menanyakan data Amazon Aurora Tanpa Server dan untuk digunakan oleh aplikasi React, lihat contoh lengkapnya di. GitHub

Untuk kode sumber lengkap dan instruksi tentang cara menyiapkan dan menjalankan contoh yang menggunakan JDBC API, lihat contoh lengkapnya di. GitHub

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS
- Layanan Data Amazon RDS
- Amazon SES

JavaScript

SDK untuk JavaScript (v3)

Menunjukkan cara menggunakan AWS SDK for JavaScript (v3) untuk membuat aplikasi web yang melacak item pekerjaan dalam database Amazon Aurora dan laporan email dengan menggunakan Amazon Simple Email Service (Amazon SES). Contoh ini menggunakan sisi depan yang dibangun dengan React.js untuk berinteraksi dengan backend Express Node.js.

- Integrasikan aplikasi web React.js dengan Layanan AWS.
- Cantumkan, tambahkan, dan perbarui butir di tabel Aurora.
- Kirim laporan email tentang butir kerja terfilter dengan menggunakan Amazon SES.
- Menyebarkan dan mengelola sumber daya contoh dengan AWS CloudFormation skrip yang disertakan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS
- Layanan Data Amazon RDS
- Amazon SES

Kotlin

SDK for Kotlin

Menunjukkan cara membuat aplikasi web yang melacak dan melaporkan butir kerja yang tersimpan dalam basis data Amazon RDS.

Untuk kode sumber lengkap dan petunjuk tentang cara menyiapkan Spring REST API yang menanyakan data Amazon Aurora Tanpa Server dan untuk digunakan oleh aplikasi React, lihat contoh lengkapnya di. GitHub

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS

- Layanan Data Amazon RDS
- Amazon SES

PHP

SDK for PHP

Menunjukkan cara menggunakan AWS SDK for PHP untuk membuat aplikasi web yang melacak item pekerjaan dalam database Amazon RDS dan laporan email dengan menggunakan Amazon Simple Email Service (Amazon SES). Contoh ini menggunakan sisi depan yang dibangun dengan React.js untuk berinteraksi dengan backend RESTful PHP.

- Integrasikan aplikasi web React.js dengan AWS layanan.
- Cantumkan, tambahkan, perbarui, dan hapus butir di tabel Amazon RDS.
- Kirim laporan email tentang butir kerja terfilter dengan menggunakan Amazon SES.
- Menyebarkan dan mengelola sumber daya contoh dengan AWS CloudFormation skrip yang disertakan.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS
- Layanan Data Amazon RDS
- Amazon SES

Python

SDK for Python (Boto3)

Menunjukkan cara menggunakan AWS SDK for Python (Boto3) untuk membuat layanan REST yang melacak item pekerjaan di database Amazon Aurora Tanpa Server dan laporan email dengan menggunakan Amazon Simple Email Service (Amazon SES). Contoh ini menggunakan rangka kerja web Flask untuk menangani perutean HTTP dan terintegrasi dengan halaman web React untuk menyajikan aplikasi web yang berfungsi penuh.

- Bangun layanan Flask REST yang terintegrasi dengan. Layanan AWS

- Baca, tulis, dan perbarui butir kerja yang tersimpan dalam basis data Aurora Nirserver.
- Buat AWS Secrets Manager rahasia yang berisi kredensi database dan gunakan untuk mengautentikasi panggilan ke database.
- Gunakan Amazon SES untuk mengirim laporan email tentang butir kerja.

Untuk kode sumber lengkap dan instruksi tentang cara mengatur dan menjalankan, lihat contoh lengkapnya di GitHub.

Layanan yang digunakan dalam contoh ini

- Aurora
- Amazon RDS
- Layanan Data Amazon RDS
- Amazon SES

Untuk daftar lengkap panduan pengembang AWS SDK dan contoh kode, lihatMenggunakan layanan ini dengan AWS SDK. Topik ini juga mencakup informasi tentang cara memulai dan detail versi-versi SDK sebelumnya.

Praktik terbaik dengan Amazon Aurora

Di bagian berikut ini, Anda dapat menemukan informasi tentang praktik terbaik dan opsi untuk menggunakan atau memigrasikan data ke klaster DB Amazon Aurora.

Beberapa praktik terbaik untuk Amazon Aurora yang dikhususkan untuk mesin basis data tertentu. Untuk informasi selengkapnya tentang praktik terbaik Aurora yang dikhususkan untuk sebuah mesin basis data, lihat hal berikut.

Mesin basis data	Praktik terbaik
Amazon Aurora MySQL	Lihat Praktik terbaik dengan Amazon Aurora MySQL
Amazon Aurora PostgreSQL	Lihat Praktik terbaik dengan Amazon Aurora PostgreSQL

> **Note**
>
> Untuk rekomendasi umum terkait Aurora, lihat Melihat dan menanggapi rekomendasi Amazon RDS.

Topik

- Pedoman operasional dasar untuk Amazon Aurora
- Rekomendasi RAM instans DB
- Memantau Amazon Aurora
- Menggunakan grup parameter DB dan grup parameter klaster DB
- Video praktik terbaik Amazon Aurora

Pedoman operasional dasar untuk Amazon Aurora

Berikut ini adalah pedoman operasional dasar yang harus diikuti setiap orang saat menggunakan Amazon Aurora. Perjanjian Tingkat Layanan Amazon RDS mewajibkan Anda untuk mengikuti pedoman berikut ini:

- Memantau penggunaan memori, CPU, dan penyimpanan Anda. Anda dapat mengatur Amazon CloudWatch untuk memberi tahu Anda saat pola penggunaan berubah atau saat Anda mendekati kapasitas penerapan. Dengan begitu, Anda dapat mempertahankan performa sistem dan ketersediaan.
- Jika aplikasi klien Anda menyimpan data Domain Name Service (DNS) dari instans DB Anda, tetapkan nilai time-to-live (TTL) kurang dari 30 detik. Alamat IP yang mendasari dari instans DB dapat berubah setelah failover. Dengan demikian, meng-cache data DNS untuk waktu yang lama dapat menyebabkan kegagalan koneksi jika aplikasi Anda mencoba terhubung ke alamat IP yang tidak lagi digunakan. Klaster DB Aurora dengan beberapa replika baca juga dapat mengalami kegagalan koneksi ketika koneksi menggunakan titik akhir pembaca dan salah satu instans replika baca berada dalam pemeliharaan atau dihapus.
- Uji failover klaster DB Anda untuk memahami berapa lama prosesnya untuk kasus penggunaan Anda. Pengujian failover dapat membantu Anda memastikan bahwa aplikasi yang mengakses klaster DB Anda dapat secara otomatis terhubung ke klaster DB baru setelah failover.

Rekomendasi RAM instans DB

Untuk mengoptimalkan performa, alokasikan RAM yang cukup sehingga working set berada hampir sepenuhnya di dalam memori. Untuk menentukan apakah perangkat kerja Anda sudah hampir seluruhnya dalam memori, AMI metrik berikut di Amazon CloudWatch:

- `VolumeReadIOPS` – Metrik ini mengukur jumlah rata-rata operasi I/O baca dari volume klaster, yang dilaporkan pada interval 5 menit. Nilai `VolumeReadIOPS` harus kecil dan stabil. Dalam beberapa kasus, Anda mungkin menemukan I/O baca Anda melonjak atau lebih tinggi dari biasanya. Jika demikian, selidiki instans DB dalam klaster DB Anda untuk melihat instans DB mana yang menyebabkan peningkatan I/O.

> **Tip**
>
> Jika klaster Aurora MySQL Anda menggunakan kueri paralel, Anda mungkin melihat peningkatan nilai `VolumeReadIOPS`. Kueri paralel tidak menggunakan kumpulan buffer. Jadi, meskipun kuerinya cepat, pemrosesan yang dioptimalkan ini dapat menghasilkan peningkatan operasi baca dan biaya terkait.

- `BufferCacheHitRatio` – Metrik ini mengukur persentase permintaan yang dilayani oleh cache buffer instans DB dalam klaster DB Anda. Metrik ini memberikan wawasan tentang jumlah data yang dilayani dari memori.

  Rasio hit tinggi menunjukkan bahwa instans DB Anda memiliki cukup memori yang tersedia. Rasio hit rendah menunjukkan bahwa kueri Anda pada instans DB ini sering kali masuk ke disk. Selidiki beban kerja Anda untuk melihat kueri mana yang menyebabkan perilaku ini.

Jika, setelah menyelidiki beban kerja Anda, Anda mendapati bahwa Anda memerlukan lebih banyak memori, pertimbangkan untuk menaikkan skala kelas instans DB ke kelas yang memiliki lebih banyak RAM. Setelah itu, Anda dapat menyelidiki metrik yang dibahas sebelumnya dan terus menaikkan skalanya sesuai kebutuhan. Jika klaster Aurora Anda lebih besar dari 40 TB, jangan gunakan kelas instans db.t2, db.t3, atau db.t4g.

Untuk informasi selengkapnya, lihat CloudWatch Metrik Amazon untuk Amazon Aurora.

Memantau Amazon Aurora

Amazon Aurora menyediakan berbagai metrik dan wawasan yang dapat Anda pantau untuk menentukan kondisi dan performa klaster DB Aurora Anda. Anda dapat menggunakan berbagai alat, seperti,, dan CloudWatch API AWS Management ConsoleAWS CLI, untuk melihat metrik Aurora. Anda dapat melihat gabungan Performance Insights dan CloudWatch metrik di dasbor Performance Insights dan memantau instans DB Anda. Untuk menggunakan tampilan pemantauan ini, Wawasan Performa harus diaktifkan untuk instans DB Anda. Untuk informasi tentang tampilan pemantauan ini, lihat Menampilkan metrik gabungan di konsol Amazon RDS.

Anda dapat membuat laporan analisis performa untuk periode waktu tertentu dan melihat wawasan yang diidentifikasi serta rekomendasi untuk menyelesaikan masalah. Untuk informasi selengkapnya, lihat Membuat laporan analisis performa.

Menggunakan grup parameter DB dan grup parameter klaster DB

Kami menyarankan agar Anda mencoba perubahan grup parameter DB dan grup parameter klaster DB pada klaster DB uji sebelum menerapkan perubahan grup parameter ke klaster DB produksi Anda. Pengaturan parameter mesin DB yang tidak tepat dapat menimbulkan efek merugikan yang tidak diinginkan, termasuk penurunan performa dan ketidakstabilan sistem.

Berhati-hatilah selalu saat memodifikasi parameter mesin DB dan cadangkan klaster DB sebelum memodifikasi grup parameter DB. Untuk informasi tentang pencadangan klaster DB Anda, lihat Mencadangkan dan memulihkan klaster DB Amazon Aurora.

Video praktik terbaik Amazon Aurora

Saluran AWS Online Tech Talks on YouTube menyertakan presentasi video tentang praktik terbaik untuk membuat dan mengonfigurasi kluster Amazon Aurora DB agar lebih aman dan sangat tersedia. Lihat Praktik terbaik Amazon Aurora untuk ketersediaan tinggi.

Melakukan pembuktian konsep dengan Amazon Aurora

Dalam informasi berikut ini, Anda dapat menemukan penjelasan tentang cara menyiapkan dan menjalankan bukti konsep untuk Aurora. Pembuktian konsep adalah penyelidikan yang Anda lakukan untuk melihat apakah Aurora cocok dengan aplikasi Anda. Pembuktian konsep dapat membantu Anda memahami fitur Aurora dalam konteks aplikasi basis data Anda sendiri dan perbandingan Aurora dengan lingkungan basis data Anda saat ini. Pembuktian konsep juga dapat menunjukkan tingkat upaya yang Anda butuhkan untuk memindahkan data, mem-porting kode SQL, menyetel performa, dan menyesuaikan prosedur manajemen Anda saat ini.

Dalam topik ini, Anda dapat menemukan ikhtisar dan step-by-step garis besar prosedur dan keputusan tingkat tinggi yang terlibat dalam menjalankan bukti konsep, tercantum berikut. Untuk petunjuk mendetail, Anda dapat membuka tautan ke dokumentasi lengkap untuk subjek spesifik.

Gambaran umum bukti konsep Aurora

Saat Anda melakukan pembuktian konsep untuk Amazon Aurora, Anda akan mempelajari apa saja yang diperlukan untuk memindahkan data yang ada dan aplikasi SQL Anda ke Aurora. Anda mencoba aspek penting Aurora dalam skala besar, menggunakan volume data dan aktivitas yang sesuai dengan lingkungan produksi Anda. Tujuannya adalah untuk menjadi yakin bahwa kecanggihan Aurora memang cocok untuk mengatasi tantangan yang menyebabkan Anda tidak dapat lagi mengandalkan infrastruktur basis data sebelumnya. Pada akhir pembuktian konsep, Anda akan memiliki rencana yang andal untuk melakukan penilaian tolok ukur performa dan pengujian aplikasi berskala lebih besar. Pada tahap ini, Anda memahami item pekerjaan terbesar dalam proses menuju deployment produksi.

Saran berikut tentang praktik terbaik dapat membantu Anda menghindari kesalahan umum yang dapat menyebabkan masalah selama proses penilaian tolok ukur. Namun, topik ini tidak mencakup step-by-step proses melakukan tolok ukur dan melakukan penyetelan kinerja. Prosedur tersebut bisa bervariasi, tergantung pada beban kerja Anda dan fitur Aurora yang Anda gunakan. Untuk informasi selengkapnya, baca dokumentasi yang terkait dengan performa seperti Mengelola performa dan penskalaan untuk klaster DB Aurora, Peningkatan performa Amazon Aurora MySQL, Mengelola Amazon Aurora PostgreSQL, dan Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

Informasi dalam topik ini berlaku terutama untuk aplikasi tempat organisasi Anda menulis kode dan mendesain skema, serta yang mendukung mesin basis data sumber terbuka MySQL dan

PostgreSQL. Jika Anda menguji aplikasi komersial atau kode yang dihasilkan oleh kerangka kerja aplikasi, Anda mungkin tidak memiliki fleksibilitas untuk menerapkan semua pedoman. Dalam kasus semacam itu, tanyakan kepada perwakilan AWS Anda untuk melihat apakah ada praktik terbaik Aurora atau studi kasus yang cocok untuk jenis aplikasi Anda.

1. Identifikasi tujuan Anda

Saat Anda mengevaluasi Aurora sebagai bagian dari pembuktian konsep, Anda perlu memilih jenis pengukuran apa yang akan dilakukan dan cara mengevaluasi keberhasilan percobaan ini.

Anda harus memastikan bahwa semua fungsionalitas aplikasi Anda kompatibel dengan Aurora. Karena versi mayor Aurora kompatibel kabel dengan versi mayor MySQL dan PostgreSQL, sebagian besar aplikasi yang dikembangkan untuk mesin tersebut juga kompatibel dengan Aurora. Meski demikian, Anda masih harus memvalidasi kompatibilitas per aplikasi.

Misalnya, beberapa pilihan konfigurasi yang Anda buat saat menyiapkan klaster Aurora akan memengaruhi apakah Anda bisa atau harus menggunakan fitur basis data tertentu. Anda dapat memulai dengan klaster Aurora yang paling umum, yang dikenal sebagai klaster terprovisi. Anda kemudian dapat memutuskan apakah konfigurasi khusus seperti kueri nirserver atau kueri paralel mampu memberi manfaat untuk menangani beban kerja Anda.

Gunakan pertanyaan-pertanyaan berikut untuk membantu mengidentifikasi dan mengukur tujuan Anda:

- Apakah Aurora mendukung semua kasus penggunaan fungsional beban kerja Anda?
- Berapa ukuran set data atau tingkat beban yang Anda inginkan? Bisakah Anda meningkatkan skala ke tingkat itu?
- Apa persyaratan latensi atau throughput kueri spesifik Anda? Bisakah Anda memenuhi persyaratannya?
- Berapa lama waktu henti minimum terencana atau tidak terencana yang Anda sanggupi untuk beban kerja Anda? Bisakah Anda mencapainya?
- Apa metrik yang diperlukan untuk meraih efisiensi operasional? Bisakah Anda memantaunya secara akurat?
- Apakah Aurora mendukung tujuan bisnis spesifik Anda, seperti pengurangan biaya, peningkatan deployment, atau kecepatan penyediaan? Apakah Anda memiliki cara untuk mengukur tujuan ini?
- Dapatkah Anda memenuhi semua persyaratan keamanan dan kepatuhan untuk beban kerja Anda?

Luangkan waktu untuk membangun pengetahuan tentang mesin basis data dan kemampuan platform Aurora, serta tinjau dokumentasi layanan ini. Catat semua fitur yang dapat membantu Anda mencapai hasil yang Anda inginkan. Salah satunya mungkin berupa konsolidasi beban kerja, yang dijelaskan dalam postingan Blog Basis Data AWS Cara merencanakan dan mengoptimalkan Amazon Aurora dengan kompatibilitas MySQL untuk beban kerja terkonsolidasi. Fitur lainnya adalah penskalaan berbasis permintaan, yang dijelaskan dalam Menggunakan Amazon Aurora Auto Scaling dengan Replika Aurora di Panduan Pengguna Amazon Aurora. Fiturnya yang lain adalah keuntungan performa atau operasi basis data yang disederhanakan.

2. Pahami karakteristik beban kerja Anda

Evaluasi Aurora dalam konteks kasus penggunaan yang Anda inginkan. Aurora adalah pilihan yang cocok untuk beban kerja pemrosesan transaksi online (OLTP). Anda juga bisa menjalankan laporan di klaster yang menyimpan data OLTP waktu nyata tanpa harus menyediakan klaster gudang data terpisah. Anda dapat mengenali apakah kasus penggunaan Anda termasuk dalam kategori ini atau tidak dengan melihat ciri-ciri berikut:

- Konkurensi yang tinggi, dengan puluhan, ratusan, atau bahkan ribuan klien simultan.
- Kueri berlatensi rendah dalam volume besar (milidetik hingga detik).
- Transaksi singkat dan waktu nyata.
- Pola kueri yang sangat selektif, dengan pencarian berbasis indeks.
- Untuk HTAP, kueri analitik yang bisa memanfaatkan kueri paralel Aurora.

Salah satu faktor utama yang memengaruhi pilihan basis data Anda adalah kecepatan datanya. Kecepatan tinggi berkaitan dengan data yang sangat sering disisipkan dan diperbarui. Sistem seperti itu mungkin memiliki ribuan koneksi dan ratusan ribu kueri simultan yang membaca dan menulis ke basis data. Kueri dalam sistem berkecepatan tinggi biasanya akan memengaruhi jumlah baris yang relatif kecil, dan biasanya mengakses beberapa kolom dalam baris yang sama.

Aurora dirancang untuk menangani data berkecepatan tinggi. Bergantung pada beban kerjanya, klaster Aurora dengan satu instans DB r4.16xlarge dapat memproses lebih dari 600.000 pernyataan `SELECT` per detik. Sekali lagi, bergantung pada beban kerjanya, klaster seperti itu dapat memproses 200.000 pernyataan `INSERT`, `UPDATE`, dan `DELETE` per detik. Aurora adalah basis data penyimpanan baris dan sangat cocok untuk beban kerja OLTP yang memiliki volume tinggi, throughput tinggi, dan paralelisasi tinggi.

Aurora juga dapat menjalankan kueri pelaporan pada klaster yang sama yang menangani beban kerja OLTP. Aurora mendukung hingga 15 replika, yang masing-masing memiliki lag replikasi rata-rata 10-20 milidetik dari instans primer. Analis dapat mengkueri data OLTP secara waktu nyata tanpa menyalin data ke klaster gudang data terpisah. Dengan menggunakan fitur kueri paralel untuk klaster Aurora, Anda dapat memindahkan sebagian besar pemrosesan, pemfilteran, dan agregasi ke subsistem penyimpanan Aurora yang didistribusikan secara masif.

Gunakan fase perencanaan ini agar Anda mengenali kemampuan Aurora, layanan AWS lainnya, AWS Management Console, dan AWS CLI. Selain itu, periksa bagaimana kecocokan fungsinya dengan alat-alat lain yang nantinya Anda gunakan dalam pembuktian konsep.

3. Berlatih dengan AWS Management Console atau AWS CLI

Untuk langkah berikutnya, berlatihlah dengan AWS Management Console atau AWS CLI, untuk membiasakan diri dengan kedua alat ini dan Aurora.

Berlatih dengan AWS Management Console

Aktivitas awal berikut dengan klaster basis data Aurora terutama dimaksudkan agar Anda dapat membiasakan diri dengan lingkungan AWS Management Console serta berlatih menyiapkan dan memodifikasi klaster Aurora. Jika menggunakan mesin basis data yang kompatibel dengan MySQL dan PostgreSQL bersama Amazon RDS, Anda dapat memanfaatkan pengetahuan tersebut saat menggunakan Aurora.

Dengan memanfaatkan model dan fitur penyimpanan bersama dari Aurora seperti replikasi dan snapshot, Anda dapat memperlakukan seluruh klaster basis data sebagai jenis objek lain yang dapat Anda manipulasi dengan bebas. Anda dapat menyusun, merombak, dan mengubah kapasitas klaster Aurora selama proses pembuktian konsep. Anda tidak akan terkunci pada pilihan awal tentang kapasitas, pengaturan basis data, dan tata letak data fisik.

Untuk memulai, siapkan klaster Aurora yang kosong. Pilih jenis kapasitas terprovisi dan lokasi wilayah untuk eksperimen awal Anda.

Hubungkan ke klaster tersebut menggunakan program klien seperti aplikasi baris perintah SQL. Awalnya, Anda akan terhubung menggunakan titik akhir klaster. Anda terhubung ke titik akhir tersebut untuk melakukan operasi tulis apa pun, seperti proses pernyataan bahasa definisi data (DDL) dan proses extract, transform, and load (ETL). Selanjutnya, dalam pembuktian konsep, Anda perlu menghubungkan sesi sarat kueri menggunakan titik akhir pembaca, yang mendistribusikan beban kerja kueri ke beberapa instans DB di klaster.

Skalakan ke luar klaster dengan menambahkan lebih banyak Replika Aurora. Untuk prosedurnya, lihat Replikasi dengan Amazon Aurora. Naikkan atau turunkan skala instans DB dengan mengubah kelas instans AWS. Pahami cara Aurora dapat menyederhanakan jenis operasi ini, sehingga jika perkiraan awal Anda untuk kapasitas sistem tidak akurat, Anda dapat menyesuaikannya nanti tanpa harus memulai dari awal.

Buat snapshot dan pulihkan ke klaster yang berbeda.

Periksa metrik klaster untuk melihat aktivitas dari waktu ke waktu, dan kecocokan metrik tersebut untuk instans DB di klaster.

Sebaiknya pahami cara melakukannya melalui AWS Management Console sejak awal. Setelah memahami apa yang dapat Anda lakukan dengan Aurora, Anda dapat melanjutkan untuk mengotomatiskan operasi tersebut menggunakan AWS CLI. Di bagian berikut, Anda dapat menemukan detail lebih lanjut tentang prosedur dan praktik terbaik untuk kegiatan ini selama proof-of-concept periode tersebut.

Berlatih dengan AWS CLI

Kami merekomendasikan untuk mengotomatiskan prosedur penerapan dan manajemen, bahkan dalam pengaturan. proof-of-concept Untuk melakukannya, pelajari AWS CLI terlebih dahulu jika belum. Jika menggunakan mesin basis data yang kompatibel dengan MySQL dan PostgreSQL bersama Amazon RDS, Anda dapat memanfaatkan pengetahuan tersebut saat menggunakan Aurora.

Aurora biasanya menangani grup instans DB yang disusun dalam klaster. Dengan demikian, banyak operasi akan mengharuskan Anda menentukan instans DB mana yang terkait dengan sebuah klaster, lalu melakukan operasi administratif dalam satu rangkaian untuk semua instans.

Misalnya, Anda dapat mengotomatiskan langkah-langkah seperti pembuatan klaster Aurora, lalu menaikkan skalanya dengan kelas instans yang lebih besar atau menskalakan klaster tersebut ke luar dengan instans DB tambahan. Dengan demikian, Anda akan dapat mengulangi tahapan mana pun dalam proses pembuktian konsep Anda dan mengkaji skenario bagaimana-jika dengan berbagai jenis atau konfigurasi klaster Aurora.

Pelajari kemampuan dan batasan alat deployment infrastruktur seperti AWS CloudFormation. Anda mungkin menemukan aktivitas yang Anda lakukan dalam proof-of-concept konteks tidak cocok untuk penggunaan produksi. Misalnya, perilaku AWS CloudFormation untuk modifikasi adalah membuat instans baru dan menghapus instans saat ini, termasuk data di dalamnya. Untuk detail selengkapnya

tentang perilaku ini, lihat Pembaruan perilaku sumber daya tumpukan dalam Panduan Pengguna AWS CloudFormation.

4. Buat klaster Aurora Anda

Dengan Aurora, Anda dapat mengkaji skenario bagaimana-jika dengan menambahkan instans DB ke klaster dan menaikkan skala instans DB ke kelas instans yang berperforma lebih tinggi. Anda juga dapat membuat klaster dengan pengaturan konfigurasi yang berbeda-beda untuk menjalankan beban kerja yang sama secara berdampingan. Dengan Aurora, Anda memiliki banyak fleksibilitas untuk menyiapkan, merombak, dan mengonfigurasi ulang klaster DB. Mengingat hal ini, akan sangat membantu untuk mempraktikkan teknik-teknik ini pada tahap awal proof-of-concept proses. Untuk prosedur umum membuat klaster Aurora, lihat Membuat klaster DB Amazon Aurora.

Jika memungkinkan, mulailah dengan sebuah klaster yang menggunakan pengaturan berikut. Lewati langkah ini jika Anda memiliki kasus penggunaan khusus tertentu. Misalnya, Anda mungkin perlu melewati langkah ini jika kasus penggunaan Anda memerlukan jenis klaster Aurora khusus. Atau Anda dapat melewati langkah ini jika Anda memerlukan kombinasi mesin dan versi basis data khusus.

- Nonaktifkan Pembuatan mudah. Sebagai pembuktian konsep, kami menyarankan Anda untuk mengetahui semua pengaturan yang Anda pilih sehingga Anda dapat membuat klaster yang identik atau sedikit berbeda nantinya.
- Gunakan versi mesin DB terbaru. Kombinasi mesin database dan versi ini memiliki kompatibilitas yang luas dengan fitur Aurora lainnya dan penggunaan pelanggan yang substansif untuk aplikasi produksi.
  - Aurora MySQL versi 3.x (MySQL 8.0 kompatibilitas)
  - Aurora PostgreSQL versi 15.x atau 16.x
- Pilih templat Dev/Tes. Pilihan ini tidak signifikan untuk proof-of-concept aktivitas Anda.
- Untuk Kelas instans DB, pilih Kelas teroptimasi memori dan salah satu kelas instans xlarge. Anda dapat menyesuaikan kelas instans ke atas atau ke bawah nantinya.
- Di bagian Deployment Multi-AZ, pilih Buat Replika Aurora atau simpul Pembaca di AZ yang berbeda. Banyak aspek Aurora yang paling berguna menangani klaster yang terdiri dari beberapa instans DB. Sebaiknya mulai selalu dengan setidaknya dua instans DB di klaster baru. Penggunaan Zona Ketersediaan yang berbeda untuk instans DB kedua akan membantu menguji berbagai skenario ketersediaan tinggi.

- Saat memilih nama untuk instans DB, gunakan konvensi penamaan yang generik. Jangan merujuk ke instans DB cluster apa pun sebagai “penulis,” karena instance DB yang berbeda mengambil peran tersebut sesuai kebutuhan. Kami merekomendasikan format seperti `clustername-az-serialnumber`, misalnya `myprodappdb-a-01`. Bagian-bagian ini mengidentifikasi secara unik instans DB dan penempatannya.
- Tetapkan retensi cadangan yang tinggi untuk klaster Aurora. Dengan periode retensi yang lama, Anda dapat melakukan point-in-time pemulihan (PITR) untuk jangka waktu hingga 35 hari. Anda bisa mengatur ulang basis data Anda ke keadaan yang diketahui setelah menjalankan tes yang menangani DDL dan pernyataan bahasa manipulasi data (Data Manipulation Language atau DML). Anda juga dapat memulihkannya jika Anda tidak sengaja menghapus atau mengubah data.
- Aktifkan fitur pemulihan, logging, dan pemantauan tambahan pada pembuatan klaster. Aktifkan semua pilihan yang tersedia di Backtrack, Performance Insights, Monitoring, dan ekspor Log. Dengan mengaktifkan berbagai fitur ini, Anda dapat menguji kesesuaian fitur seperti backtracking, Pemantauan yang Ditingkatkan, atau Wawasan Performa untuk beban kerja Anda. Anda juga dapat dengan mudah memeriksa performa dan melakukan pemecahan masalah selama proses pembuktian konsep.

5. Siapkan skema Anda

Pada klaster Aurora, siapkan basis data, tabel, indeks, kunci asing, dan objek skema lainnya untuk aplikasi Anda. Jika Anda pindah dari sistem basis data yang kompatibel dengan MySQL atau kompatibel dengan PostgreSQL, maka tahap ini kemungkinan akan bisa dijalankan secara sederhana dan mudah. Anda menggunakan sintaksis dan baris perintah SQL yang sama atau aplikasi klien lain yang Anda kenal untuk mesin basis data Anda.

Untuk mengeluarkan pernyataan SQL di klaster Anda, temukan titik akhir klasternya dan berikan nilai tersebut sebagai parameter koneksi ke aplikasi klien Anda. Anda dapat menemukan titik akhir klaster di tab Konektivitas di halaman detail klaster Anda. Titik akhir klaster adalah yang berlabel Penulis. Titik akhir lainnya, yang berlabel Pembaca, merepresentasikan koneksi hanya baca yang dapat Anda berikan kepada pengguna akhir yang menjalankan laporan atau kueri hanya baca lainnya. Untuk memperoleh bantuan atas masalah apa pun terkait koneksi ke klaster Anda, lihat Menghubungkan ke klaster DB Amazon Aurora.

Jika Anda mem-porting skema dan data Anda dari sistem basis data yang berbeda, Anda perlu membuat beberapa perubahan skema pada saat ini. Perubahan skema ini harus sesuai dengan sintaksis dan kemampuan SQL yang tersedia di Aurora. Anda dapat menghapus kolom, batasan, pemicu, atau objek skema tertentu pada tahap ini. Dengan melakukannya, Anda akan memperoleh

manfaat, terutama jika objek ini memerlukan pengerjaan ulang untuk kompatibilitas Aurora dan tidak signifikan untuk tujuan Anda dengan pembuktian konsep.

Jika Anda bermigrasi dari sistem basis data dengan mesin dasar yang berbeda dari Aurora, pertimbangkan untuk menggunakan AWS Schema Conversion Tool (AWS SCT) untuk menyederhanakan prosesnya. Untuk detailnya, lihat Panduan Pengguna AWS Schema Conversion Tool. Untuk detail umum tentang aktivitas migrasi dan porting, lihat laporan resmi AWS Memigrasikan Basis Data Anda ke Amazon Aurora.

Selama tahap ini, Anda dapat mengevaluasi apakah terdapat inefisiensi dalam penyiapan skema Anda, misalnya dalam strategi pengindeksan atau struktur tabel lain seperti tabel yang dipartisi. Inefisiensi seperti itu akan meningkat saat Anda men-deploy aplikasi Anda pada klaster dengan beberapa instans DB dan beban kerja yang berat. Pertimbangkan apakah Anda dapat menyempurnakan aspek performa tersebut sekarang, atau selama aktivitas selanjutnya seperti pengujian tolok ukur lengkap.

6. Impor data Anda

Selama pembuktian konsep, Anda akan membawa data, atau sampel representatif, dari sistem basis data sebelumnya. Jika memungkinkan, siapkan setidaknya beberapa data di setiap tabel Anda. Hal tersebut akan membantu menguji kompatibilitas semua jenis data dan fitur skema. Setelah Anda mencoba berbagai fitur Aurora dasar, tingkatkan jumlah datanya. Pada saat Anda menyelesaikan pembuktian konsep, Anda harus menguji alat ETL, kueri, dan beban kerja Anda secara keseluruhan dengan set data yang cukup besar untuk mengambil kesimpulan secara akurat.

Anda dapat menggunakan beberapa teknik untuk mengimpor data cadangan fisik atau logis ke Aurora. Untuk detail, lihat Memigrasikan data ke klaster DB Amazon Aurora MySQL atau Memigrasikan data ke Amazon Aurora dengan kompatibilitas PostgreSQL sesuai dengan mesin basis data yang Anda gunakan dalam bukti konsep.

Bereksperimenlah dengan alat dan teknologi ETL yang Anda pertimbangkan. Lihat mana yang paling sesuai dengan kebutuhan Anda. Pertimbangkan throughput dan fleksibilitasnya. Misalnya, beberapa alat ETL dapat melakukan transfer satu kali, dan alat yang lainnya mendukung replikasi berkelanjutan dari sistem lama ke Aurora.

Jika Anda bermigrasi dari sistem yang kompatibel dengan MySQL ke Aurora MySQL, maka Anda dapat menggunakan alat transfer data native. Hal yang sama berlaku jika Anda bermigrasi dari sistem yang kompatibel dengan PostgreSQL ke Aurora PostgreSQL. Jika Anda bermigrasi dari sistem basis data yang menggunakan mesin dasar yang berbeda dari yang digunakan Aurora, Anda

dapat bereksperimen dengan AWS Database Migration Service (AWS DMS). Untuk detail tentang AWS DMS, lihat Panduan Pengguna AWS Database Migration Service.

Untuk detail tentang migrasi dan aktivitas porting, lihat laporan resmi AWS Buku panduan migrasi Aurora.

7. Lakukan porting kode SQL Anda

Mencoba SQL dan aplikasi terkait membutuhkan tingkat upaya yang berbeda-beda, tergantung jenis kasusnya. Secara khusus, tingkat upaya akan bergantung pada apakah Anda berpindah dari sistem yang kompatibel dengan MySQL, kompatibel dengan PostgreSQL, atau jenis lainnya.

- Jika Anda beralih dari RDS for MySQL atau RDS for PostgreSQL, perubahan SQL cukup kecil sehingga Anda dapat mencoba kode SQL asli dengan Aurora dan memasukkan secara manual perubahan yang diperlukan.
- Demikian pula, jika Anda berpindah dari basis data on-premise yang kompatibel dengan MySQL atau PostgreSQL, Anda dapat mencoba kode SQL asli dan memasukkan perubahan secara manual.
- Jika Anda awalnya menggunakan basis data komersial yang berbeda, maka perubahan SQL yang diperlukan akan bersifat lebih ekstensif. Jika demikian, pertimbangkan untuk menggunakan AWS SCT.

Selama tahap ini, Anda dapat mengevaluasi apakah terdapat inefisiensi dalam penyiapan skema Anda, misalnya dalam strategi pengindeksan atau struktur tabel lain seperti tabel yang dipartisi. Pertimbangkan apakah Anda dapat menyempurnakan aspek performa tersebut sekarang, atau selama aktivitas selanjutnya seperti pengujian tolok ukur lengkap.

Anda dapat memverifikasi logika koneksi basis data di aplikasi Anda. Untuk memanfaatkan pemrosesan terdistribusi Aurora, Anda mungkin perlu menggunakan koneksi terpisah untuk operasi baca dan tulis, dan menggunakan sesi yang relatif singkat untuk operasi kueri. Untuk informasi tentang koneksi, lihat 9. Hubungkan ke Aurora.

Pertimbangkan apakah Anda harus membuat kompromi dan pengorbanan untuk mengatasi masalah dalam basis data produksi Anda. Bangun waktu ke dalam proof-of-concept jadwal untuk melakukan perbaikan pada desain dan kueri skema Anda. Untuk menilai apakah Anda dapat meraih keuntungan dari segi performa, biaya pengoperasian, dan skalabilitas yang baik, coba aplikasi asli dan yang dimodifikasi secara berdampingan di klaster Aurora yang berbeda.

Untuk detail tentang migrasi dan aktivitas porting, lihat laporan resmi AWS Buku panduan migrasi Aurora.

8. Tentukan pengaturan konfigurasi

Anda juga dapat meninjau parameter konfigurasi database Anda sebagai bagian dari latihan Aurora proof-of-concept . Anda mungkin sudah mengatur konfigurasi MySQL atau PostgreSQL untuk performa dan skalabilitas di lingkungan Anda saat ini. Subsistem penyimpanan Aurora diubah dan diatur untuk lingkungan berbasis cloud terdistribusi dengan subsistem penyimpanan berkecepatan tinggi. Akibatnya, banyak pengaturan mesin basis data sebelumnya tidak berlaku. Kami merekomendasikan untuk melakukan eksperimen awal Anda dengan pengaturan konfigurasi Aurora default. Terapkan kembali pengaturan dari lingkungan Anda saat ini hanya jika Anda mengalami hambatan performa dan skalabilitas. Jika tertarik, Anda dapat mempelajari lebih jauh tentang subjek ini dalam Memperkenalkan mesin penyimpanan Aurora di Blog Basis Data AWS.

Aurora memudahkan penggunaan kembali pengaturan konfigurasi optimal untuk aplikasi atau kasus penggunaan tertentu. Alih-alih mengedit file konfigurasi terpisah untuk setiap instans DB, Anda mengelola sekumpulan parameter yang Anda tetapkan ke seluruh klaster atau instans DB tertentu. Misalnya, pengaturan zona waktu berlaku untuk semua instans DB di klaster, dan Anda dapat menyesuaikan pengaturan ukuran cache halaman untuk setiap instans DB.

Anda memulai dengan salah satu set parameter default, dan menerapkan perubahan ke parameter yang perlu Anda sesuaikan saja. Untuk detail tentang menggunakan grup parameter, lihat Parameter instans DB dan klaster DB Amazon Aurora. Untuk pengaturan konfigurasi yang berlaku atau tidak berlaku untuk klaster Aurora, lihat Parameter konfigurasi Aurora MySQL atau Parameter Amazon Aurora PostgreSQL tergantung mesin basis data Anda.

9. Hubungkan ke Aurora

Seperti yang Anda temukan saat membuat skema awal dan pengaturan data serta menjalankan kueri sampel, Anda dapat terhubung ke titik akhir yang berbeda-beda dalam klaster Aurora. Titik akhir yang akan digunakan bergantung pada apakah operasinya adalah pembacaan, seperti pernyataan `SELECT` atau penulisan, seperti pernyataan `CREATE` atau `INSERT`. Saat Anda meningkatkan beban kerja pada klaster Aurora dan bereksperimen dengan fitur Aurora, penting bagi aplikasi Anda untuk menetapkan setiap operasi ke titik akhir yang sesuai.

Dengan menggunakan titik akhir klaster untuk operasi penulisan, Anda selalu terhubung ke instans DB di klaster yang memiliki kemampuan baca/tulis. Secara default, hanya satu instans DB dalam

klaster Aurora yang memiliki kemampuan baca/tulis. Instans DB ini disebut instans primer. Jika instans primer asli tidak tersedia, Aurora mengaktifkan mekanisme failover dan instans DB yang berbeda akan mengambil alih sebagai instans primer.

Demikian pula, dengan mengarahkan pernyataan SELECT ke titik akhir pembaca, Anda menyebarkan pemrosesan kueri di antara instans DB dalam klaster. Setiap koneksi pembaca ditetapkan ke instans DB yang berbeda menggunakan resolusi DNS round-robin. Melakukan sebagian besar pekerjaan kueri pada Replika Aurora DB hanya baca akan mengurangi beban pada instans primer, sehingga instans tersebut dapat menangani pernyataan DDL dan DML.

Menggunakan titik akhir ini akan mengurangi ketergantungan pada nama host yang di-hard-coding, dan membantu aplikasi Anda dipulihkan lebih cepat jika ada kegagalan instans DB.

> **Note**
>
> Aurora juga memiliki titik akhir kustom yang Anda buat. Titik akhir tersebut biasanya tidak diperlukan selama proses pembuktian konsep.

Replika Aurora dapat mengalami lag replika, meskipun lag ini biasanya berlangsung 10 hingga 20 milidetik. Anda bisa memantau lag replikasi dan memutuskan apakah lag ini masih dalam kisaran persyaratan konsistensi data Anda. Dalam beberapa kasus, kueri baca Anda mungkin memerlukan konsistensi baca yang kuat (read-after-writekonsistensi). Jika demikian, Anda dapat terus menggunakan titik akhir klaster dan bukan titik akhir pembaca.

Untuk memanfaatkan kemampuan Aurora secara utuh untuk eksekusi paralel terdistribusi, Anda mungkin perlu mengubah logika koneksinya. Tujuan Anda adalah menghindari pengiriman semua permintaan baca ke instans primer. Replika Aurora hanya baca akan bersiaga, dengan semua data yang sama, untuk menangani pernyataan SELECT. Kodekan logika aplikasi Anda untuk menggunakan titik akhir yang sesuai untuk setiap jenis operasi. Ikuti pedoman umum berikut:

- Hindari menggunakan satu string koneksi hard-code untuk semua sesi basis data.
- Jika memungkinkan, sertakan operasi tulis seperti pernyataan DDL dan DML dalam fungsi di kode aplikasi klien Anda. Dengan demikian, Anda dapat membuat berbagai jenis operasi menggunakan koneksi tertentu.
- Buat fungsi terpisah untuk operasi kueri. Aurora menetapkan setiap koneksi baru dengan titik akhir pembaca ke Replika Aurora yang berbeda untuk menyeimbangkan beban aplikasi sarat baca.

- Untuk operasi yang menangani sekumpulan kueri, tutup dan buka kembali koneksi ke titik akhir pembaca saat setiap kumpulan kueri terkait selesai. Gunakan pooling koneksi jika fiturnya tersedia di tumpukan perangkat lunak Anda. Mengarahkan kueri ke koneksi yang berbeda akan membantu Aurora mendistribusikan beban kerja baca di beberapa instans DB di klaster.

Untuk informasi umum tentang manajemen koneksi dan titik akhir untuk Aurora, lihat Menghubungkan ke klaster DB Amazon Aurora. Untuk mempelajari lebih dalam tentang bahasan ini, lihat Buku panduan administrator basis data Aurora MySQL – Manajemen koneksi.

10. Jalankan beban kerja Anda

Setelah pengaturan skema, data, dan konfigurasi dilakukan, Anda dapat mulai mencoba klaster dengan menjalankan beban kerja Anda. Gunakan beban kerja sebagai bukti konsep yang mencerminkan aspek utama beban kerja produksi Anda. Kami menyarankan Anda untuk selalu membuat keputusan seputar performa menggunakan pengujian dan beban kerja dunia nyata daripada penilaian tolok ukur sintetis seperti sysbench atau TPC-C. Jika memungkinkan, kumpulkan pengukuran berdasarkan skema, pola kueri, dan volume penggunaan Anda sendiri.

Sebisa mungkin, tiru kondisi sebenarnya yang akan dialami aplikasi yang berjalan nantinya. Misalnya, Anda biasanya menjalankan kode aplikasi Anda pada instans Amazon EC2 di Wilayah AWS yang sama dan cloud privat virtual (VPC) yang sama dengan klaster Aurora. Jika aplikasi produksi Anda berjalan pada beberapa instans EC2 yang mencakup beberapa Availability Zone, atur proof-of-concept lingkungan Anda dengan cara yang sama. Untuk informasi selengkapnya tentang Wilayah AWS, lihat Wilayah dan Zona Ketersediaan dalam Panduan Pengguna Amazon RDS. Untuk mempelajari selengkapnya tentang layanan Amazon VPC, lihat Apa Itu Amazon VPC? dalam Panduan Pengguna Amazon VPC.

Setelah Anda memastikan bahwa fitur dasar aplikasi Anda berfungsi dan Anda dapat mengakses data melalui Aurora, Anda akan dapat mencoba berbagai aspek klaster Aurora. Beberapa fitur yang mungkin ingin Anda coba adalah koneksi konkuren dengan penyeimbangan beban, transaksi konkuren, dan replikasi otomatis.

Pada tahap ini, mekanisme transfer datanya seharusnya sudah dikenal, sehingga Anda dapat menjalankan pengujian dengan proporsi data sampel yang lebih besar.

Di tahap inilah Anda dapat melihat efek dari perubahan pengaturan konfigurasi seperti batas memori dan batas koneksi. Tinjau kembali prosedur yang Anda kaji dalam 8. Tentukan pengaturan konfigurasi.

Anda juga dapat bereksperimen dengan mekanisme seperti membuat dan memulihkan snapshot. Misalnya, Anda dapat membuat klaster dengan kelas instans AWS yang berbeda, jumlah Replika AWS yang berbeda, dan lain sebagainya. Kemudian, di setiap klaster, Anda dapat memulihkan snapshot yang sama yang berisi skema dan semua data Anda. Untuk detail tentang siklus tersebut, lihat Membuat snapshot klaster DB dan Memulihkan dari snapshot klaster DB.

11. Ukur performa

Praktik terbaik dalam aspek ini dirancang untuk memastikan bahwa semua alat dan proses disiapkan secara tepat untuk dapat mengisolasi perilaku abnormal selama operasi beban kerja. Praktik terbaik ini juga disiapkan untuk memastikan bahwa Anda dapat mengidentifikasi kemungkinan penyebab masalah yang berlaku.

Anda selalu dapat melihat status klaster Anda saat ini, atau memeriksa tren dari waktu ke waktu, dengan memeriksa tab Pemantauan. Tab ini tersedia dari halaman detail konsol untuk setiap klaster atau instans DB Aurora. Ini menampilkan metrik dari layanan CloudWatch pemantauan Amazon dalam bentuk bagan. Anda dapat memfilter metrik berdasarkan nama, instans DB, dan periode waktu.

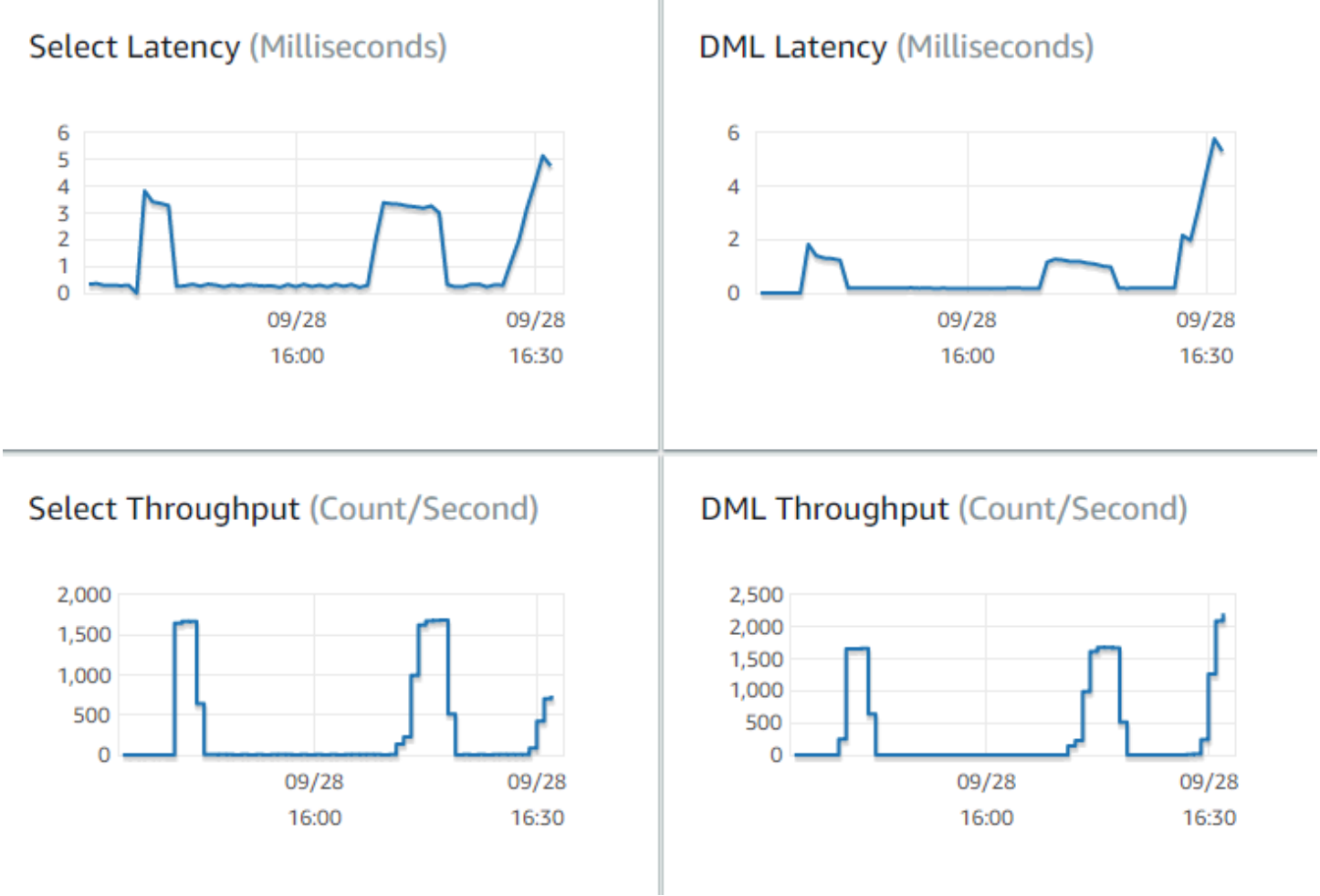
Untuk memiliki lebih banyak pilihan di tab Pemantauan, aktifkan Pemantauan yang Ditingkatkan dan Wawasan Performa di pengaturan klaster. Anda juga dapat mengaktifkan pilihan tersebut nanti jika Anda tidak memilihnya saat menyiapkan klaster.

Untuk mengukur performa, Anda sebagian besar akan mengandalkan bagan yang menampilkan aktivitas untuk seluruh klaster Aurora. Anda dapat memastikan apakah Replika Aurora memiliki waktu pemuatan dan waktu respons yang sama. Anda juga dapat melihat pemisahan pekerjaan antara instans primer baca/tulis dan Replika Aurora hanya baca. Jika ada ketidakseimbangan antara instans DB atau masalah yang hanya memengaruhi satu instans DB, Anda dapat memeriksa tab Pemantauan untuk instans yang bermasalah tersebut.

Setelah lingkungan dan beban kerja sebenarnya disiapkan untuk meniru aplikasi produksi Anda, Anda dapat mengukur seberapa baik performa Aurora. Berikut adalah pertanyaan terpenting yang perlu dijawab:

- Berapa banyak kueri per detik yang diproses Aurora? Anda dapat memeriksa metrik Throughput untuk melihat angka untuk berbagai jenis operasi yang ada.
- Berapa lama rata-rata waktu yang diperlukan Aurora untuk memproses kueri tertentu? Anda dapat memeriksa metrik Latensi untuk melihat angka untuk berbagai jenis operasi yang ada.

Untuk melakukannya, lihat tab Pemantauan untuk klaster Aurora tertentu di konsol Amazon RDS seperti yang diilustrasikan berikut ini.



Jika memungkinkan, tetapkan nilai acuan dasar untuk metrik ini di lingkungan Anda saat ini. Jika tidak memungkinkan, maka buat acuan dasar di klaster Aurora dengan menjalankan beban kerja yang setara dengan aplikasi produksi Anda. Misalnya, jalankan beban kerja Aurora Anda dengan jumlah pengguna dan kueri simultan yang sama. Kemudian, amati perubahan nilainya saat Anda bereksperimen dengan berbagai kelas instans, ukuran klaster, pengaturan konfigurasi, dan sebagainya.

Jika jumlah throughput lebih rendah dari yang Anda harapkan, periksa lebih lanjut faktor-faktor yang memengaruhi performa basis data untuk beban kerja Anda. Selain itu, lakukan pemeriksaan lebih lanjut jika angka latensi lebih tinggi dari yang Anda harapkan. Untuk melakukannya, pantau metrik sekunder untuk server DB (CPU, memori, dan sebagainya). Anda dapat melihat apakah instans DB mulai mendekati batasnya. Anda juga dapat melihat berapa banyak kapasitas ekstra yang dimiliki

instans DB Anda untuk menangani lebih banyak kueri konkuren, kueri terhadap tabel yang lebih besar, dan sebagainya.

> **Tip**
>
> Untuk mendeteksi nilai metrik yang berada di luar rentang yang diharapkan, atur CloudWatch alarm.

Saat mengevaluasi ukuran dan kapasitas klaster Aurora yang ideal, Anda akan dapat menemukan konfigurasi yang mendekati performa aplikasi maksimal tanpa penyediaan sumber daya yang berlebihan. Salah satu faktor penting adalah menemukan ukuran yang sesuai untuk instans DB di klaster Aurora. Mulailah dengan memilih ukuran instans yang memiliki CPU dan kapasitas memori yang serupa dengan lingkungan produksi Anda saat ini. Kumpulkan angka throughput dan latensi untuk beban kerja pada ukuran instans tersebut. Kemudian, naikkan skala instans ini ke ukuran yang lebih besar berikutnya. Lihat apakah angka throughput dan latensi menjadi bertambah baik. Selain itu, turunkan ukuran instans, dan lihat apakah angka latensi dan throughput tetap sama. Sasaran Anda adalah mendapatkan throughput tertinggi, dengan latensi terendah, pada instans sekecil mungkin.

> **Tip**
>
> Tentukan ukuran klaster Aurora Anda dan instans DB terkait dengan kapasitas yang cukup untuk menangani lonjakan lalu lintas yang tiba-tiba dan tidak dapat diprediksi. Untuk basis data yang bersifat vital, sisakan setidaknya 20 persen kapasitas CPU dan memori cadangan.

Jalankan uji performa selama mungkin untuk mengukur performa basis data dalam kondisi siap dan stabil. Anda mungkin perlu menjalankan beban kerja selama beberapa menit atau bahkan beberapa jam sebelum mencapai kondisi stabil ini. Varians yang bermunculan di awal proses menjalankan beban kerja adalah hal yang normal. Varians tersebut terjadi karena setiap Replika Aurora menyiapkan cache-nya berdasarkan kueri `SELECT` yang ditanganinya.

Aurora beperforma paling baik untuk beban kerja transaksional yang menangani beberapa pengguna dan kueri secara bersamaan. Untuk memastikan bahwa Anda memiliki beban yang cukup untuk performa yang optimal, jalankan penilaian tolok ukur yang menggunakan multithreading, atau jalankan beberapa uji performa secara bersamaan. Ukur performa dengan ratusan atau bahkan ribuan thread konkuren klien. Simulasikan jumlah thread konkuren yang Anda perkirakan di dalam

lingkungan produksi Anda. Anda juga dapat melakukan uji stres tambahan dengan lebih banyak thread untuk mengukur skalabilitas Aurora.

12. Coba ketersediaan tinggi Aurora

Sebagian besar fitur utama Aurora mendukung ketersediaan yang tinggi. Fitur-fitur ini termasuk replikasi otomatis, failover otomatis, backup otomatis dengan point-in-time restore, dan kemampuan untuk menambahkan instans DB ke cluster. Keamanan dan keandalan dari fitur-fitur seperti ini penting untuk aplikasi yang bersifat vital.

Pola pikir tertentu diperlukan untuk mengevaluasi fitur tersebut. Dalam aktivitas sebelumnya, seperti pengukuran performa, Anda mengamati bagaimana sistem berfungsi ketika semuanya dapat bekerja dengan benar. Menguji ketersediaan tinggi mengharuskan Anda mempertimbangkan perilaku kasus terburuk. Anda harus mempertimbangkan potensi berbagai macam kegagalan, sekalipun kondisi kegagalan tersebut jarang terjadi. Anda dapat sengaja menimbulkan masalah untuk memastikan bahwa sistem dipulihkan dengan benar dan cepat.

> **Tip**
>
> Untuk bukti konsep, atur semua instans DB dalam klaster Aurora dengan kelas instans AWS yang sama. Dengan melakukannya, Anda akan dapat mencoba fitur ketersediaan Aurora tanpa perubahan besar pada performa dan skalabilitas saat Anda membuat instans DB offline untuk mensimulasikan kegagalan.

Kami merekomendasikan penggunaan setidaknya dua instans di setiap klaster Aurora. Instans DB dalam klaster Aurora dapat mencakup hingga tiga Zona Ketersediaan (AZ). Tempatkan dua atau tiga instans DB pertama di AZ yang berbeda. Saat mulai menggunakan klaster yang lebih besar, sebarkan instans DB Anda ke semua AZ di Wilayah AWS Anda. Dengan melakukannya, kemampuan toleransi kesalahan akan meningkat. Bahkan jika ada masalah yang memengaruhi seluruh AZ, Aurora dapat beralih ke instans DB di AZ yang berbeda. Jika Anda menjalankan klaster dengan lebih dari tiga instans, distribusikan instans DB semerata mungkin di tiga AZ.

> **Tip**
>
> Penyimpanan untuk klaster Aurora tidak tergantung pada instans DB. Penyimpanan untuk setiap klaster Aurora selalu mencakup tiga AZ.

> Saat Anda menguji fitur ketersediaan tinggi, selalu gunakan instans DB dengan kapasitas yang sama di klaster pengujian Anda. Hal tersebut akan menghindari perubahan yang tidak dapat diprediksi dalam performa, latensi, dan sebagainya setiap kali ada satu instans DB yang mengambil alih instans DB lain.

Untuk mempelajari cara mensimulasikan kondisi kegagalan untuk menguji fitur ketersediaan tinggi, lihat Menguji Amazon Aurora MySQL menggunakan kueri injeksi kesalahan.

Sebagai bagian dari proof-of-concept latihan Anda, salah satu tujuannya adalah untuk menemukan jumlah instans DB yang ideal dan kelas instans optimal untuk instans DB tersebut. Untuk melakukannya, diperlukan keseimbangan antara ketersediaan tinggi dan performa.

Untuk Aurora, semakin banyak instans DB yang Anda miliki dalam sebuah klaster, semakin besar manfaat untuk ketersediaan tinggi. Memiliki lebih banyak instans DB juga meningkatkan skalabilitas aplikasi sarat baca. Aurora dapat mendistribusikan beberapa koneksi untuk kueri `SELECT` di beberapa Replika Aurora hanya baca.

Di sisi lain, membatasi jumlah instans DB akan mengurangi lalu lintas replikasi dari simpul primer. Lalu lintas replikasi akan menggunakan bandwidth jaringan, yang merupakan aspek lain dari performa dan skalabilitas secara keseluruhan. Jadi, untuk aplikasi OLTP sarat tulis, sebaiknya Anda memiliki sedikit instans DB besar dibandingkan dengan banyak instans DB kecil.

Dalam klaster Aurora umum, satu instans DB (instans primer) menangani semua pernyataan DDL dan DML. Instans DB lainnya (Replika Aurora) hanya menangani pernyataan `SELECT`. Meskipun instans DB tidak melakukan jumlah pekerjaan yang persis sama, kami merekomendasikan penggunaan kelas instans yang sama untuk semua instans DB dalam klaster. Dengan begitu, jika terjadi kegagalan dan Aurora menjadikan salah satu instans DB hanya baca sebagai instans primer yang baru, instans primer ini akan memiliki kapasitas yang sama seperti sebelumnya.

Jika Anda perlu menggunakan instans DB dengan kapasitas berbeda dalam klaster yang sama, maka atur tingkat failover untuk instans DB. Tingkatan ini menentukan urutan promosi Replika Aurora oleh mekanisme failover. Tempatkan instans DB yang jauh lebih besar atau lebih kecil dari yang lain ke tingkat failover yang lebih rendah. Dengan melakukannya, Anda akan memastikan bahwa instans tersebut akan dipilih terakhir untuk promosi.

Latih fitur pemulihan data Aurora, seperti pemulihan otomatis, snapshot dan point-in-time pemulihan manual, dan backtracking cluster. Jika memungkinkan, salin snapshot ke Wilayah AWS lain dan pulihkan ke Wilayah AWS lain untuk meniru skenario DR.

Selidiki persyaratan organisasi Anda untuk sasaran waktu pemulihan (RTO), sasaran titik pemulihan (RPO), dan redundansi geografis. Sebagian besar organisasi mengelompokkan item ini ke dalam kategori pemulihan bencana. Evaluasi fitur ketersediaan tinggi Aurora yang dijelaskan di bagian ini dalam konteks proses pemulihan bencana Anda untuk memastikan bahwa persyaratan RTO dan RPO Anda terpenuhi.

13. Apa yang harus dilakukan selanjutnya

Di akhir proof-of-concept proses yang sukses, Anda mengonfirmasi bahwa Aurora adalah solusi yang cocok untuk Anda berdasarkan beban kerja yang diantisipasi. Sepanjang proses sebelumnya, Anda telah memeriksa cara kerja Aurora dalam lingkungan operasional yang realistis dan mengukurnya dengan kriteria kesuksesan Anda.

Setelah lingkungan basis data Anda aktif dan berfungsi dengan Aurora, Anda dapat melanjutkan ke langkah evaluasi yang lebih mendetail, yang mengarah ke migrasi akhir dan deployment produksi. Tergantung pada situasi Anda, langkah-langkah lain ini mungkin atau mungkin tidak termasuk dalam proof-of-concept proses. Untuk detail tentang migrasi dan aktivitas porting, lihat laporan resmi AWS Buku panduan migrasi Aurora.

Di langkah berikutnya, pertimbangkan konfigurasi keamanan yang relevan dengan beban kerja Anda dan yang dirancang untuk memenuhi persyaratan keamanan Anda dalam lingkungan produksi. Rencanakan kontrol apa yang akan diterapkan untuk melindungi akses ke kredensial pengguna master klaster Aurora. Tentukan peran dan tanggung jawab pengguna basis data untuk mengontrol akses ke data yang disimpan di klaster Aurora. Pertimbangkan persyaratan akses basis data untuk aplikasi, skrip, dan alat atau layanan pihak ketiga. Pelajari layanan dan fitur AWS seperti autentikasi AWS Secrets Manager dan AWS Identity and Access Management (IAM).

Pada tahap ini, Anda akan memahami prosedur dan praktik terbaik untuk menjalankan pengujian penilaian tolok ukur dengan Aurora. Anda mungkin perlu melakukan penyesuaian performa tambahan. Untuk detailnya, lihat Mengelola performa dan penskalaan untuk klaster DB Aurora, Peningkatan performa Amazon Aurora MySQL, Mengelola Amazon Aurora PostgreSQL, dan Memantau muatan DB dengan Wawasan Performa di Amazon Aurora. Jika Anda melakukan penyesuaian tambahan, pastikan Anda memahami metrik yang Anda kumpulkan selama pembuktian konsep. Untuk langkah berikutnya, Anda mungkin perlu membuat klaster baru dengan pilihan berbeda untuk pengaturan konfigurasi, mesin basis data, dan versi basis data. Anda juga dapat membuat jenis klaster Aurora khusus untuk menyesuaikan dengan kebutuhan kasus penggunaan tertentu.

Misalnya, Anda dapat mengkaji klaster kueri paralel Aurora untuk aplikasi pemrosesan transaksi/analitik hibrid (Hybrid Transaction/Analytical Processing atau HTAP). Jika distribusi geografis yang luas sangat penting untuk pemulihan bencana atau untuk meminimalkan latensi, Anda dapat mengkaji basis data global Aurora. Jika beban kerja Anda bersifat intermiten atau Anda menggunakan Aurora dalam skenario pengembangan/pengujian, Anda dapat mengkaji klaster Aurora Serverless.

Klaster produksi Anda mungkin juga perlu menangani koneksi masuk dalam jumlah yang besar. Untuk mempelajari teknik tersebut, lihat laporan resmi AWS Buku panduan administrator basis data Aurora MySQL – Manajemen koneksi.

Jika, setelah pembuktian konsep, Anda memutuskan bahwa kasus penggunaan Anda tidak sesuai untuk Aurora, pertimbangkan layanan AWS lainnya berikut ini:

- Untuk kasus penggunaan analitik murni, beban kerja akan mendapatkan keuntungan dari format penyimpanan kolom dan fitur lain yang lebih cocok untuk beban kerja OLAP. Layanan AWS yang menangani kasus penggunaan tersebut termasuk yang berikut ini:
  - Amazon Redshift
  - Amazon EMR
  - Amazon Athena
- Sebagian besar beban kerja akan mendapatkan keuntungan dari kombinasi Aurora dengan satu atau beberapa layanan ini. Anda dapat memindahkan data di antara layanan-layanan ini dengan menggunakan hal berikut ini:
  - AWS Glue
  - AWS DMS
  - Mengimpor dari Amazon S3, sebagaimana dijelaskan di Panduan Pengguna Amazon Aurora
  - Mengekspor ke Amazon S3, sebagaimana dijelaskan di Panduan Pengguna Amazon Aurora
  - Banyak alat ETL populer lainnya

Keamanan dalam Amazon Aurora

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai pelanggan AWS, Anda mendapatkan manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan dari organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara AWS dan Anda. Model tanggung jawab bersama menjelaskan hal ini sebagai keamanan dari cloud dan keamanan di cloud:

- Keamanan cloud – AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan layanan AWS di Cloud AWS. AWS juga memberikan Anda layanan yang dapat digunakan dengan aman. Auditor pihak ketiga secara berkala menguji dan memverifikasi efektivitas keamanan kami sebagai bagian dari AWS program kepatuhan. Untuk mempelajari program kepatuhan yang berlaku pada Amazon Aurora (Aurora), lihat layanan AWS dalam cakupan menurut program kepatuhan.
- Keamanan di cloud – Tanggung jawab Anda ditentukan oleh AWS yang Anda gunakan. Anda juga bertanggung jawab atas faktor lain, termasuk sensitivitas data, persyaratan perusahaan, serta hukum dan peraturan yang berlaku.

Dokumentasi ini akan membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan Amazon Aurora. Topik berikut menunjukkan kepada Anda cara mengonfigurasi Amazon Aurora untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga akan mempelajari cara menggunakan layanan AWS lain yang membantu Anda memantau dan mengamankan sumber daya Amazon Aurora.

Anda dapat mengelola akses ke sumber daya Amazon Aurora dan basis data Anda di klaster DB. Metode yang Anda gunakan untuk mengelola akses ditentukan oleh jenis tugas yang harus dilakukan oleh pengguna dengan Amazon Aurora:

- Jalankan instans DB Anda dalam cloud privat virtual (VPC) berdasarkan layanan Amazon VPC untuk kontrol akses jaringan tertinggi. Untuk informasi selengkapnya tentang cara membuat instans DB di VPC, lihat Amazon VPC dan Amazon Aurora.
- Gunakan kebijakan (IAM) AWS Identity and Access Management untuk menetapkan izin yang menentukan siapa yang diizinkan untuk mengelola sumber daya Amazon Aurora. Misalnya, Anda dapat menggunakan IAM untuk menentukan siapa yang diizinkan untuk membuat, menjelaskan, memodifikasi, dan menghapus instans DB, memberi tag pada sumber daya, atau memodifikasi grup keamanan.

Untuk meninjau contoh kebijakan IAM, lihat Contoh kebijakan berbasis identitas untuk Amazon Aurora.

- Gunakan grup keamanan untuk mengontrol alamat IP atau instans Amazon EC2 yang dapat terhubung ke basis data Anda di instans DB. Saat Anda pertama kali membuat instans DB, firewall-nya mencegah semua akses basis data kecuali melalui aturan yang ditentukan oleh grup keamanan terkait.
- Gunakan koneksi Lapisan Soket Keamanan (SSL) atau Keamanan Lapisan Pengangkutan (TLS) dengan instans DB yang menjalankan Aurora MySQL atau Aurora PostgreSQL. Untuk informasi selengkapnya tentang cara menggunakan SSL/TLS dengan klaster DB, lihat .
- Gunakan enkripsi Amazon Aurora untuk mengamankan klaster DB dan snapshot yang tersimpan. Enkripsi Amazon Aurora menggunakan algoritma enkripsi AES-256 standar industri untuk mengenkripsi data Anda di server yang meng-host instans DB Anda. Untuk informasi selengkapnya, lihat Mengenkripsi sumber daya Amazon Aurora.
- Gunakan fitur keamanan pada mesin DB Anda untuk mengontrol siapa yang dapat login ke basis data di instans DB. Fitur ini berfungsi seolah-olah basis data berada di jaringan lokal Anda.

  Untuk informasi selengkapnya terkait keamanan dengan Aurora MySQL, lihat Keamanan dengan Amazon Aurora MySQL. Untuk informasi selengkapnya terkait keamanan dengan Aurora PostgreSQL, lihat Keamanan dengan Amazon Aurora PostgreSQL.

Aurora adalah bagian dari layanan basis data terkelola Amazon Relational Database Service (Amazon RDS). Amazon RDS adalah layanan web yang memudahkan penyiapan, pengoperasian, dan penskalaan basis data relasional di cloud. Jika Anda belum memahami tentang Amazon RDS, lihat Panduan pengguna Amazon RDS.

Aurora mencakup subsistem penyimpanan performa tinggi. Mesin basis data-nya yang kompatibel dengan MySQL dan PostgreSQL disesuaikan untuk memanfaatkan penyimpanan terdistribusi cepat tersebut. Aurora juga mengotomatiskan dan membakukan pengelompokan dan replikasi basis data, yang biasanya termasuk dalam aspek konfigurasi dan administrasi basis data yang paling menantang.

Untuk Amazon RDS dan Aurora, Anda dapat mengakses API RDS secara terprogram, dan Anda dapat menggunakan AWS CLI untuk mengakses API RDS secara interaktif. Beberapa pengoperasian API RDS dan perintah AWS CLI berlaku untuk Amazon RDS dan Aurora, sementara yang lainnya berlaku untuk Amazon RDS atau Aurora. Untuk informasi selengkapnya tentang pengoperasian API

RDS, lihat Referensi API Amazon RDS. Untuk informasi lebih lanjut tentang AWS CLI, lihat Referensi untuk Amazon RDS AWS Command Line Interface.

> **Note**
>
> Anda hanya perlu mengonfigurasikan keamanan untuk kasus penggunaan Anda. Anda tidak perlu mengonfigurasi akses keamanan untuk proses yang dikelola Amazon Aurora. Ini termasuk membuat cadangan, failover otomatis, dan proses lainnya.

Untuk informasi selengkapnya tentang cara mengelola akses ke sumber daya Amazon Aurora dan basis data Anda di instans DB, lihat topik berikut.

Topik

- Autentikasi basis data dengan Amazon Aurora
- Manajemen kata sandi dengan dan AWS Secrets Manager
- Perlindungan data di Amazon RDS
- Manajemen identitas dan akses untuk Amazon Aurora
- Pencatatan dan pemantauan di Amazon Aurora
- Validasi kepatuhan untuk Amazon Aurora
- Ketangguhan di Amazon Aurora
- Keamanan infrastruktur dalam Amazon Aurora
- API Amazon RDS dan titik akhir VPC antarmuka (AWS PrivateLink)
- Praktik terbaik keamanan untuk Amazon Aurora
- Mengontrol akses dengan grup keamanan
- Hak akses akun pengguna master
- Menggunakan peran tertaut layanan untuk Amazon Aurora
- Amazon VPC dan Amazon Aurora

Autentikasi basis data dengan Amazon Aurora

Amazon Aurora mendukung beberapa cara untuk mengautentikasi pengguna basis data.

Autentikasi kata sandi tersedia secara bawaan untuk semua klaster basis data. Untuk Aurora MySQL dan Aurora PostgreSQL, Anda juga dapat menambahkan salah satu dari autentikasi basis data IAM dan autentikasi Kerberos atau kedua-duanya bagi klaster basis data yang sama.

Autentikasi kata sandi, Kerberos, dan basis data IAM menggunakan metode autentikasi yang berbeda ke basis data. Oleh karena itu, pengguna tertentu dapat masuk ke basis data dengan menggunakan hanya satu metode autentikasi.

Untuk PostgreSQL, gunakan hanya salah satu dari setelah peran berikut untuk pengguna basis data tertentu:

- Untuk menggunakan autentikasi basis data IAM, tetapkan peran `rds_iam` untuk pengguna.
- Untuk menggunakan autentikasi Kerberos, tetapkan peran `rds_ad` untuk pengguna.
- Untuk menggunakan autentikasi kata sandi, jangan tetapkan peran `rds_iam` atau `rds_ad` untuk pengguna.

Jangan tetapkan peran `rds_iam` dan `rds_ad` sekaligus untuk pengguna basis data PostgreSQL baik secara langsung maupun tidak langsung dengan akses pemberian bersarang. Jika peran `rds_iam` ditambahkan ke pengguna master, autentikasi IAM diutamakan atas autentikasi kata sandi sehingga pengguna master harus masuk sebagai pengguna IAM.

> **Important**
>
> Kami sangat menyarankan agar Anda tidak menggunakan pengguna master secara langsung di aplikasi Anda. Sebagai gantinya, ikuti praktik terbaik menggunakan pengguna basis data yang dibuat dengan hak akses minimal yang diperlukan untuk aplikasi Anda.

Topik

- Autentikasi kata sandi
- Autentikasi basis data IAM
- Autentikasi Kerberos

Autentikasi kata sandi

Dengan autentikasi kata sandi, basis data Anda melakukan semua administrasi akun pengguna. Anda membuat pengguna dengan pernyataan SQL seperti `CREATE USER`, dengan klausa yang tepat

yang diperlukan oleh mesin basis data untuk menentukan kata sandi. Misalnya, pernyataannya di MySQL adalah `CREATE USER nama IDENTIFIED BY kata sandi`, sedangkan pernyataannya di PostgreSQL adalah `CREATE USER nama WITH PASSWORD kata sandi`.

Dengan autentikasi kata sandi, basis data Anda mengendalikan dan mengautentikasi akun pengguna. Jika mesin basis data memiliki fitur pengelolaan kata sandi kuat, mesin itu dapat meningkatkan keamanan. Autentikasi basis data mungkin lebih mudah dikelola dengan menggunakan autentikasi kata sandi apabila komunitas pengguna Anda kecil. Karena kata sandi teks jelas dihasilkan dalam kasus ini, integrasi dengan AWS Secrets Manager dapat meningkatkan keamanan.

Lihat informasi tentang cara menggunakan Secrets Manager dengan Amazon Aurora di Membuat rahasia dasar dan Merotasi rahasia untuk basis data Amazon RDS yang didukung dalam Panduan Pengguna AWS Secrets Manager. Lihat informasi tentang cara mengambil rahasia secara terprogram pada aplikasi kustom Anda di Mengambil nilai rahasia dalam Panduan Pengguna AWS Secrets Manager.

Autentikasi basis data IAM

Anda dapat mengautentikasi ke instans basis data menggunakan autentikasi basis data AWS Identity and Access Management (IAM). Autentikasi basis data IAM bekerja dengan Aurora MySQL dan Aurora PostgreSQL. Dengan metode autentikasi ini, Anda tidak perlu menggunakan kata sandi saat menghubungkan dengan klaster basis data. Sebagai gantinya, gunakan token autentikasi.

Lihat informasi selengkapnya tentang autentikasi basis data IAM, termasuk informasi tentang ketersediaan mesin basis data tertentu, di Autentikasi basis data IAM.

Autentikasi Kerberos

Amazon Aurora mendukung autentikasi eksternal pengguna basis data dengan menggunakan Kerberos dan Microsoft Active Directory. Kerberos adalah protokol autentikasi jaringan yang menggunakan tiket dan kriptografi kunci simetris untuk menghilangkan kebutuhan mengirim kata sandi melalui jaringan. Kerberos telah tertanam ke dalam Active Directory dan dirancang untuk mengautentikasi pengguna ke sumber daya jaringan, seperti basis data.

Dukungan Amazon Aurora untuk Kerberos dan Active Directory memberikan manfaat upaya masuk tunggal dan autentikasi terpusat pengguna basis data. Anda dapat menyimpan kredensial pengguna Anda di Active Directory. Active Directory menyediakan tempat terpusat untuk menyimpan dan mengelola kredensial bagi beberapa klaster basis data.

Anda dapat memungkinkan pengguna basis data untuk mengautentikasi klaster basis data dengan dua cara. Pengguna dapat menggunakan kredensial yang disimpan di AWS Directory Service for Microsoft Active Directory atau di Active Directory on-premise Anda.

Aurora mendukung autentikasi Kerberos untuk klaster basis data Aurora MySQL dan Aurora PostgreSQL. Lihat informasi selengkapnya tentang autentikasi Kerberos untuk Aurora MySQL di Menggunakan autentikasi Kerberos untuk Aurora MySQL.

Dengan autentikasi Kerberos, klaster basis data Aurora PostgreSQL mendukung hubungan kepercayaan rimba satu dan dua arah. Lihat informasi selengkapnya di Menggunakan autentikasi Kerberos dengan Aurora PostgreSQL.

Manajemen kata sandi dengan dan AWS Secrets Manager

Amazon Aurora terintegrasi dengan Secrets Manager untuk mengelola kata sandi pengguna utama untuk klaster .

Topik

- Ketersediaan wilayah dan versi
- Batasan untuk integrasi Secrets Manager dengan Amazon Aurora
- Ikhtisar mengelola kata sandi pengguna master dengan AWS Secrets Manager
- Manfaat mengelola kata sandi pengguna utama dengan Secrets Manager
- Izin yang diperlukan untuk integrasi Secrets Manager
- Menegakkan manajemen dari kata sandi pengguna utama di AWS Secrets Manager
- Mengelola kata sandi pengguna utama untuk klaster DB dengan Secrets Manager
- Merotasi rahasia kata sandi pengguna utama untuk klaster DB
- Melihat detail tentang rahasia untuk klaster DB

Ketersediaan wilayah dan versi

Ketersediaan dan dukungan fitur bervariasi di seluruh versi khusus dari setiap mesin basis data dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang ketersediaan versi dan Wilayah dengan integrasi Secrets Manager dengan Amazon Aurora, lihat Integrasi Secrets Manager.

Batasan untuk integrasi Secrets Manager dengan Amazon Aurora

Mengelola kata sandi pengguna utama dengan Secrets Manager tidak didukung untuk fitur berikut:

- Deployment Blue/Green Amazon RDS
- Klaster DB yang merupakan bagian dari basis data global Aurora
- Klaster DB Aurora Serverless v1
- Replika baca lintas Wilayah Aurora MySQL
- Mengelola kata sandi pengguna utama dengan Secrets Manager untuk replika baca

Ikhtisar mengelola kata sandi pengguna master dengan AWS Secrets Manager

Dengan AWS Secrets Manager, Anda dapat mengganti kredensi hard-code dalam kode Anda, termasuk kata sandi database, dengan panggilan API ke Secrets Manager untuk mengambil rahasia secara terprogram. Untuk mengetahui informasi selengkapnya tentang Secrets Manager, lihat Panduan Pengguna AWS Secrets Manager.

Ketika Anda menyimpan rahasia database di Secrets Manager, Anda akan Akun AWS dikenakan biaya. Untuk informasi tentang harga, lihat AWS Secrets Manager Harga.

Anda dapat menentukan bahwa Aurora mengelola kata sandi pengguna utama di Secrets Manager untuk klaster DB Amazon Aurora saat Anda melakukan salah satu operasi berikut:

- Membuat replika baca klaster DB
- Mengubah klaster DB
- Memulihkan klaster DB dari Amazon S3 (khusus Aurora MySQL)

Saat Anda menentukan bahwa Aurora mengelola kata sandi pengguna utama di Secrets Manager, Aurora menghasilkan kata sandi dan menyimpannya dalam Secrets Manager. Anda dapat berinteraksi langsung dengan rahasia untuk mengambil kredensial untuk pengguna utama. Anda juga dapat menentukan kunci yang dikelola pelanggan untuk mengenkripsi rahasia, atau menggunakan kunci KMS default yang disediakan oleh Secrets Manager.

Secara default, Aurora mengelola pengaturan untuk rahasia dan merotasi rahasia setiap tujuh hari. Anda dapat mengubah beberapa pengaturan, seperti jadwal rotasi. Jika Anda menghapus klaster DB yang mengelola rahasia di Secrets Manager, rahasia dan metadata terkaitnya juga akan dihapus.

Untuk terhubung ke klaster DB dengan kredensial dalam rahasia, Anda dapat mengambil rahasia dari Secrets Manager. Untuk informasi selengkapnya, lihat Mengambil rahasia dari AWS Secrets Manager dan Connect ke database SQL dengan kredensi dalam AWS Secrets Manager rahasia di Panduan Pengguna. AWS Secrets Manager

Manfaat mengelola kata sandi pengguna utama dengan Secrets Manager

Mengelola kata sandi pengguna utama Aurora dengan Secrets Manager memberikan manfaat berikut:

- Aurora secara otomatis menghasilkan kredensial basis data.
- Aurora secara otomatis menyimpan dan mengelola kredensil database di. AWS Secrets Manager
- Aurora merotasi kredensial basis data secara teratur, tanpa mewajibkan perubahan aplikasi.
- Secrets Manager mengamankan kredensial basis data dari akses manusia dan tampilan teks biasa.
- Secrets Manager memungkinkan pengambilan kredensial basis data rahasia untuk koneksi basis data.
- Secrets Manager memungkinkan kontrol akses terperinci ke kredensial basis data dalam rahasia menggunakan IAM.
- Secara opsional, Anda dapat memisahkan enkripsi basis data dari enkripsi kredensial dengan kunci KMS lainnya.
- Anda dapat menghilangkan rotasi dan manajemen manual kredensial basis data.
- Anda dapat memantau kredensi database dengan mudah dengan dan AWS CloudTrail Amazon. CloudWatch

Untuk informasi selengkapnya tentang manfaat Secrets Manager, lihat Panduan Pengguna AWS Secrets Manager.

Izin yang diperlukan untuk integrasi Secrets Manager

Pengguna harus memiliki izin yang diperlukan untuk melakukan operasi yang terkait dengan integrasi Secrets Manager. Anda dapat membuat kebijakan IAM yang memberikan izin untuk melakukan operasi API tertentu pada sumber daya spefisik yang diperlukan. Anda kemudian dapat melampirkan kebijakan tersebut ke set izin IAM atau peran yang memerlukan izin tersebut. Untuk informasi selengkapnya, lihat Manajemen identitas dan akses untuk Amazon Aurora.

Untuk membuat, memodifikasi, atau memulihkan operasi, pengguna yang menentukan bahwa Aurora mengelola kata sandi pengguna utama di Secrets Manager harus memiliki izin untuk melakukan operasi berikut:

- `kms:DescribeKey`
- `secretsmanager:CreateSecret`
- `secretsmanager:TagResource`

Untuk membuat, memodifikasi, atau memulihkan operasi, pengguna yang menentukan kunci yang dikelola pelanggan untuk mengenkripsi rahasia dalam Secrets Manager harus memiliki izin untuk melakukan operasi berikut:

- `kms:Decrypt`
- `kms:GenerateDataKey`
- `kms:CreateGrant`

Untuk mengubah operasi, pengguna yang merotasi kata sandi pengguna utama dalam Secrets Manager harus memiliki izin untuk melakukan operasi berikut:

- `secretsmanager:RotateSecret`

Menegakkan manajemen dari kata sandi pengguna utama di AWS Secrets Manager

Anda dapat menggunakan kunci kondisi IAM untuk menerapkan manajemen Aurora kata sandi pengguna utama di AWS Secrets Manager. Kebijakan berikut tidak mengizinkan pengguna untuk membuat atau memulihkan instans DB atau klaster DB kecuali kata sandi pengguna utama dikelola oleh Aurora di Secrets Manager.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": ["rds:CreateDBInstance", "rds:CreateDBCluster",
 "rds:RestoreDBInstanceFromS3", "rds:RestoreDBClusterFromS3"],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "rds:ManageMasterUserPassword": false
                }
            }
        }
    ]
}
```

> **Note**
>
> Kebijakan ini memberlakukan manajemen kata sandi pada AWS Secrets Manager saat pembuatan. Namun, Anda masih dapat menonaktifkan integrasi Secrets Manager dan mengatur kata sandi utama secara manual dengan mengubah klaster.
> Untuk mencegahnya, sertakan `rds:ModifyDBInstance`, `rds:ModifyDBCluster` dalam blok tindakan kebijakan. Perhatikan bahwa tindakan ini akan mencegah pengguna menerapkan perubahan lebih lanjut pada klaster yang ada yang Secrets Manager-nya tidak diaktifkan.

Untuk informasi lebih lanjut tentang penggunaan kunci kondisi dalam kebijakan IAM, lihat Kunci kondisi kebijakan untuk Aurora dan Contoh Kebijakan: Menggunakan kunci kondisi.

Mengelola kata sandi pengguna utama untuk klaster DB dengan Secrets Manager

Anda dapat mengonfigurasi manajemen Aurora kata sandi pengguna utama di Secrets Manager saat Anda melakukan tindakan berikut:

- Membuat klaster DB Amazon Aurora
- Memodifikasi klaster DB Amazon Aurora
- Memigrasikan data dari basis data MySQL eksternal ke klaster DB Amazon Aurora MySQL

Anda dapat menggunakan konsol RDS, API AWS CLI, atau RDS untuk melakukan tindakan ini.

Konsol

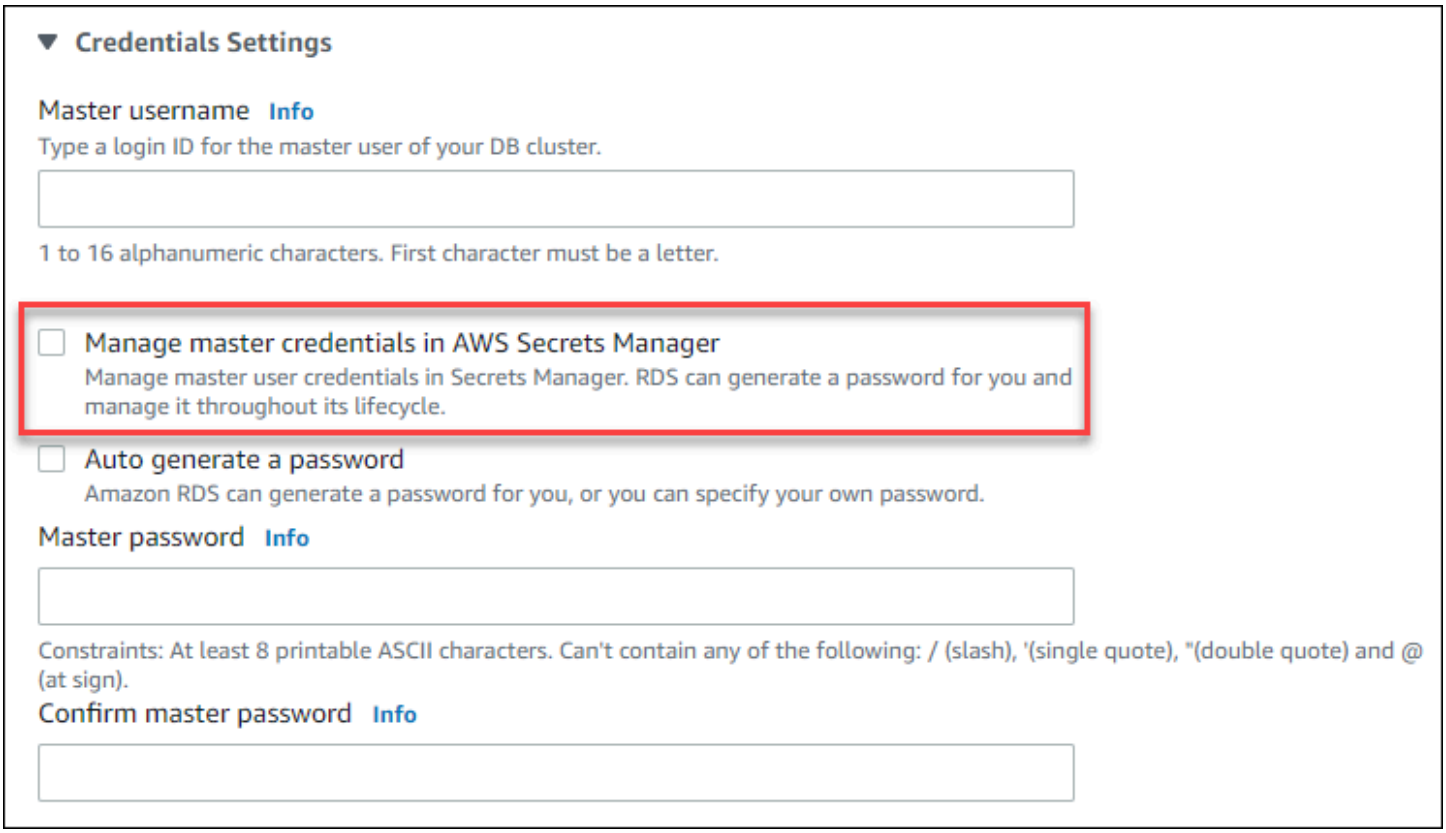
Ikuti petunjuk untuk membuat atau mengubah klaster DB dengan konsol RDS:

- Membuat klaster DB
- Memodifikasi instans DB dalam klaster DB

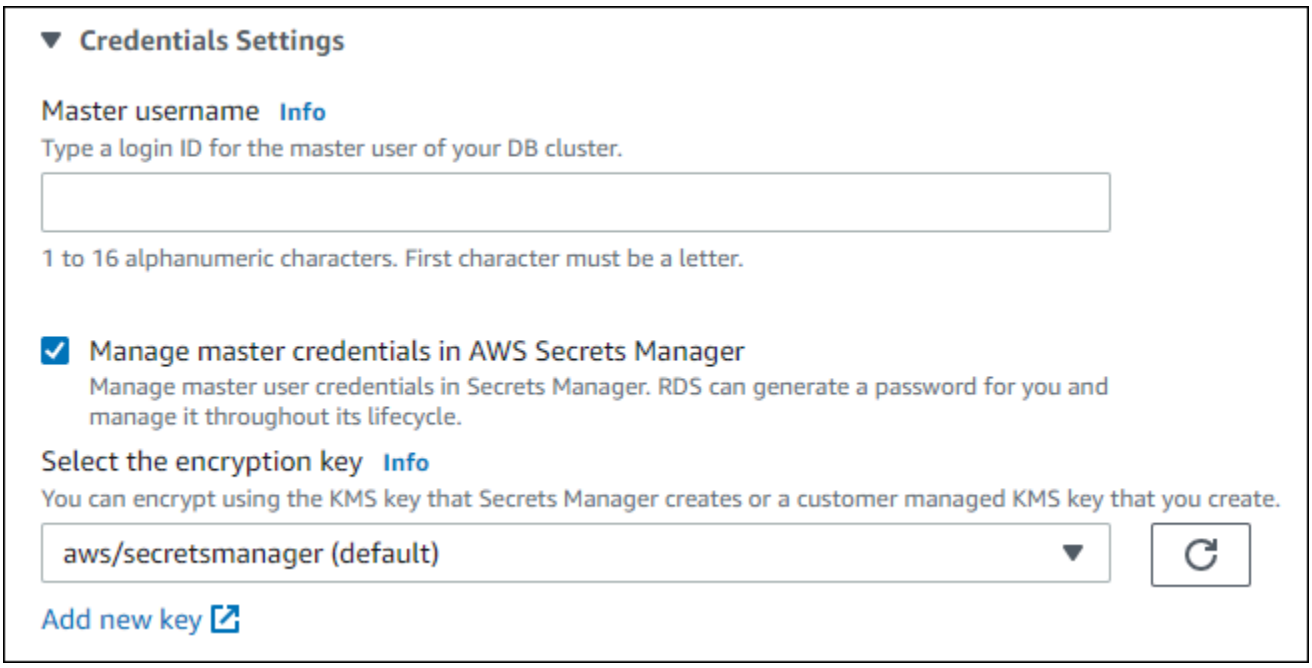
  Di konsol RDS, Anda dapat mengubah instans DB apa pun untuk menentukan pengaturan manajemen kata sandi pengguna utama untuk seluruh klaster DB.
- Memulihkan klaster DB Amazon Aurora MySQL dari bucket Amazon S3

Saat menggunakan konsol RDS untuk melakukan salah satu operasi ini, Anda dapat menentukan bahwa kata sandi pengguna utama dikelola oleh Aurora di Secrets Manager. Untuk melakukannya saat membuat atau memulihkan klaster DB, pilih Kelola kredensial utama di AWS Secrets Manager dalam Pengaturan kredensial. Saat Anda mengubah klaster DB, pilih Kelola kredensial utama di AWS Secrets Manager dalam Pengaturan.

Gambar berikut adalah contoh pengaturan Kelola kredensial utama di AWS Secrets Manager saat Anda membuat atau memulihkan klaster DB.



Jika Anda memilih opsi ini, Aurora akan menghasilkan kata sandi pengguna utama dan mengelolanya sepanjang siklus pemakaiannya di Secrets Manager.



Anda dapat memilih untuk mengenkripsi rahasia dengan kunci KMS yang disediakan Secrets Manager atau dengan kunci yang dikelola pelanggan yang Anda buat. Setelah Aurora mengelola kredensial basis data untuk klaster DB, Anda tidak dapat mengubah kunci KMS yang digunakan untuk mengenkripsi rahasia.

Anda dapat memilih pengaturan lain untuk memenuhi kebutuhan Anda.

Untuk informasi selengkapnya tentang pengaturan yang tersedia saat Anda membuat klaster DB, lihat Pengaturan untuk klaster Aurora DB. Untuk informasi selengkapnya tentang pengaturan yang tersedia saat Anda mengubah klaster DB, lihat Pengaturan untuk Amazon Aurora.

AWS CLI

Untuk menentukan bahwa Aurora mengelola kata sandi pengguna utama di Secrets Manager, tentukan opsi `--manage-master-user-password` di salah satu perintah berikut:

- create-db-cluster
- modify-db-cluster
- restore-db-cluster-from-s3

Jika Anda menentukan opsi `--manage-master-user-password` dalam perintah ini, Aurora akan menghasilkan kata sandi pengguna utama dan mengelolanya sepanjang siklus pemakaiannya di Secrets Manager.

Untuk mengenkripsi rahasia, Anda dapat menentukan kunci yang dikelola pelanggan atau menggunakan kunci KMS default yang disediakan oleh Secrets Manager. Gunakan opsi `--master-user-secret-kms-key-id` untuk menentukan kunci yang dikelola pelanggan. Pengidentifikasi kunci AWS KMS adalah kunci ARN, ID kunci, alias ARN, atau nama alias untuk kunci KMS. Untuk menggunakan kunci KMS yang berbeda Akun AWS, tentukan kunci ARN atau alias ARN. Setelah Aurora mengelola kredensial basis data untuk klaster DB, Anda tidak dapat mengubah kunci KMS yang digunakan untuk mengenkripsi rahasia.

Anda dapat memilih pengaturan lain untuk memenuhi kebutuhan Anda.

Untuk informasi selengkapnya tentang pengaturan yang tersedia saat Anda membuat klaster DB, lihat Pengaturan untuk klaster Aurora DB. Untuk informasi selengkapnya tentang pengaturan yang tersedia saat Anda mengubah klaster DB, lihat Pengaturan untuk Amazon Aurora.

Contoh ini membuat klaster DB dan menentukan bahwa Aurora mengelola kata sandi di Secrets Manager. Rahasianya dienkripsi menggunakan kunci KMS yang disediakan oleh Secrets Manager.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds create-db-cluster \
     --db-cluster-identifier sample-cluster \
     --engine aurora-mysql \
     --engine-version 8.0 \
     --master-username admin \
     --manage-master-user-password
```

Untuk Windows:

```
aws rds create-db-cluster ^
     --db-cluster-identifier sample-cluster ^
     --engine aurora-mysql ^
     --engine-version 8.0 ^
     --master-username admin ^
     --manage-master-user-password
```

API RDS

Untuk menentukan bahwa Aurora mengelola kata sandi pengguna utama di Secrets Manager, atur parameter `ManageMasterUserPassword` ke `true` di salah satu operasi berikut:

- CreateDBCluster
- ModifyDBCluster
- DipulihkanB S3 ClusterFrom

Jika Anda mengatur parameter `ManageMasterUserPassword` ke `true` di salah satu operasi ini, Aurora akan menghasilkan kata sandi pengguna utama dan mengelolanya sepanjang siklus pemakaiannya di Secrets Manager.

Untuk mengenkripsi rahasia, Anda dapat menentukan kunci yang dikelola pelanggan atau menggunakan kunci KMS default yang disediakan oleh Secrets Manager. Gunakan parameter `MasterUserSecretKmsKeyId` untuk menentukan kunci yang dikelola pelanggan. Pengidentifikasi kunci AWS KMS adalah kunci ARN, ID kunci, alias ARN, atau nama alias untuk kunci KMS. Untuk menggunakan kunci KMS di Akun AWS yang berbeda, tentukan ARN kunci atau ARN alias. Setelah Aurora mengelola kredensial basis data untuk klaster DB, Anda tidak dapat mengubah kunci KMS yang digunakan untuk mengenkripsi rahasia.

Merotasi rahasia kata sandi pengguna utama untuk klaster DB

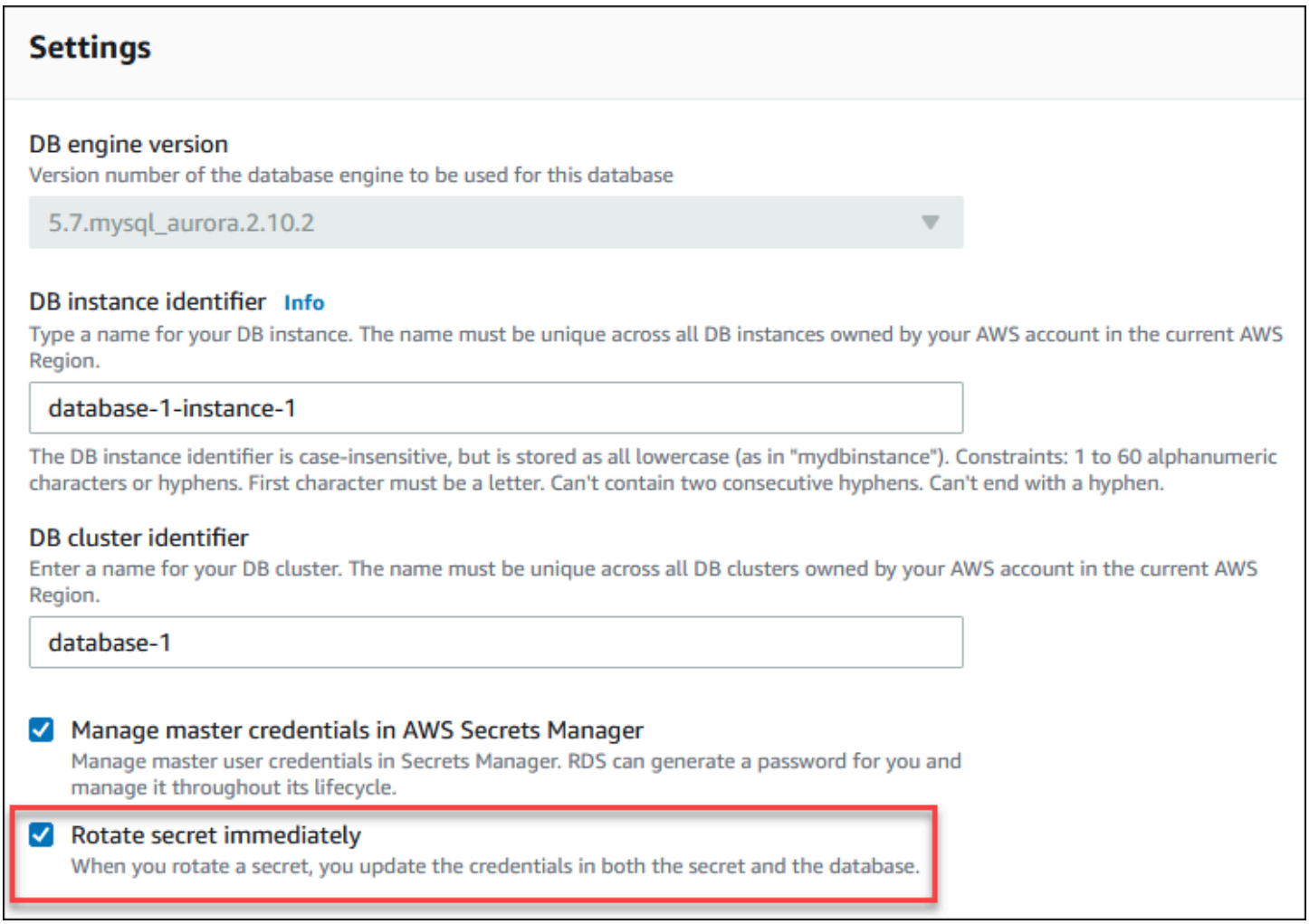
Ketika Aurora merotasi rahasia kata sandi pengguna utama, Secrets Manager akan menghasilkan versi rahasia baru untuk rahasia yang sudah ada. Rahasia versi baru berisi kata sandi pengguna utama baru. Aurora mengubah kata sandi pengguna utama untuk klaster DB agar sesuai dengan kata sandi versi rahasia baru.

Anda dapat segera merotasi rahasia, alih-alih menunggu rotasi yang dijadwalkan. Untuk merotasi rahasia kata sandi pengguna utama di Secrets Manager, ubah klaster DB . Untuk informasi tentang cara mengubah klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Anda dapat memutar rahasia kata sandi pengguna master segera dengan konsol RDS, API AWS CLI, atau RDS. Kata sandi baru selalu sepanjang 28 karakter dan berisi setidaknya satu karakter huruf besar dan kecil, satu angka, dan satu tanda baca.

Konsol

Untuk merotasi rahasia kata sandi pengguna utama menggunakan konsol RDS, ubah klaster DB dan pilih Rotasi rahasia secara langsung di Pengaturan.



Ikuti petunjuk untuk mengubah klaster DB dengan konsol RDS di Memodifikasi klaster DB dengan menggunakan konsol, CLI, dan API. Anda harus memilih Terapkan langsung di halaman konfirmasi.

AWS CLI

Untuk memutar rahasia kata sandi pengguna master menggunakan AWS CLI, gunakan modify-db-clusterperintah dan tentukan `--rotate-master-user-password` opsi. Anda harus menentukan opsi `--apply-immediately` saat merotasi kata sandi utama.

Contoh ini merotasi rahasia kata sandi pengguna utama.

Example

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-cluster \
```

```
    --db-cluster-identifier mydbcluster \
    --rotate-master-user-password \
    --apply-immediately
```

Untuk Windows:

```
aws rds modify-db-cluster ^
    --db-cluster-identifier mydbcluster ^
    --rotate-master-user-password ^
    --apply-immediately
```

API RDS

Anda dapat merotasi rahasia kata sandi pengguna utama menggunakan operasi ModifyDBCluster dan mengatur parameter `RotateMasterUserPassword` ke `true`. Anda harus mengatur parameter `ApplyImmediately` ke `true` saat merotasi kata sandi utama.

Melihat detail tentang rahasia untuk klaster DB

Anda dapat mengambil rahasia Anda menggunakan konsol (https://console.aws.amazon.com/secretsmanager/) atau perintah AWS CLI (get-secret-valueSecrets Manager).

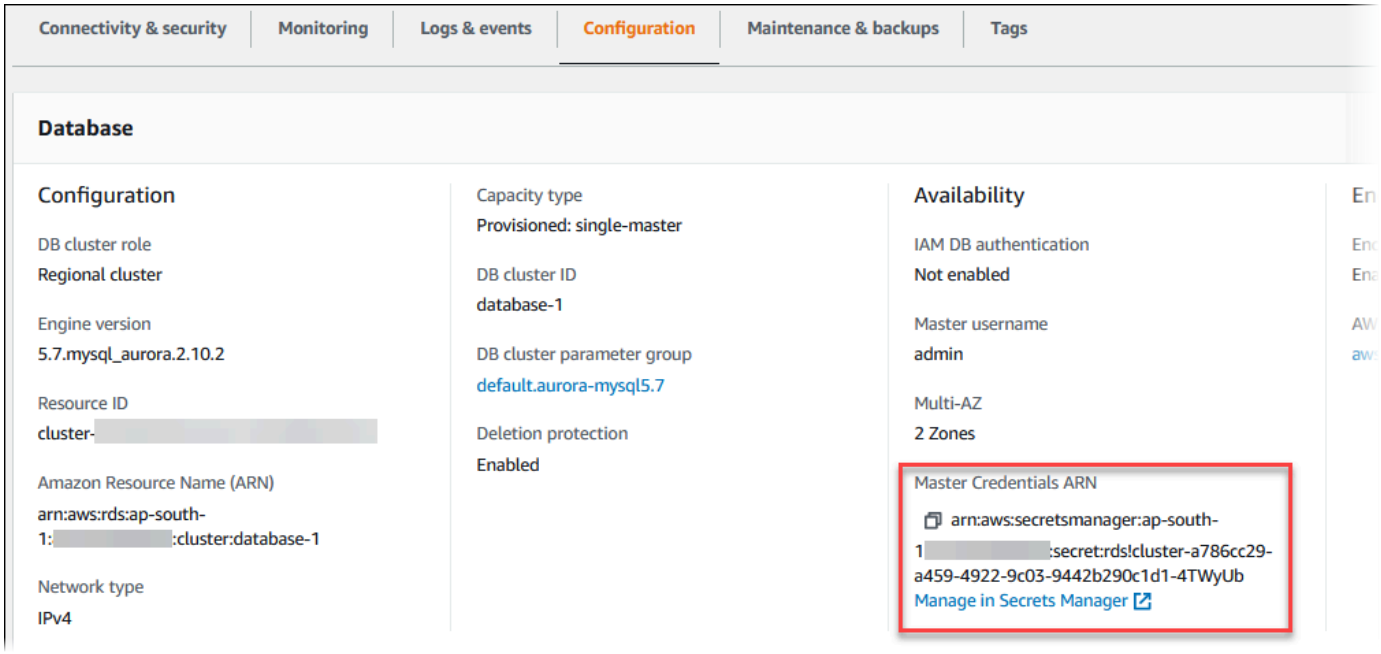
Anda dapat menemukan Amazon Resource Name (ARN) dari rahasia yang dikelola oleh Aurora di Secrets Manager dengan konsol RDS AWS CLI,, atau RDS API.

Konsol

Untuk melihat detail tentang rahasia yang dikelola oleh Aurora di Secrets Manager

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih basis data.
3. Pilih nama klaster DB untuk menampilkan detailnya.
4. Pilih tab Konfigurasi.

   Di ARN Kredensial Utama, Anda dapat melihat ARN rahasia.



Anda dapat mengikuti tautan Mengelola di Secrets Manager untuk melihat dan mengelola rahasia di konsol Secrets Manager.

AWS CLI

Anda dapat menggunakan AWS CLI describe-db-clustersperintah RDS untuk menemukan informasi berikut tentang rahasia yang dikelola oleh di Secrets Manager:

- `SecretArn` – ARN rahasia
- `SecretStatus` – Status rahasia

  Kemungkinan nilai statusnya meliputi:

  - `creating` – Rahasia sedang dibuat.
  - `active` – Rahasia tersedia untuk penggunaan normal dan rotasi.
  - `rotating` – Rahasia sedang dirotasi.
  - `impaired` – Rahasia dapat digunakan untuk mengakses kredensial basis data, tetapi tidak dapat dirotasi. Rahasia mungkin memiliki status ini jika, misalnya, izin diubah sehingga RDS tidak dapat lagi mengakses rahasia atau kunci KMS untuk rahasia tersebut.

    Ketika rahasia memiliki status ini, Anda dapat memperbaiki kondisi yang menyebabkan status tersebut. Jika Anda memperbaiki kondisi yang menyebabkan status, status tersebut tetap `impaired` hingga rotasi berikutnya. Sebagai alternatif, Anda dapat mengubah klaster DB untuk

menonaktifkan manajemen otomatis kredensial basis data, dan kemudian mengubah klaster DB lagi untuk mengaktifkan manajemen otomatis kredensial basis data. Untuk memodifikasi cluster DB, gunakan `--manage-master-user-password` opsi dalam modify-db-clusterperintah.

- `KmsKeyId` – ARN kunci KMS yang digunakan untuk mengenkripsi rahasia

Tentukan opsi `--db-cluster-identifier` untuk menampilkan output untuk klaster DB tertentu. Contoh ini menunjukkan output untuk rahasia yang digunakan oleh klaster DB.

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydbcluster
```

Contoh berikut menunjukkan output untuk rahasia:

```
"MasterUserSecret": {
                "SecretArn": "arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx",
                "SecretStatus": "active",
                "KmsKeyId": "arn:aws:kms:eu-
west-1:123456789012:key/0987dcba-09fe-87dc-65ba-ab0987654321"
            }
```

Ketika Anda memiliki ARN rahasia, Anda dapat melihat detail tentang rahasia menggunakan perintah get-secret-valueSecrets Manager CLI.

Contoh ini menunjukkan detail untuk rahasia dalam output contoh sebelumnya.

Example

Untuk Linux, macOS, atau Unix:

```
aws secretsmanager get-secret-value \
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

Untuk Windows:

```
aws secretsmanager get-secret-value ^
```

```
    --secret-id 'arn:aws:secretsmanager:eu-west-1:123456789012:secret:rds!
cluster-033d7456-2c96-450d-9d48-f5de3025e51c-xmJRDx'
```

API RDS

Anda dapat melihat ARN, status, dan kunci KMS untuk rahasia yang dikelola oleh Aurora di Secrets Manager menggunakan operasi RDS DescribeDBClusters dan mengatur parameter `DBClusterIdentifier` ke ID klaster DB. Detil tentang rahasia disertakan dalam output.

Ketika Anda memiliki ARN rahasia, Anda dapat melihat detail tentang rahasia menggunakan operasi GetSecretValueSecrets Manager.

Perlindungan data di Amazon RDS

Model tanggung jawab bersama AWS berlaku untuk perlindungan data di Amazon Relational Database Service. Sebagaimana diuraikan dalam model ini, AWS bertanggung jawab untuk melindungi infrastruktur global yang menjalankan semua AWS Cloud. Anda bertanggung jawab untuk memelihara kendali atas isi yang dihost pada infrastruktur ini. Anda juga bertanggung jawab atas tugas-tugas konfigurasi dan manajemen keamanan untuk Layanan AWS yang Anda gunakan. Lihat informasi yang lebih lengkap tentang privasi data dalam Pertanyaan Umum Privasi Data. Lihat informasi tentang perlindungan data di Eropa di pos blog Model Tanggung Jawab Bersama dan GDPR AWS di Blog Keamanan AWS.

Untuk tujuan perlindungan data, sebaiknya lindungi kredensial Akun AWS dan siapkan untuk masing-masing pengguna AWS IAM Identity Center atau AWS Identity and Access Management (IAM). Dengan cara itu, setiap pengguna hanya diberi izin yang diperlukan untuk memenuhi tanggung jawab tugasnya. Kami juga menyarankan supaya Anda mengamankan data dengan cara-cara berikut:

- Gunakan autentikasi multi-faktor (MFA) pada setiap akun.
- Gunakan SSL/TLS untuk berkomunikasi dengan sumber daya AWS. Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Siapkan API dan pengelogan aktivitas pengguna dengan AWS CloudTrail.
- Gunakan solusi enkripsi AWS, bersama semua kontrol keamanan bawaan dalam Layanan AWS.
- Gunakan layanan keamanan terkelola lanjut seperti Amazon Macie, yang membantu menemukan dan mengamankan data sensitif yang disimpan di Amazon S3.
- Jika Anda memerlukan modul kriptografi tervalidasi FIPS 140-2 ketika mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Lihat informasi yang lebih lengkap tentang titik akhir FIPS yang tersedia di Standar Pemrosesan Informasi Federal (FIPS) 140-2.

Kami sangat menganjurkan supaya Anda tidak memasukkan informasi rahasia atau sensitif, seperti alamat email pelanggan, ke dalam tag atau bidang teks isian bebas seperti bidang Nama. Hal ini mencakup ketika Anda bekerja dengan Amazon RDS atau Layanan AWS lain dengan menggunakan konsol, API, AWS CLI, atau AWS SDK. Data apa pun yang Anda masukkan ke dalam tag atau bidang teks isian bebas yang digunakan untuk nama dapat digunakan untuk log penagihan atau diagnostik. Saat Anda memberikan URL ke server eksternal, kami sangat menganjurkan supaya Anda tidak menyertakan informasi kredensial di dalam URL untuk memvalidasi permintaan Anda ke server itu.

Topik

- Melindungi data menggunakan enkripsi
- Privasi lalu lintas antarjaringan

Melindungi data menggunakan enkripsi

Anda dapat mengaktifkan enkripsi untuk sumber daya basis data. Anda juga dapat mengenkripsi koneksi ke klaster DB.

Topik

- Mengenkripsi sumber daya Amazon Aurora
- Manajemen AWS KMS key
-
- Merotasi sertifikat SSL/TLS

Mengenkripsi sumber daya Amazon Aurora

Amazon Aurora dapat mengenkripsi klaster DB Amazon Aurora Anda. Data yang dienkripsi saat diam termasuk penyimpanan dasar untuk klaster DB, pencadangan otomatisnya, replika baca, dan snapshot.

klaster DB terenkripsi Amazon Aurora menggunakan algoritma AES-256 standar industri untuk mengenkripsi data Anda di server yang meng-host klaster DB Amazon Aurora Anda. Setelah data Anda dienkripsi, Amazon Aurora menangani autentikasi akses dan dekripsi data Anda secara transparan dengan dampak minimal terhadap performa. Anda tidak perlu memodifikasi aplikasi klien basis data untuk menggunakan enkripsi.

> **Note**
>
> Untuk cluster DB terenkripsi dan tidak terenkripsi, data yang sedang transit antara sumber dan replika baca dienkripsi, bahkan saat mereplikasi di seluruh Wilayah. AWS

Topik

- Ikhtisar mengenkripsi sumber daya Amazon Aurora
- Membuat klaster DB Amazon Aurora
- Menentukan apakah enkripsi untuk klaster DB diaktifkan
- Ketersediaan enkripsi Amazon Aurora
- Enkripsi dalam bergerak

Ikhtisar mengenkripsi sumber daya Amazon Aurora

Klaster DB terenkripsi Amazon Aurora menyediakan lapisan perlindungan data tambahan dengan mengamankan data Anda dari akses tidak sah ke penyimpanan dasar. Anda dapat menggunakan enkripsi Amazon Aurora untuk meningkatkan perlindungan data di aplikasi Anda yang di-deploy di cloud, dan untuk memenuhi persyaratan kepatuhan enkripsi diam.

Untuk klaster DB terenkripsi Amazon Aurora, semua instans DB, log, backup, dan snapshot dienkripsi. Anda juga dapat mengenkripsi replika baca dari klaster terenkripsi Amazon Aurora. Amazon Aurora menggunakan AWS KMS key untuk mengenkripsi sumber daya ini. Untuk informasi selengkapnya tentang kunci KMS, lihat AWS KMS keys di Panduan Developer AWS Key Management Service dan Manajemen AWS KMS key. Setiap instans DB di klaster DB dienkripsi menggunakan kunci KMS yang sama dengan klaster DB. Jika Anda menyalin snapshot terenkripsi, Anda dapat menggunakan kunci KMS yang berbeda untuk mengenkripsi snapshot target, bukan yang digunakan untuk mengenkripsi snapshot sumber.

Anda dapat menggunakan Kunci yang dikelola AWS, atau Anda dapat membuat kunci yang dikelola pelanggan. Untuk mengelola kunci yang dikelola pelanggan yang digunakan untuk mengenkripsi dan mendekripsi sumber daya Amazon Aurora Anda, gunakan AWS Key Management Service (AWS KMS). AWS KMS menggabungkan perangkat keras dan perangkat lunak yang aman dan selalu tersedia untuk menyediakan sistem manajemen kunci yang disesuaikan untuk cloud. Dengan menggunakan AWS KMS, Anda dapat membuat kunci terkelola pelanggan dan menentukan

kebijakan yang mengontrol bagaimana kunci yang dikelola pelanggan ini dapat digunakan. AWS KMS mendukung CloudTrail, sehingga Anda dapat mengaudit penggunaan kunci KMS untuk memverifikasi bahwa kunci yang dikelola pelanggan digunakan dengan tepat. Anda dapat menggunakan kunci yang dikelola pelanggan dengan Amazon Aurora dan AWS layanan yang didukung seperti Amazon S3, Amazon EBS, dan Amazon Redshift. Untuk daftar layanan yang terintegrasi AWS KMS, lihat Integrasi AWS Layanan.

Membuat klaster DB Amazon Aurora

Untuk mengenkripsi klaster DB baru, pilih Aktifkan enkripsi di konsol. Untuk informasi tentang membuat klaster DB baru, lihat Membuat klaster DB Amazon Aurora.

Jika Anda menggunakan create-db-cluster AWS CLI perintah untuk membuat cluster DB terenkripsi, atur parameternya. `--storage-encrypted` Jika Anda menggunakan Operasi API CreateDBCluster, atur parameter `StorageEncrypted` ke true.

Saat Anda membuat klaster DB terenkripsi, Anda dapat memilih kunci yang dikelola pelanggan atau Amazon Aurora Kunci yang dikelola AWS untuk mengenkripsi klaster DB Anda. Jika Anda tidak menentukan pengenal kunci untuk kunci yang dikelola pelanggan, Amazon Aurora menggunakan Kunci yang dikelola AWS untuk klaster DB baru Anda. Amazon Aurora membuat untuk Amazon Aurora Kunci yang dikelola AWS untuk akun Anda. AWS AWS Akun Anda memiliki perbedaan Kunci yang dikelola AWS untuk Amazon Aurora untuk setiap AWS Wilayah.

Untuk informasi selengkapnya tentang kunci KMS, lihat AWS KMS keys di Panduan Developer AWS Key Management Service .

Setelah Anda membuat klaster DB terenkripsi, Anda tidak dapat mengubah kunci KMS yang digunakan oleh klaster DB tersebut. Oleh karena itu, pastikan untuk menentukan persyaratan kunci KMS Anda sebelum membuat klaster DB terenkripsi Anda.

Jika Anda menggunakan AWS CLI `create-db-cluster` perintah untuk membuat cluster DB terenkripsi dengan kunci yang dikelola pelanggan, setel `--kms-key-id` parameter ke pengidentifikasi kunci apa pun untuk kunci KMS. Jika Anda menggunakan operasi `CreateDBInstance` API Amazon RDS, atur parameter `KmsKeyId` ke pengidentifikasi kunci mana pun untuk kunci KMS. Untuk menggunakan kunci yang dikelola pelanggan di akun AWS yang berbeda, tentukan ARN kunci atau ARN alias.

Important

Amazon Aurora dapat kehilangan akses ke kunci KMS untuk klaster DB. Misalnya, Aurora kehilangan akses saat kunci KMS tidak diaktifkan, atau saat akses Aurora ke kunci KMS dicabut. Dalam kasus ini, klaster DB terenkripsi menjadi berstatus `inaccessible-encryption-credentials-recoverable`. Klaster DB tetap berlangsung selama tujuh hari. Jika selama waktu ini Anda memulai klaster DB, klaster akan memeriksa apakah kunci KMS aktif, dan jika aktif, maka klaster DB akan dipulihkan. Mulai ulang cluster DB menggunakan AWS CLI perintah start-db-clusteratau AWS Management Console.
Jika klaster DB tidak dipulihkan, maka klaster DB tersebut menjadi berstatus `inaccessible-encryption-credentials` terminal. Dalam hal ini, Anda hanya dapat memulihkan klaster DB dari pencadangan. Sebaiknya selalu aktifkan pencadangan instans DB terenkripsi agar data terenkripsi di basis data Anda tidak hilang.

Menentukan apakah enkripsi untuk klaster DB diaktifkan

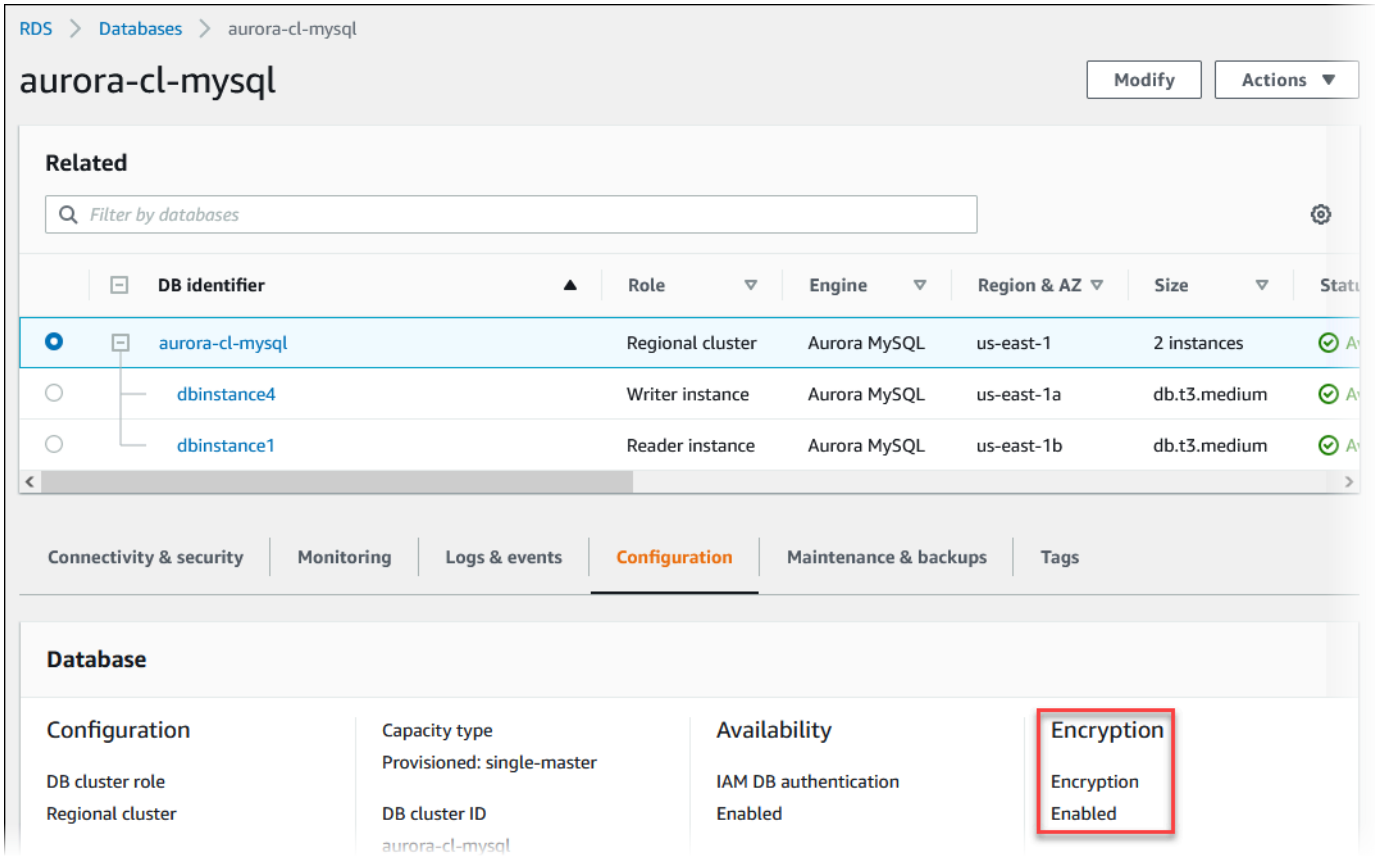
Anda dapat menggunakan AWS Management Console, AWS CLI, atau RDS API untuk menentukan apakah enkripsi saat istirahat diaktifkan untuk cluster DB.

Konsol

Untuk menentukan apakah enkripsi saat istirahat diaktifkan untuk klaster DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis Data.
3. Pilih nama klaster DB yang ingin Anda periksa untuk melihat detailnya.
4. Pilih tab Konfigurasi dan periksa nilai Enkripsi.

   Akan muncul Diaktifkan atau Tidak diaktifkan.



AWS CLI

Untuk menentukan apakah enkripsi saat istirahat diaktifkan untuk cluster DB dengan menggunakan AWS CLI, panggil describe-db-clustersperintah dengan opsi berikut:

- `--db-cluster-identifier` – Nama klaster DB.

Contoh berikut menggunakan kueri untuk mengembalikan salah satu `TRUE` atau `FALSE` mengenai enkripsi saat istirahat untuk klaster DB `mydb`.

Example

```
aws rds describe-db-clusters --db-cluster-identifier mydb --query "*[].
{StorageEncrypted:StorageEncrypted}" --output text
```

RDS API

Untuk menentukan apakah enkripsi diam untuk klaster DB diaktifkan menggunakan Amazon RDS API, panggil operasi DescribeDBClusters dengan parameter berikut:

- `DBClusterIdentifier` – Nama klaster DB.

Ketersediaan enkripsi Amazon Aurora

Enkripsi Amazon Aurora saat ini tersedia untuk semua mesin basis data dan jenis penyimpanan, .

> **Note**
>
> Enkripsi Amazon Aurora tidak tersedia untuk kelas instans DB db.t2.micro.

Enkripsi dalam bergerak

AWS menyediakan konektivitas aman dan pribadi antara instans DB dari semua jenis. Selain itu, beberapa tipe instans menggunakan kemampuan offload dari perangkat keras Nitro System yang mendasarinya untuk secara otomatis mengenkripsi lalu lintas dalam transit antar instans. Enkripsi ini menggunakan algoritma Authenticated Encryption with Associated Data (AEAD), dengan enkripsi 256-bit. Tidak ada dampak terhadap performa jaringan. Untuk mendukung enkripsi lalu lintas dalam transit tambahan ini antara instans, persyaratan-persyaratan berikut harus dipenuhi:

- Instans-instans tersebut menggunakan tipe instans berikut:
  - Tujuan umum: M6i, M6iD, M6in, M6idn, M7g
  - Memori yang dioptimalkan: R6i, R6id, R6in, R6idn, R7g, X2idn, X2IEDN, X2IEZN
- Contohnya sama Wilayah AWS.
- Instans-instans tersebut berada dalam VPC yang sama atau VPC yang di-peering yang sama, dan lalu lintas tidak melewati perangkat atau layanan jaringan virtual, seperti penyeimbang beban atau gateway transit.

Batasan berikut ada untuk klaster DB dienkripsi dengan Amazon Aurora:

- Anda tidak dapat menonaktifkan enkripsi di klaster DB terenkripsi.
- Anda tidak dapat membuat snapshot terenkripsi dari klaster DB tidak terenkripsi.
- Snapshot klaster DB terenkripsi harus dienkripsi menggunakan kunci KMS yang sama seperti klaster DB.

- Anda tidak dapat mengonversi klaster DB yang tidak terenkripsi ke klaster terenkripsi. Namun, Anda dapat memulihkan snapshot tidak terenkripsi ke klaster DB Aurora terenkripsi. Untuk melakukannya, tentukan kunci KMS saat Anda memulihkan dari snapshot yang tidak terenkripsi.
- Anda tidak dapat membuat Aurora Replica terenkripsi dari klaster DB Aurora yang tidak dienkripsi. Anda tidak dapat membuat Aurora Replicas yang tidak terenkripsi untuk klaster DB Aurora terenkripsi.
- Untuk menyalin snapshot terenkripsi dari satu AWS Wilayah ke wilayah lain, Anda harus menentukan kunci KMS di Wilayah tujuan. AWS Ini karena kunci KMS khusus untuk AWS Wilayah tempat mereka dibuat.

  Snapshot sumber tetap terenkripsi selama proses penyalinan. Amazon Aurora menggunakan enkripsi amplop untuk melindungi data selama proses penyalinan. Untuk informasi selengkapnya tentang enkripsi amplop, lihat Enkripsi amplop dalam Panduan Developer AWS Key Management Service .
- Anda tidak dapat menghilangkan enkripsi klaster DB yang terenkripsi. Namun, Anda dapat mengekspor data dari klaster DB terenkripsi dan mengimpor data ke klaster DB tidak terenkripsi.

Manajemen AWS KMS key

Amazon Aurora terintegrasi secara otomatis dengan AWS Key Management Service (AWS KMS) untuk manajemen kunci. Amazon Aurora menggunakan enkripsi amplop. Untuk informasi selengkapnya tentang enkripsi amplop, lihat Enkripsi amplop dalam Panduan Developer AWS Key Management Service.

Anda dapat menggunakan dua jenis kunci AWS KMS untuk mengenkripsi klaster DB Anda.

- Jika Anda menginginkan kontrol penuh atas kunci KMS, Anda harus membuat kunci yang dikelola pelanggan. Untuk informasi selengkapnya tentang kunci yang dikelola pelanggan, lihat Kunci yang dikelola pelanggan di Panduan Developer AWS Key Management Service.

  Anda tidak dapat berbagi snapshot yang telah dienkripsi menggunakan Kunci yang dikelola AWS dari akun AWS yang berbagi snapshot tersebut.
- Kunci yang dikelola AWS adalah kunci KMS di akun Anda yang dibuat, dikelola, dan digunakan atas nama Anda oleh layanan AWS yang terintegrasi dengan AWS KMS. Secara default, RDS Kunci yang dikelola AWS (`aws/rds`) digunakan untuk enkripsi. Anda tidak dapat mengelola, merotasi, atau menghapus RDS Kunci yang dikelola AWS. Untuk informasi selengkapnya tentang

Kunci yang dikelola AWS, lihat Kunci yang dikelola AWS di Panduan Developer AWS Key Management Service.

Untuk mengelola kunci KMS yang digunakan untuk klaster DB terenkripsi Amazon Aurora, gunakan AWS Key Management Service (AWS KMS) di AWS KMS konsol, AWS CLI, atau AWS KMS API. Untuk melihat log Audit setiap tindakan yang diambil dengan kunci yang dikelola pelanggan atau AWS, gunakan AWS CloudTrail. Untuk informasi selengkapnya tentang rotasi kunci, lihat Merotasi kunci AWS KMS.

> **Important**
>
> Jika Anda menonaktifkan atau mencabut izin ke kunci KMS yang digunakan oleh basis data RDS, RDS akan menempatkan basis data Anda ke status terminal saat akses ke kunci KMS diperlukan. Perubahan ini bisa langsung, atau ditangguhkan, tergantung pada kasus penggunaan yang memerlukan akses ke kunci KMS. Dalam kondisi ini, kluster DB tidak lagi tersedia, dan status basis data saat ini tidak dapat dipulihkan. Untuk memulihkan kluster DB, Anda harus mengaktifkan kembali akses ke kunci KMS untuk RDS, lalu mengembalikan kluster DB dari cadangan terbaru yang tersedia.

Mengizinkan penggunaan kunci yang dikelola pelanggan

Ketika Aurora menggunakan kunci yang dikelola pelanggan dalam operasi kriptografi, itu bertindak atas nama pengguna yang membuat atau mengubah sumber daya Aurora.

Untuk membuat sumber daya Aurora menggunakan kunci yang dikelola pelanggan, pengguna harus memiliki izin untuk memanggil operasi berikut pada kunci tersebut:

- kms:CreateGrant
- kms:DescribeKey

Anda dapat menentukan izin yang diperlukan ini dalam kebijakan kunci, atau dalam kebijakan IAM jika kebijakan kunci memungkinkan hal tersebut.

Anda dapat membuat kebijakan IAM lebih ketat dalam berbagai cara. Misalnya, jika Anda ingin mengizinkan kunci yang dikelola pelanggan hanya digunakan untuk permintaan yang berasal dari Aurora, Anda dapat menggunakan kunci kondisi kms:ViaService dengan nilai `rds.<region>.amazonaws.com`. Selain itu, Anda dapat menggunakan kunci atau nilai dalam

Konteks enkripsi Amazon RDS sebagai syarat untuk menggunakan kunci yang dikelola pelanggan untuk enkripsi.

Untuk informasi selengkapnya, lihat Mengizinkan pengguna di akun lain untuk menggunakan kunci KMS di Panduan Developer AWS Key Management Service dan Kebijakan kunci di AWS KMS.

Konteks enkripsi Amazon RDS

Ketika Aurora menggunakan kunci KMS Anda, atau ketika Amazon EBS menggunakan kunci KMS atas nama Aurora, layanan akan menentukan konteks enkripsi. Konteks enkripsi adalah data terautentikasi tambahan (AAD) yang digunakan AWS KMS untuk memastikan integritas data. Ketika konteks enkripsi ditentukan untuk operasi enkripsi, layanan harus menentukan konteks enkripsi yang sama untuk operasi dekripsi. Jika tidak, dekripsi akan gagal. Konteks enkripsi juga ditulis ke log AWS CloudTrail untuk membantu Anda memahami mengapa kunci KMS tertentu digunakan. Log CloudTrail Anda mungkin berisi banyak entri yang menjelaskan penggunaan kunci KMS, tetapi konteks enkripsi di setiap entri log dapat membantu Anda menentukan alasan penggunaan tertentu tersebut.

Minimal, Aurora selalu menggunakan ID instans DB untuk konteks enkripsi, seperti dalam contoh format JSON berikut:

```
{ "aws:rds:db-id": "db-CQYSMDPBRZ7BPMH7Y3RTDG5QY" }
```

Konteks enkripsi ini dapat membantu Anda mengidentifikasi instans DB yang menggunakan kunci KMS Anda.

Ketika kunci KMS Anda digunakan untuk instans DB tertentu dan volume Amazon EBS tertentu, baik ID instans DB dan ID volume Amazon EBS digunakan untuk konteks enkripsi, seperti dalam contoh format JSON berikut:

```
{
  "aws:rds:db-id": "db-BRG7VYS3SVIFQW7234EJQOM5RQ",
  "aws:ebs:id": "vol-ad8c6542"
}
```

Anda dapat menggunakan Lapisan Soket Aman (SSL) atau Keamanan Lapisan Pengangkutan (TLS) dari aplikasi Anda untuk mengenkripsi koneksi ke klaster DB yang menjalankan Aurora MySQL atau Aurora PostgreSQL.

Secara opsional, koneksi SSL/TLS Anda dapat melakukan verifikasi identitas server dengan memvalidasi sertifikat server yang diinstal pada database Anda. Untuk meminta verifikasi identitas server, ikuti proses umum ini:

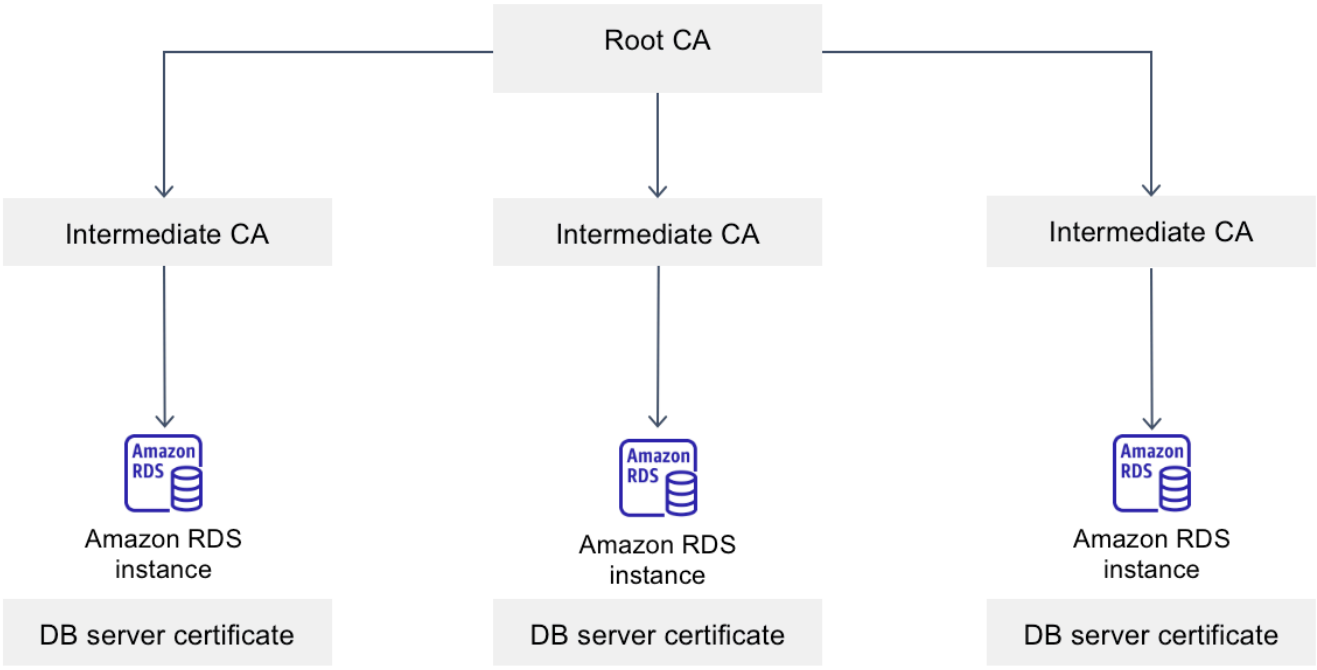
1. Pilih Otoritas Sertifikat (CA) yang menandatangani sertifikat server DB, untuk basis data Anda. Untuk informasi selengkapnya tentang otoritas sertifikat, lihat Otoritas sertifikat.
2. Unduh paket sertifikat yang akan digunakan saat Anda terhubung ke basis data. Untuk mengunduh paket sertifikat, lihat Bundel sertifikat untuk semua Wilayah AWS dan Bundel sertifikat untuk spesifik Wilayah AWS.

   > **Note**
   >
   > Semua sertifikat hanya tersedia untuk diunduh menggunakan koneksi SSL/TLS.

3. Hubungkan ke basis data menggunakan proses mesin DB Anda untuk menerapkan koneksi SSL/TLS. Setiap mesin DB memiliki prosesnya sendiri untuk menerapkan SSL/TLS. Untuk mempelajari cara menerapkan SSL/TLS untuk basis data Anda, ikuti tautan yang sesuai dengan mesin DB Anda:
   - Keamanan dengan Amazon Aurora MySQL
   - Keamanan dengan Amazon Aurora PostgreSQL

Otoritas sertifikat

Otoritas Sertifikat (CA) adalah sertifikat yang mengidentifikasi CA root di bagian atas rantai sertifikat. CA menandatangani sertifikat server DB, yang diinstal pada setiap instans DB. Sertifikat server DB mengidentifikasi instans DB sebagai server tepercaya.



Amazon RDS menyediakan CA berikut untuk menandatangani sertifikat server DB untuk database.

Otoritas sertifikat (CA)	Deskripsi
rds-ca-2019	Menggunakan otoritas sertifikat dengan algoritma kunci privat RSA 2048 dan algoritma penandata nganan SHA256. CA ini kedaluwarsa pada tahun 2024 dan tidak mendukung rotasi sertifikat server otomatis. Jika Anda menggunakan CA ini dan ingin mempertahankan standar yang sama, kami sarankan Anda beralih ke CA rds-ca-rsa 2048-g1.
rds-ca-rsa2048-g1	Menggunakan otoritas sertifikat dengan algoritma kunci privat RSA 2048 dan algoritma penandata nganan SHA256 di sebagian besar Wilayah AWS. Dalam AWS GovCloud (US) Regions, CA ini menggunakan otoritas sertifikat dengan algoritma kunci pribadi RSA 2048 dan algoritma penandata nganan SHA384.

Otoritas sertifikat (CA)	Deskripsi
	CA ini tetap berlaku lebih lama dari CA rds-ca-2019. CA ini mendukung rotasi sertifikat server otomatis.
rds-ca-rsa4096-g1	Menggunakan otoritas sertifikat dengan algoritma kunci privat RSA 4096 dan algoritma penandata nganan SHA384. CA ini mendukung rotasi sertifikat server otomatis.
rds-ca-ecc384-g1	Menggunakan otoritas sertifikat dengan algoritma kunci privat ECC 384 dan algoritma penandatanganan SHA384. CA ini mendukung rotasi sertifikat server otomatis.

> **Note**
>
> Jika Anda menggunakan AWS CLI, Anda dapat melihat validitas otoritas sertifikat yang tercantum di atas dengan menggunakan deskripsi-sertifikat.

Sertifikat CA ini termasuk dalam paket sertifikat regional dan global. Bila Anda menggunakan rds-ca-rsa 2048-g1, rds-ca-rsa 4096-g1, atau rds-ca-ecc 384-g1 CA dengan database, RDS mengelola sertifikat server DB pada database. RDS merotasi sertifikat server DB secara otomatis sebelum sertifikat ini kedaluwarsa.

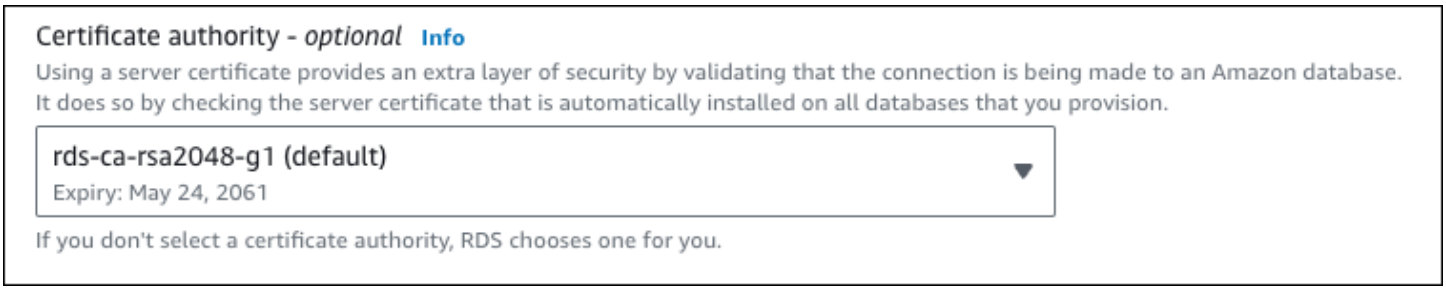
Mengatur CA untuk basis data Anda

Anda dapat mengatur CA untuk basis data saat Anda melakukan tugas berikut:

- Buat cluster Aurora DB — Anda dapat mengatur CA untuk instans DB di cluster Aurora saat Anda membuat instans DB pertama di cluster DB menggunakan atau RDS API. AWS CLI Saat ini, Anda tidak dapat mengatur CA untuk instans DB di klaster DB saat Anda membuat klaster DB menggunakan konsol RDS. Untuk petunjuk, lihat Membuat klaster DB Amazon Aurora.
- Memodifikasi instans DB – Anda dapat mengatur CA untuk instans DB di klaster DB dengan memodifikasinya. Untuk petunjuk, lihat Memodifikasi instans DB dalam klaster DB.

> **Note**
>
> CA default diatur ke rds-ca-rsa 2048-g1. Anda dapat mengganti CA default untuk Anda Akun AWS dengan menggunakan perintah modify-certificate.

CA yang tersedia bergantung pada mesin DB dan versi mesin DB. Jika menggunakan AWS Management Console, Anda dapat memilih CA menggunakan pengaturan Otoritas sertifikat, seperti yang ditampilkan pada gambar berikut.



Konsol hanya menampilkan CA yang tersedia untuk mesin DB dan versi mesin DB. Jika Anda menggunakan AWS CLI, Anda dapat mengatur CA untuk instans DB menggunakan modify-db-instanceperintah create-db-instanceor.

Jika Anda menggunakan AWS CLI, Anda dapat melihat CA yang tersedia untuk akun Anda dengan menggunakan perintah deskripsi-sertifikat. Perintah ini juga menunjukkan tanggal kedaluwarsa untuk setiap CA di `ValidTill` dalam output. Anda dapat menemukan CA yang tersedia untuk mesin DB tertentu dan versi mesin DB menggunakan describe-db-engine-versionsperintah.

Contoh berikut menunjukkan CA yang tersedia untuk versi mesin DB RDS for PostgreSQL default.

```
aws rds describe-db-engine-versions --default-only --engine postgres
```

Output Anda akan seperti yang berikut ini. CA yang tersedia tercantum di `SupportedCACertificateIdentifiers`. Output ini juga menunjukkan apakah versi mesin DB mendukung rotasi sertifikat tanpa pengaktifan ulang di `SupportsCertificateRotationWithoutRestart`.

```
{
    "DBEngineVersions": [
        {
            "Engine": "postgres",
```

```
            "MajorEngineVersion": "13",
            "EngineVersion": "13.4",
            "DBParameterGroupFamily": "postgres13",
            "DBEngineDescription": "PostgreSQL",
            "DBEngineVersionDescription": "PostgreSQL 13.4-R1",
            "ValidUpgradeTarget": [],
            "SupportsLogExportsToCloudwatchLogs": false,
            "SupportsReadReplica": true,
            "SupportedFeatureNames": [
                "Lambda"
            ],
            "Status": "available",
            "SupportsParallelQuery": false,
            "SupportsGlobalDatabases": false,
            "SupportsBabelfish": false,
            "SupportsCertificateRotationWithoutRestart": true,
            "SupportedCACertificateIdentifiers": [
                "rds-ca-2019",
                "rds-ca-rsa2048-g1",
                "rds-ca-ecc384-g1",
                "rds-ca-rsa4096-g1"
            ]
        }
    ]
}
```

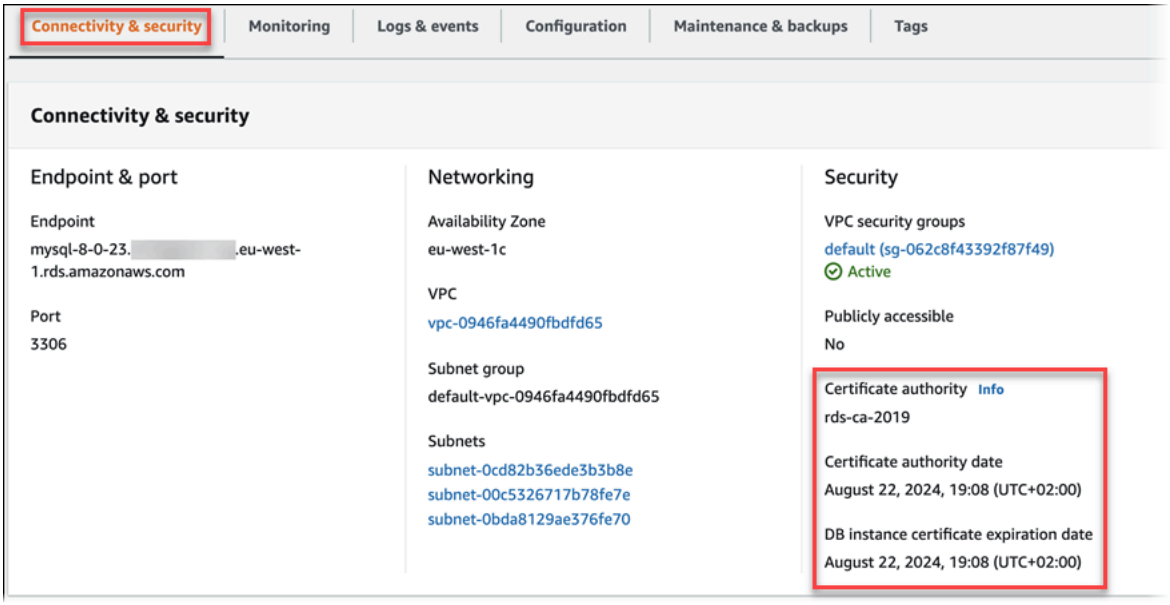
Validitas sertifikat server DB

Validitas sertifikat server DB bergantung pada mesin DB dan versi mesin DB. Jika versi mesin DB mendukung rotasi sertifikat tanpa pengaktifan ulang, validitas sertifikat server DB adalah 1 tahun. Jika tidak, validitasnya adalah 3 tahun.

Untuk informasi selengkapnya tentang rotasi sertifikat server DB, lihat Rotasi sertifikat server otomatis.

Melihat CA untuk instans DB Anda

Anda dapat melihat detail tentang CA untuk database dengan melihat tab Konektivitas & keamanan di konsol, seperti pada gambar berikut.



Jika Anda menggunakan AWS CLI, Anda dapat melihat detail tentang CA untuk instans DB dengan menggunakan describe-db-instancesperintah.

Untuk memeriksa konten paket sertifikat CA Anda, gunakan perintah berikut:

```
keytool -printcert -v -file global-bundle.pem
```

Bundel sertifikat untuk semua Wilayah AWS

Untuk mendapatkan bundel sertifikat yang berisi sertifikat perantara dan root untuk semua Wilayah AWS, unduh dari https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem.

Jika aplikasi Anda ada di Micro Windows dan memerlukan file PKCS7, Anda dapat mengunduh paket sertifikat PKCS7. Paket ini berisi sertifikat perantara dan root di https://truststore.pki.rds.amazonaws.com/global/global-bundle.p7b.

> **Note**
>
> Amazon RDS Proxy dan Aurora Serverless v1 penggunaan sertifikat dari AWS Certificate Manager (ACM). Jika Anda menggunakan RDS Proxy, Anda tidak perlu mengunduh sertifikat Amazon RDS atau memperbarui aplikasi yang menggunakan koneksi Proxy RDS. Untuk informasi selengkapnya, lihat Penggunaan TLS/SSL dengan Proksi RDS.

Jika Anda menggunakanAurora Serverless v1, mengunduh sertifikat Amazon RDS tidak diperlukan. Untuk informasi selengkapnya, lihat Menggunakan TLS/SSL dengan Aurora Serverless v1.

Bundel sertifikat untuk spesifik Wilayah AWS

Untuk mendapatkan bundel sertifikat yang berisi sertifikat perantara dan root untuk Wilayah AWS, unduh dari tautan untuk tabel berikut. Wilayah AWS

AWS Wilayah	Paket sertifikat (PEM)	Paket sertifikat (PKCS7)
AS Timur (Virginia Utara)	us-east-1-bundle.pem	us-east-1-bundle.p7b
AS Timur (Ohio)	us-east-2-bundle.pem	us-east-2-bundle.p7b
AS Barat (California Utara)	us-west-1-bundle.pem	us-west-1-bundle.p7b
AS Barat (Oregon)	us-west-2-bundle.pem	us-west-2-bundle.p7b
Afrika (Cape Town)	af-south-1-bundle.pem	af-south-1-bundle.p7b
Asia Pasifik (Hong Kong)	ap-east-1-bundle.pem	ap-east-1-bundle.p7b
Asia Pasifik (Hyderabad)	ap-south-2-bundle.pem	ap-south-2-bundle.p7b
Asia Pasifik (Jakarta)	ap-southeast-3-bundle.pem	ap-southeast-3-bundle.p7b
Asia Pasifik (Melbourne)	ap-southeast-4-bundle.pem	ap-southeast-4-bundle.p7b
Asia Pasifik (Mumbai)	ap-south-1-bundle.pem	ap-south-1-bundle.p7b
Asia Pasifik (Osaka)	ap-northeast-3-bundle.pem	ap-northeast-3-bundle.p7b
Asia Pasifik (Tokyo)	ap-northeast-1-bundle.pem	ap-northeast-1-bundle.p7b
Asia Pasifik (Seoul)	ap-northeast-2-bundle.pem	ap-northeast-2-bundle.p7b
Asia Pasifik (Singapura)	ap-southeast-1-bundle.pem	ap-southeast-1-bundle.p7b
Asia Pasifik (Sydney)	ap-southeast-2-bundle.pem	ap-southeast-2-bundle.p7b

AWS Wilayah	Paket sertifikat (PEM)	Paket sertifikat (PKCS7)
Kanada (Pusat)	ca-central-1-bundle.pem	ca-central-1-bundle.p7b
Kanada Barat (Calgary)	ca-barat-1-bundle.pem	ca-barat-1-bundle.p7b
Eropa (Frankfurt)	eu-central-1-bundle.pem	eu-central-1-bundle.p7b
Eropa (Irlandia)	eu-west-1-bundle.pem	eu-west-1-bundle.p7b
Eropa (London)	eu-west-2-bundle.pem	eu-west-2-bundle.p7b
Eropa (Milan)	eu-south-1-bundle.pem	eu-south-1-bundle.p7b
Eropa (Paris)	eu-west-3-bundle.pem	eu-west-3-bundle.p7b
Eropa (Spanyol)	eu-south-2-bundle.pem	eu-south-2-bundle.p7b
Eropa (Stockholm)	eu-north-1-bundle.pem	eu-north-1-bundle.p7b
Eropa (Zürich)	eu-central-2-bundle.pem	eu-central-2-bundle.p7b
Israel (Tel Aviv)	il-central-1-bundle.pem	il-central-1-bundle.p7b
Timur Tengah (Bahrain)	me-south-1-bundle.pem	me-south-1-bundle.p7b
Timur Tengah (UEA)	me-central-1-bundle.pem	me-central-1-bundle.p7b
Amerika Selatan (Sao Paulo)	sa-east-1-bundle.pem	sa-east-1-bundle.p7b

Sertifikat AWS GovCloud (US)

Untuk mendapatkan bundel sertifikat yang berisi sertifikat perantara dan root untuk AWS GovCloud (US) Region s, unduh dari https://truststore.pki. us-gov-west-1.rds.amazonaws.com/global/global-bundle.pem.

Jika aplikasi Anda berada di Microsoft Windows dan memerlukan file PKCS7, Anda dapat mengunduh paket sertifikat PKCS7. Bundel ini berisi sertifikat perantara dan root di https://truststore.pki. us-gov-west-1.rds.amazonaws.com/global/global-bundle.p7b.

Untuk mendapatkan bundel sertifikat yang berisi sertifikat perantara dan root untuk AWS GovCloud (US) Region, unduh dari tautan untuk tabel berikut. AWS GovCloud (US) Region

AWS GovCloud (US) Region	Paket sertifikat (PEM)	Paket sertifikat (PKCS7)
AWS GovCloud (AS-Timur)	us-gov-east-1-bundel.pem	us-gov-east-1-bundel.p7b
AWS GovCloud (AS-Barat)	us-gov-west-1-bundel.pem	us-gov-west-1-bundel.p7b

Merotasi sertifikat SSL/TLS

Sertifikat Otoritas Sertifikat Amazon RDS rds-ca-2019 akan kedaluwarsa pada Agustus 2024. Jika Anda menggunakan atau berencana untuk menggunakan Secure Sockets Layer (SSL) atau Transport Layer Security (TLS) dengan verifikasi sertifikat untuk terhubung ke instans RDS DB , pertimbangkan untuk menggunakan salah satu sertifikat rds-ca-rsa CA baru 2048-g1, 4096-g1 atau 384-g1. rds-ca-rsa rds-ca-ecc Jika saat ini Anda tidak menggunakan SSL/TLS dengan verifikasi sertifikat, Anda mungkin masih memiliki sertifikat CA yang kedaluwarsa dan harus memperbaruinya ke sertifikat CA baru jika Anda berencana untuk menggunakan SSL/TLS dengan verifikasi sertifikat untuk terhubung ke basis data RDS Anda.

Ikuti petunjuk ini untuk menyelesaikan pembaruan Anda. Sebelum memperbarui instans DB untuk menggunakan sertifikat CA baru, pastikan Anda memperbarui klien atau aplikasi yang terhubung ke database RDS Anda.

Amazon RDS menyediakan sertifikat CA baru sebagai praktik terbaik AWS keamanan. Untuk informasi tentang sertifikat baru dan AWS Wilayah yang didukung, lihat.

> **Note**
>
> Amazon RDS Proxy dan Aurora Serverless v1 penggunaan sertifikat dari AWS Certificate Manager (ACM). Jika Anda menggunakan RDS Proxy, ketika Anda memutar sertifikat SSL/TLS Anda, Anda tidak perlu memperbarui aplikasi yang menggunakan koneksi Proxy RDS. Untuk informasi selengkapnya, lihat Penggunaan TLS/SSL dengan Proksi RDS.
> Jika Anda menggunakanAurora Serverless v1, mengunduh sertifikat Amazon RDS tidak diperlukan. Untuk informasi selengkapnya, lihat Menggunakan TLS/SSL dengan Aurora Serverless v1.

> **Note**
>
> Jika Anda menggunakan aplikasi Go versi 1.15 dengan instans DB yang dibuat atau diperbarui ke sertifikat rds-ca-2019 sebelum 28 Juli 2020, Anda harus memperbarui sertifikat lagi. Jalankan `modify-db-instance` perintah , menggunakan pengidentifikasi sertifikat CA baru. Anda dapat menemukan CA yang tersedia untuk mesin DB dan versi mesin DB tertentu menggunakan perintah `describe-db-engine-versions`.
> Jika Anda membuat database atau memperbarui sertifikatnya setelah 28 Juli 2020, tidak ada tindakan yang diperlukan. Untuk informasi selengkapnya, lihat Go GitHub issue #39568.

Topik

- Memperbarui sertifikat CA Anda dengan memodifikasi instans cluster DB
- Memperbarui sertifikat CA Anda dengan menerapkan pemeliharaan
- Rotasi sertifikat server otomatis
- Contoh skrip untuk mengimpor sertifikat ke trust store Anda

Memperbarui sertifikat CA Anda dengan memodifikasi instans cluster DB

Contoh berikut memperbarui sertifikat CA Anda dari rds-ca-2019 ke 2048-g1. rds-ca-rsa Anda dapat memilih sertifikat yang berbeda. Untuk informasi selengkapnya, lihat Otoritas sertifikat.

Untuk memperbarui sertifikat CA Anda dengan memodifikasi instans cluster DB

1. Unduh sertifikat SSL/TLS yang baru seperti yang dijelaskan dalam .
2. Perbarui aplikasi Anda untuk menggunakan sertifikat SSL/TLS baru.

   Metode untuk memperbarui aplikasi untuk sertifikat SSL/TLS baru bergantung pada aplikasi spesifik Anda. Bekerja samalah dengan developer aplikasi Anda untuk memperbarui sertifikat SSL/TLS untuk aplikasi Anda.

   Untuk informasi tentang pemeriksaan koneksi SSL/TLS dan pembaruan aplikasi untuk setiap mesin DB, lihat topik berikut:

   - Memperbarui aplikasi untuk terhubung ke klaster DB Aurora MySQL menggunakan sertifikat TLS baru

- Memperbarui aplikasi untuk terhubung ke klaster DB Aurora PostgreSQL menggunakan sertifikat SSL/TLS baru

Untuk contoh skrip yang memperbarui trust store untuk sistem operasi Linux, lihat Contoh skrip untuk mengimpor sertifikat ke trust store Anda.

> **Note**
>
> Paket sertifikat berisi sertifikat untuk CA lama dan baru, sehingga Anda dapat meningkatkan aplikasi Anda dengan aman dan mempertahankan konektivitas selama periode transisi. Jika Anda menggunakan AWS Database Migration Service untuk memigrasikan database ke cluster, sebaiknya gunakan bundel sertifikat untuk memastikan konektivitas selama migrasi.

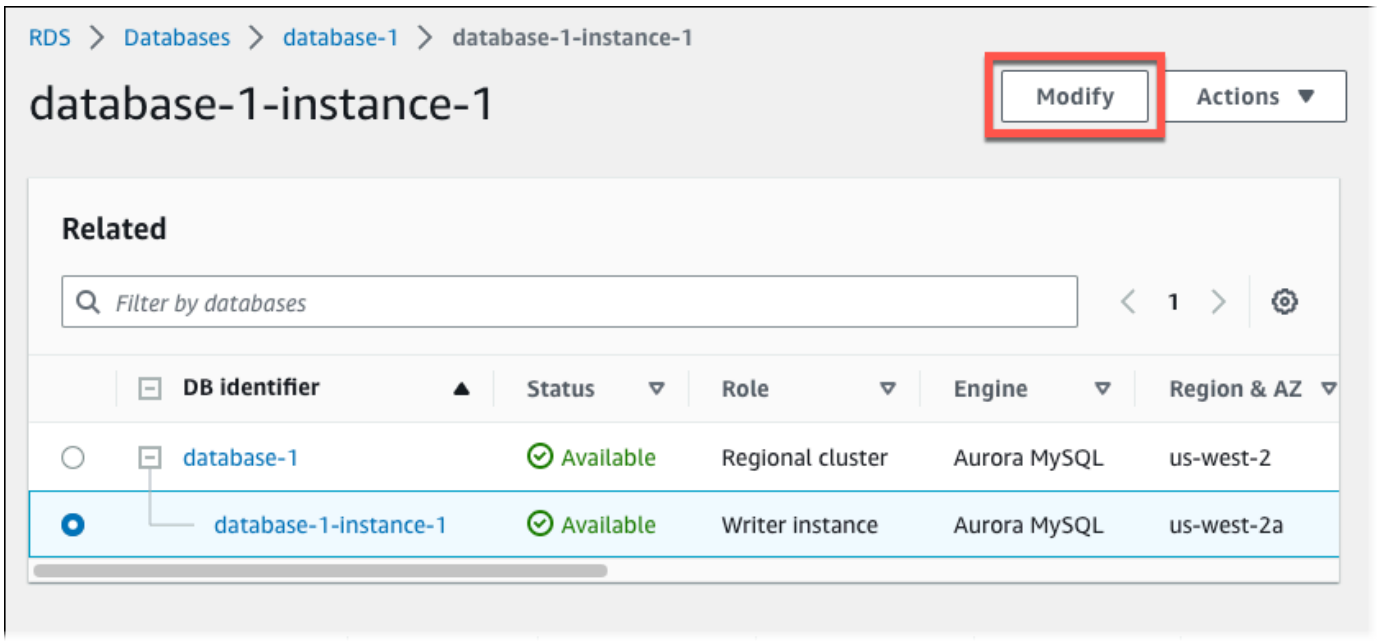
3. Ubah instans DB untuk mengubah CA dari rds-ca-2019 menjadi 2048-g1. rds-ca-rsa Untuk memeriksa apakah database Anda memerlukan restart untuk memperbarui sertifikat CA, gunakan describe-db-engine-versionsperintah dan periksa `SupportsCertificateRotationWithoutRestart` bendera.

> **Note**
>
> Boot ulang klaster Babelfish Anda setelah memodifikasi untuk memperbarui sertifikat CA.

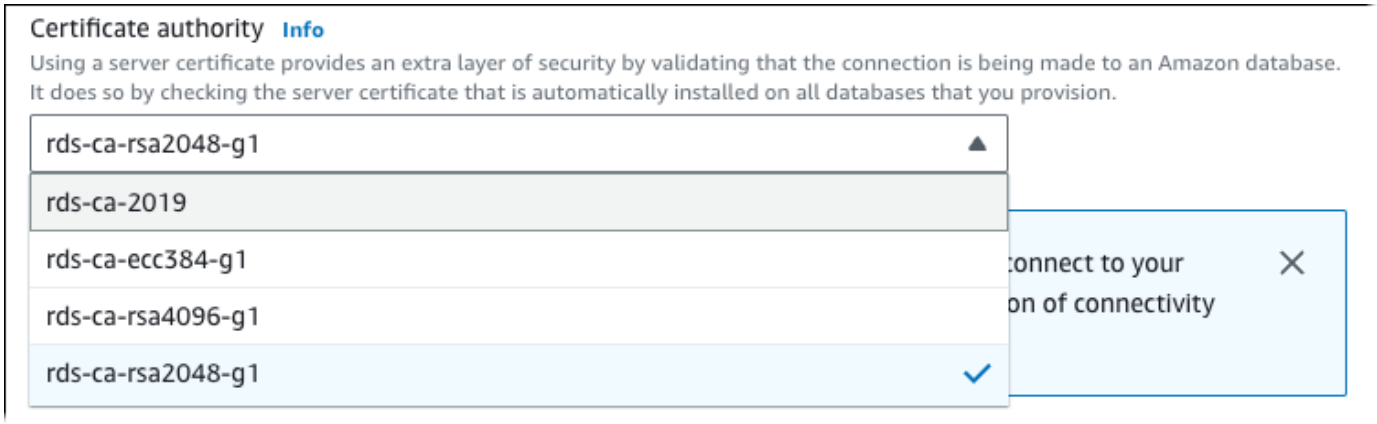
> **Important**
>
> Jika Anda mengalami masalah konektivitas setelah masa berlaku sertifikat berakhir, gunakan opsi terapkan segera dengan menentukan Terapkan segera di konsol atau dengan menentukan opsi `--apply-immediately` menggunakan AWS CLI. Secara default, operasi ini dijadwalkan untuk berjalan selama periode pemeliharaan berikutnya. Untuk mengatur penggantian CA instans Anda yang berbeda dari CA RDS default, gunakan perintah CLI modify-certificates.

Konsol

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Databases, lalu pilih instans DB yang ingin Anda modifikasi.
3. Pilih Ubah.



4. Di bagian Konektivitas, pilih rds-ca-rsa2048-g1.



5. Pilih Lanjutkan dan periksa ringkasan modifikasi.
6. Untuk segera menerapkan perubahan, pilih Terapkan segera.
7. Di halaman konfirmasi, tinjau perubahan Anda. Jika benar, pilih Modify DB Instance untuk menyimpan perubahan Anda.

> **Important**
>
> Saat Anda menjadwalkan operasi ini, pastikan bahwa Anda telah memperbarui trust store sisi klien sebelumnya.

Atau pilih Kembali untuk mengedit perubahan atau Batalkan untuk membatalkan perubahan Anda.

AWS CLI

Untuk menggunakan AWS CLI untuk mengubah CA dari rds-ca-2019 ke rds-ca-rsa2048-g1 untuk instans DB atau cluster DB multi-AZ, panggil perintah . modify-db-instancemodify-db-cluster Tentukan instans DB dan `--ca-certificate-identifier` opsi.

Gunakan `--apply-immediately` parameter untuk segera menerapkan pembaruan. Secara default, operasi ini dijadwalkan untuk berjalan selama periode pemeliharaan berikutnya.

> **Important**
>
> Saat Anda menjadwalkan operasi ini, pastikan bahwa Anda telah memperbarui trust store sisi klien sebelumnya.

Example

Contoh berikut memodifikasi `mydbinstance` dengan menyetel sertifikat CA `kerds-ca-rsa2048-g1`.

Untuk Linux, macOS, atau Unix:

```
aws rds modify-db-instance \
    --db-instance-identifier mydbinstance \
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

Untuk Windows:

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

```
    --db-instance-identifier mydbinstance ^
    --ca-certificate-identifier rds-ca-rsa2048-g1
```

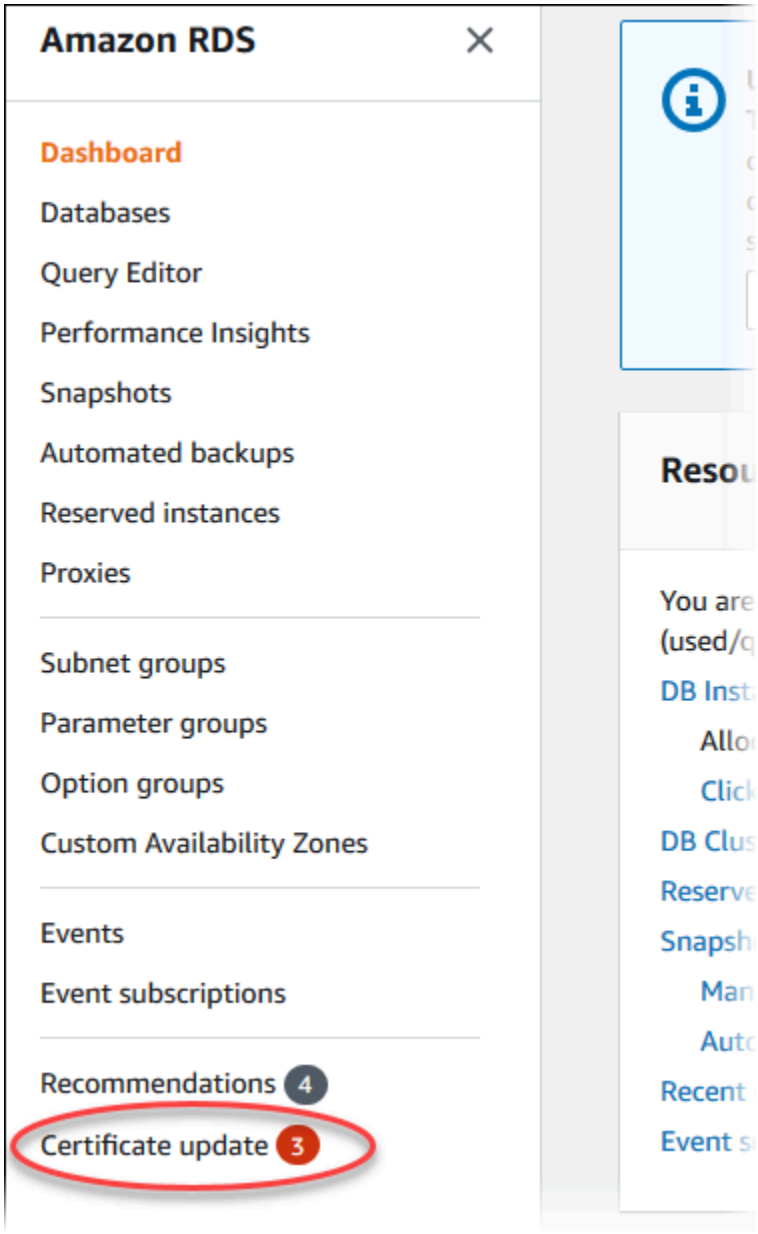
> **Note**
>
> Jika instance Anda memerlukan reboot, Anda dapat menggunakan perintah modify-db-instanceCLI dan menentukan opsi. `--no-certificate-rotation-restart`

Memperbarui sertifikat CA Anda dengan menerapkan pemeliharaan

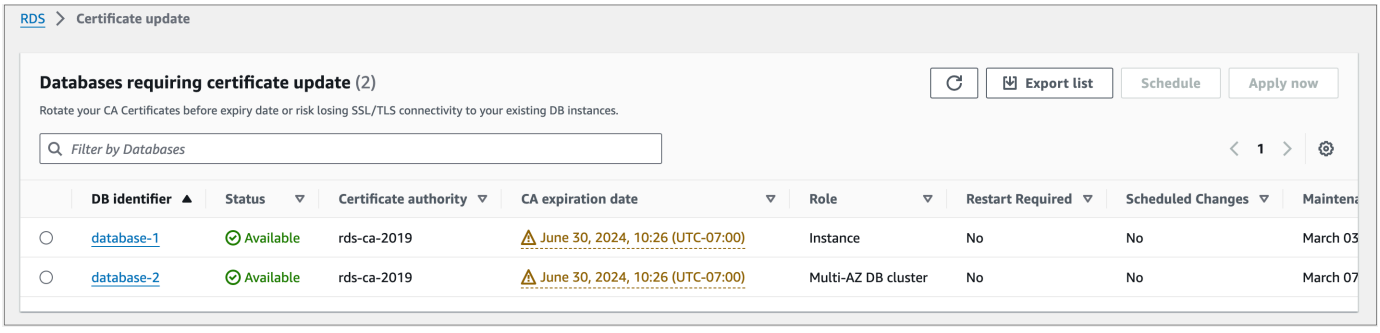
Lakukan langkah-langkah berikut untuk memperbarui sertifikat CA Anda dengan menerapkan pemeliharaan.

Untuk memperbarui sertifikat CA Anda dengan menerapkan pemeliharaan

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Pembaruan sertifikat.



Halaman Basis data yang memerlukan pembaruan sertifikat akan muncul.



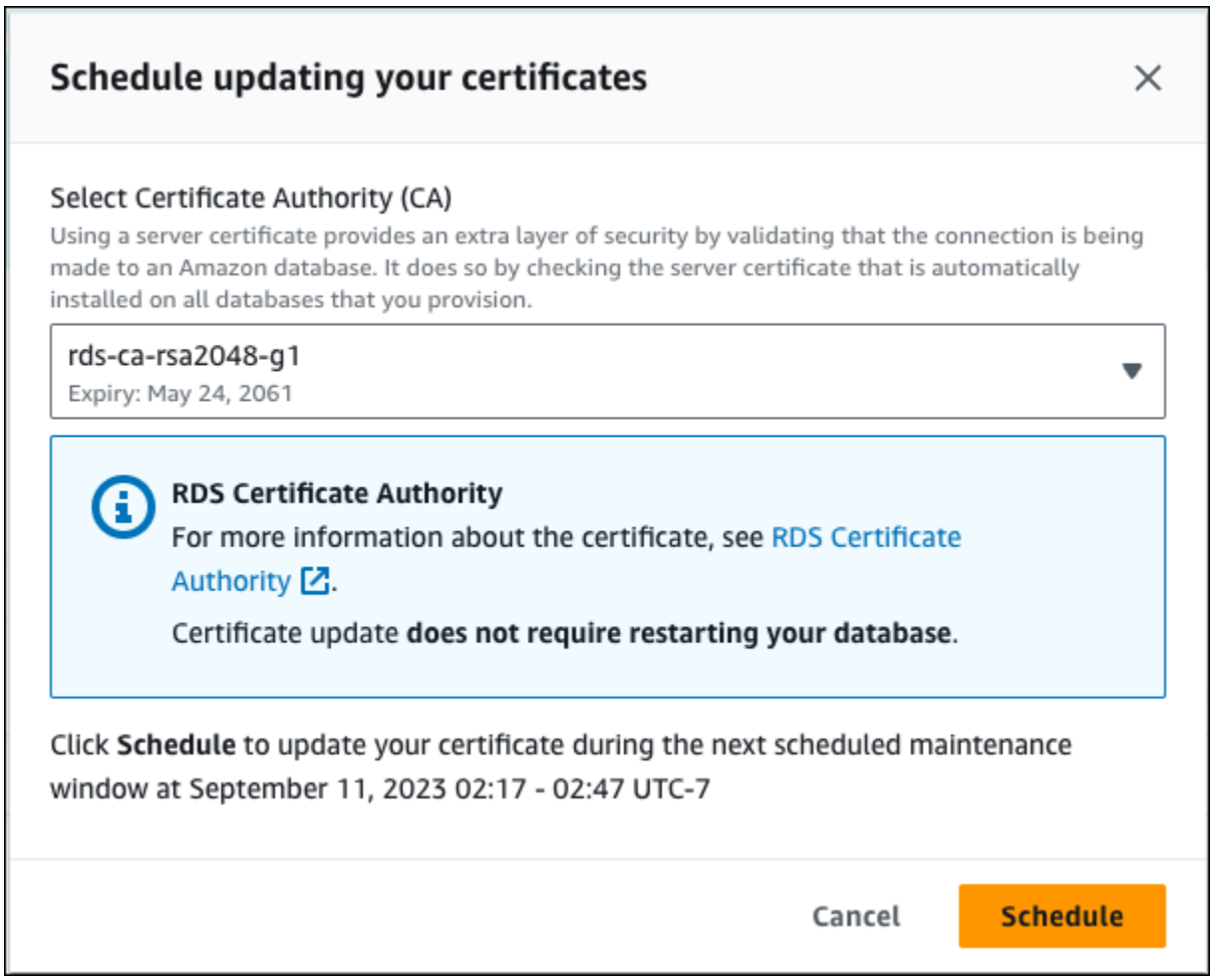
> **Note**
>
> Halaman ini hanya menampilkan instans DB untuk saat ini. Wilayah AWS Jika Anda memiliki database di lebih dari satu Wilayah AWS, periksa halaman ini di masing-masing Wilayah AWS untuk melihat semua instance DB dengan sertifikat SSL/TLS lama.

3. Pilih instans DB yang ingin Anda perbarui.

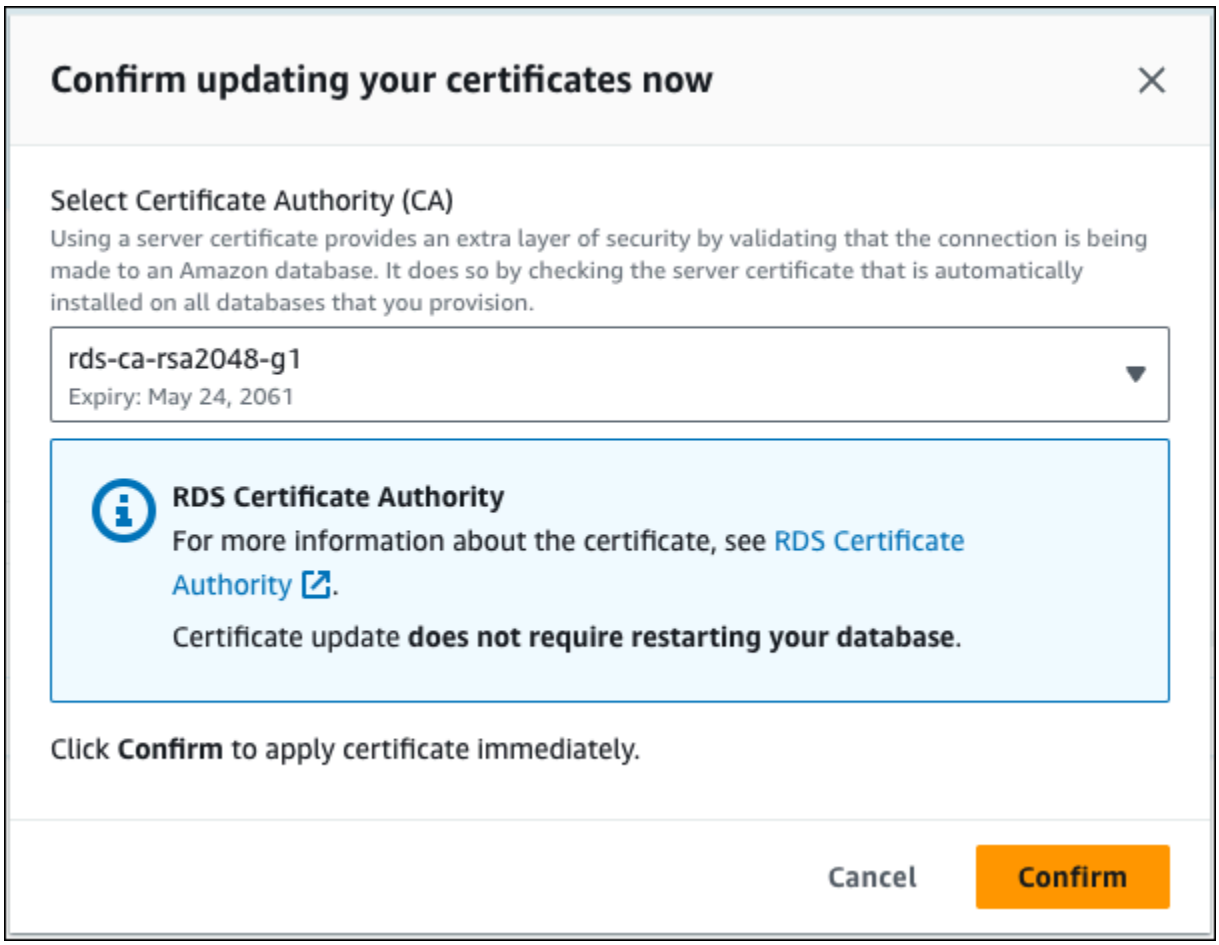
   Anda dapat menjadwalkan rotasi sertifikat untuk periode pemeliharaan berikutnya dengan memilih Jadwal. Segera terapkan rotasi dengan memilih Terapkan sekarang.

   > **Important**
   >
   > Jika Anda mengalami masalah konektivitas setelah sertifikat kedaluwarsa, gunakan opsi Terapkan sekarang.

4. a. Jika Anda memilih Jadwal, Anda akan diminta untuk mengonfirmasi rotasi sertifikat CA. Prompt ini juga menyatakan periode terjadwal untuk pembaruan Anda.



b. Jika Anda memilih Terapkan sekarang, Anda akan diminta untuk mengonfirmasi rotasi sertifikat CA.



> **Important**
>
> Sebelum menjadwalkan rotasi sertifikat CA di basis data Anda, perbarui aplikasi klien yang menggunakan SSL/TLS dan sertifikat server untuk terhubung. Pembaruan ini khusus untuk mesin DB Anda. Setelah Anda memperbarui aplikasi klien ini, Anda dapat mengonfirmasi rotasi sertifikat CA.

Untuk melanjutkan, pilih kotak centang, lalu pilih Konfirmasi.

5. Ulangi langkah 3 dan 4 untuk setiap instans DB yang ingin Anda perbarui.

Rotasi sertifikat server otomatis

Jika CA Anda mendukung rotasi sertifikat server otomatis, RDS secara otomatis menangani rotasi sertifikat server DB. RDS menggunakan CA root yang sama untuk rotasi otomatis ini, jadi Anda tidak perlu mengunduh paket CA baru. Lihat Otoritas sertifikat.

Rotasi dan validitas sertifikat server DB Anda bergantung pada mesin DB Anda:

- Jika mesin DB Anda mendukung rotasi tanpa pengaktifan ulang, RDS secara otomatis merotasi sertifikat server DB tanpa memerlukan tindakan apa pun dari Anda. RDS mencoba merotasi sertifikat server DB Anda dalam periode pemeliharaan yang Anda pilih di waktu paruh sertifikat server DB. Sertifikat server DB baru berlaku selama 12 bulan.
- Jika mesin DB Anda tidak mendukung rotasi tanpa pengaktifan ulang, RDS memberi tahu Anda tentang peristiwa pemeliharaan setidaknya 6 bulan sebelum sertifikat server DB kedaluwarsa. Sertifikat server DB baru berlaku selama 36 bulan.

Gunakan describe-db-engine-versionsperintah dan periksa `SupportsCertificateRotationWithoutRestart` bendera untuk mengidentifikasi apakah versi mesin DB mendukung memutar sertifikat tanpa memulai ulang. Untuk informasi selengkapnya, lihat Mengatur CA untuk basis data Anda.

Contoh skrip untuk mengimpor sertifikat ke trust store Anda

Berikut adalah contoh skrip shell yang mengimpor paket sertifikat ke trust store.

Setiap skrip shell menggunakan keytool, yang merupakan bagian dari Java Development Kit (JDK). Untuk informasi tentang cara menginstal JDK, lihat JDK Installation Guide.

Topik

- Contoh skrip untuk mengimpor sertifikat di Linux
- Contoh skrip untuk mengimpor sertifikat di macOS

Contoh skrip untuk mengimpor sertifikat di Linux

Berikut adalah contoh skrip shell yang mengimpor paket sertifikat ke trust store di sistem operasi Linux.

```
mydir=tmp/certs
```

```
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi

truststore=${mydir}/rds-truststore.jks
storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" >
 ${mydir}/global-bundle.pem
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/ {split_after=1}
{print > "rds-ca-" n+1 ".pem"}' < ${mydir}/global-bundle.pem

for CERT in rds-ca-*; do
  alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /Subject:/;
 s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -keystore
 ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias | cut
 -d " " -f3- | while read alias
do
   expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -alias
 "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

Contoh skrip untuk mengimpor sertifikat di macOS

Berikut adalah contoh skrip shell yang mengimpor paket sertifikat ke trust store di sistem operasi macOS.

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
```

```
then
mkdir -p "${mydir}"
fi

truststore=${mydir}/rds-truststore.jks
storepassword=changeit

curl -sS "https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem" >
 ${mydir}/global-bundle.pem
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/global-bundle.pem rds-ca-

for CERT in rds-ca-*; do
  alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /Subject:/;
 s/.*(CN=|CN = )//; print')
  echo "Importing $alias"
  keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -keystore
 ${truststore} -noprompt
  rm $CERT
done

rm ${mydir}/global-bundle.pem

echo "Trust store content is: "

keytool -list -v -keystore "$truststore" -storepass ${storepassword} | grep Alias | cut
 -d " " -f3- | while read alias
do
   expiry=`keytool -list -v -keystore "$truststore" -storepass ${storepassword} -alias
 "${alias}" | grep Valid | perl -ne 'if(/until: (.*?)\n/) { print "$1\n"; }'`
   echo " Certificate ${alias} expires in '$expiry'"
done
```

Privasi lalu lintas antarjaringan

Koneksi dilindungi antara Amazon Aurora dan aplikasi on-premise serta antara Amazon Aurora dan sumber daya AWS lainnya dengan Wilayah AWS yang sama.

Lalu lintas antara layanan dan aplikasi dan klien on-premise

Anda memiliki dua opsi konektivitas antara jaringan privat dan AWS:

- Koneksi AWS Site-to-Site VPN. Untuk informasi selengkapnya tentang ini, lihat Apa itu AWS Site-to-Site VPN?
- Koneksi AWS Direct Connect. Untuk informasi selengkapnya tentang ini, lihat Apa itu AWS Direct Connect?

Anda mendapatkan akses ke Amazon Aurora melalui jaringan dengan menggunakan operasi API yang dipublikasikan AWS. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Pengangkutan (TLS). Kami membutuhkan TLS 1.2 dan merekomendasikan TLS 1.3.
- Cipher cocok dengan perfect forward secrecy (PFS) seperti DHE (Ephemeral Diffie-Hellman) atau ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Sebagian besar sistem-sistem modern seperti Java 7 dan versi yang lebih baru mendukung mode-mode ini.

Selain itu, permintaan harus ditandatangani menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan pengguna utama IAM. Atau Anda dapat menggunakan AWS Security Token Service (AWS STS) untuk membuat kredensial keamanan sementara untuk menandatangani permintaan.

Manajemen identitas dan akses untuk Amazon Aurora

AWS Identity and Access Management (IAM) adalah Layanan AWS yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator IAM mengontrol siapa yang dapat diautentikasi (masuk) dan diotorisasi (memiliki izin) untuk menggunakan sumber daya Amazon RDS. IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik

- Audiens
- Mengautentikasi dengan identitas
- Mengelola akses menggunakan kebijakan
- Cara kerja Amazon Aurora dengan IAM
- Contoh kebijakan berbasis identitas untuk Amazon Aurora
- AWS kebijakan terkelola untuk Amazon RDS
- Amazon RDS memperbarui kebijakan AWS terkelola
- Mencegah masalah confused deputy lintas layanan
- Autentikasi basis data IAM
- Memecahkan masalah identitas dan akses Amazon Aurora

Audiens

Cara Anda menggunakan AWS Identity and Access Management (IAM) berbeda, tergantung pada pekerjaan yang Anda lakukan di Aurora.

Pengguna layanan – Jika Anda menggunakan layanan Aurora untuk melakukan pekerjaan, administrator Anda akan memberikan kredensial dan izin yang Anda butuhkan. Saat Anda menggunakan lebih banyak fitur Aurora untuk melakukan pekerjaan, Anda mungkin memerlukan izin tambahan. Memahami bagaimana cara mengelola akses dapat membantu Anda meminta izin yang tepat dari administrator Anda. Jika Anda tidak dapat mengakses fitur di Aurora, lihat Memecahkan masalah identitas dan akses Amazon Aurora.

Administrator layanan – Jika Anda bertanggung jawab atas sumber daya Aurora di perusahaan, Anda mungkin memiliki akses penuh ke Aurora. Tugas Anda adalah menentukan fitur Aurora dan sumber

daya mana yang dapat diakses karyawan Anda. Kemudian, Anda harus mengirim permintaan kepada administrator untuk mengubah izin pengguna layanan. Tinjau informasi di halaman ini untuk memahami konsep dasar IAM. Untuk mempelajari selengkapnya tentang bagaimana perusahaan Anda dapat menggunakan IAM dengan Aurora, lihat Cara kerja Amazon Aurora dengan IAM.

Administrator – Jika Anda adalah seorang administrator, Anda mungkin ingin mengetahui detail tentang cara menulis kebijakan untuk mengelola akses ke Aurora. Untuk melihat contoh kebijakan berbasis identitas Aurora yang dapat Anda gunakan di IAM, lihat Contoh kebijakan berbasis identitas untuk Amazon Aurora.

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensi identitas Anda. Anda harus diautentikasi (masuk ke AWS) sebagai Pengguna root akun AWS, sebagai pengguna IAM, atau dengan mengasumsikan peran IAM.

Anda dapat masuk AWS sebagai identitas federasi dengan menggunakan kredensil yang disediakan melalui sumber identitas. AWS IAM Identity Center Pengguna (IAM Identity Center), autentikasi masuk tunggal perusahaan Anda, dan kredensi Google atau Facebook Anda adalah contoh identitas federasi. Saat Anda masuk sebagai identitas gabungan, administrator Anda sebelumnya menyiapkan federasi identitas menggunakan peran IAM. Ketika Anda mengakses AWS dengan menggunakan federasi, Anda secara tidak langsung mengambil peran.

Bergantung pada jenis pengguna Anda, Anda dapat masuk ke AWS Management Console atau portal AWS akses. Untuk informasi selengkapnya tentang masuk AWS, lihat Cara masuk ke Panduan AWS Sign-In Pengguna Anda Akun AWS.

Jika Anda mengakses AWS secara terprogram, AWS sediakan kit pengembangan perangkat lunak (SDK) dan antarmuka baris perintah (CLI) untuk menandatangani permintaan Anda secara kriptografis dengan menggunakan kredensil Anda. Jika Anda tidak menggunakan AWS alat, Anda harus menandatangani permintaan sendiri. Untuk informasi selengkapnya tentang penggunaan metode yang disarankan untuk menandatangani permintaan sendiri, lihat Menandatangani permintaan AWS API di Panduan Pengguna IAM.

Apa pun metode autentikasi yang digunakan, Anda mungkin diminta untuk menyediakan informasi keamanan tambahan. Misalnya, AWS merekomendasikan agar Anda menggunakan otentikasi multi-faktor (MFA) untuk meningkatkan keamanan akun Anda. Untuk mempelajari selengkapnya, lihat Autentikasi multi-faktor dalam Panduan Pengguna AWS IAM Identity Center dan Menggunakan autentikasi multi-faktor (MFA) di AWS dalam Panduan Pengguna IAM.

AWS pengguna root akun

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya di akun. Identitas ini disebut pengguna Akun AWS root dan diakses dengan masuk dengan alamat email dan kata sandi yang Anda gunakan untuk membuat akun. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari Anda. Lindungi kredensial pengguna root Anda dan gunakan kredensial tersebut untuk melakukan tugas yang hanya dapat dilakukan pengguna root. Untuk daftar lengkap tugas yang mengharuskan Anda masuk sebagai pengguna root, lihat Tugas yang memerlukan kredensial pengguna root dalam Panduan Pengguna IAM.

Identitas terfederasi

Sebagai praktik terbaik, mewajibkan pengguna manusia, termasuk pengguna yang memerlukan akses administrator, untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS dengan menggunakan kredensi sementara.

Identitas federasi adalah pengguna dari direktori pengguna perusahaan Anda, penyedia identitas web, direktori Pusat Identitas AWS Directory Service, atau pengguna mana pun yang mengakses Layanan AWS dengan menggunakan kredensil yang disediakan melalui sumber identitas. Ketika identitas federasi mengakses Akun AWS, mereka mengambil peran, dan peran memberikan kredensi sementara.

Untuk manajemen akses terpusat, kami sarankan Anda menggunakan AWS IAM Identity Center. Anda dapat membuat pengguna dan grup di Pusat Identitas IAM, atau Anda dapat menghubungkan dan menyinkronkan ke sekumpulan pengguna dan grup di sumber identitas Anda sendiri untuk digunakan di semua aplikasi Akun AWS dan aplikasi Anda. Untuk informasi tentang Pusat Identitas IAM, lihat Apa itu Pusat Identitas IAM? dalam Panduan Pengguna AWS IAM Identity Center .

Pengguna dan grup IAM

Pengguna IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus untuk satu orang atau aplikasi. Jika memungkinkan, sebaiknya andalkan kredensial sementara daripada membuat pengguna IAM yang memiliki kredensial jangka panjang seperti kata sandi dan kunci akses. Namun, jika Anda memiliki kasus penggunaan khusus yang memerlukan kredensial jangka panjang dengan pengguna IAM, kami sarankan Anda merotasi kunci akses. Untuk informasi selengkapnya, lihat Rotasikan kunci akses secara rutin untuk kasus penggunaan yang memerlukan kredensial jangka panjang dalam Panduan Pengguna IAM.

Grup IAM adalah identitas yang menentukan kumpulan pengguna IAM. Anda tidak dapat masuk sebagai grup. Anda dapat menggunakan grup untuk menentukan izin untuk beberapa pengguna sekaligus. Grup membuat izin lebih mudah dikelola untuk sekelompok besar pengguna. Misalnya, Anda dapat memiliki grup yang bernama IAMAdmins dan memberikan izin kepada grup tersebut untuk mengelola sumber daya IAM.

Pengguna berbeda dari peran. Pengguna secara unik terkait dengan satu orang atau aplikasi, tetapi peran dimaksudkan untuk dapat digunakan oleh siapa pun yang membutuhkannya. Pengguna memiliki kredensial jangka panjang permanen, tetapi peran memberikan kredensial sementara. Untuk mempelajari selengkapnya, lihat Kapan harus membuat pengguna IAM (bukan peran) dalam Panduan Pengguna IAM.

Anda dapat mengautentikasi ke klaster DB Anda menggunakan autentikasi basis data IAM.

Autentikasi basis data IAM kompatibel dengan Aurora. Untuk informasi selengkapnya tentang cara mengautentikasi ke klaster DB Anda menggunakan IAM, lihat Autentikasi basis data IAM.

Peran IAM

Peran IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus. Peran ini mirip dengan pengguna, tetapi tidak terkait dengan orang tertentu. Anda dapat mengambil peran IAM untuk sementara AWS Management Console dengan beralih peran. Anda dapat mengambil peran dengan memanggil operasi AWS CLI atau AWS API atau dengan menggunakan URL kustom. Untuk informasi selengkapnya tentang cara menggunakan peran, lihat Menggunakan peran IAM dalam Panduan Pengguna IAM.

Peran IAM dengan kredensial sementara berguna dalam situasi berikut:

- Izin pengguna sementara – Pengguna dapat mengambil peran IAM untuk mendapatkan izin yang berbeda sementara waktu agar dapat melakukan tugas tertentu.
- Akses pengguna terfederasi – Untuk menetapkan izin ke identitas terfederasi, Anda membuat peran dan menentukan izin untuk peran tersebut. Saat identitas terfederasi mengautentikasi, identitas tersebut akan dikaitkan dengan peran dan diberi izin yang ditentukan oleh peran tersebut. Untuk informasi tentang peran-peran untuk federasi, lihat Membuat peran untuk Penyedia Identitas pihak ketiga dalam Panduan Pengguna IAM. Jika Anda menggunakan Pusat Identitas IAM, Anda perlu mengonfigurasi set izin. Untuk mengontrol apa yang dapat diakses identitas Anda setelah identitas tersebut diautentikasi, Pusat Identitas IAM mengorelasikan izin yang diatur ke peran dalam IAM. Untuk informasi tentang rangkaian izin, silakan lihat Rangkaian izin dalam Panduan Pengguna AWS IAM Identity Center .

- Akses lintas akun – Anda dapat menggunakan peran IAM untuk mengizinkan seseorang (prinsipal tepercaya) di akun lain untuk mengakses sumber daya di akun Anda. Peran adalah cara utama untuk memberikan akses lintas akun. Namun, dengan beberapa Layanan AWS, Anda dapat melampirkan kebijakan secara langsung ke sumber daya (alih-alih menggunakan peran sebagai proxy). Untuk mempelajari perbedaan antara kebijakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Bagaimana peran IAM berbeda dari kebijakan berbasis sumber daya dalam Panduan Pengguna IAM.
- Akses lintas layanan — Beberapa Layanan AWS menggunakan fitur lain Layanan AWS. Sebagai contoh, ketika Anda melakukan panggilan dalam suatu layanan, biasanya layanan tersebut menjalankan aplikasi di Amazon EC2 atau menyimpan objek di Amazon S3. Sebuah layanan mungkin melakukannya menggunakan izin prinsipal yang memanggil, menggunakan peran layanan, atau peran terkait layanan.
  - Sesi akses teruskan — Saat Anda menggunakan pengguna atau peran IAM untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Saat Anda menggunakan beberapa layanan, Anda mungkin melakukan tindakan yang kemudian memulai tindakan lain di layanan yang berbeda. FAS menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Permintaan FAS hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain Layanan AWS atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan saat membuat permintaan FAS, lihat Teruskan sesi akses.
  - Peran layanan – Peran layanan adalah peran IAM yang diambil oleh layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, memodifikasi, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.
  - Peran terkait layanan — Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan dapat menggunakan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.
- Aplikasi yang berjalan di Amazon EC2 — Anda dapat menggunakan peran IAM untuk mengelola kredensi sementara untuk aplikasi yang berjalan pada instans EC2 dan membuat atau permintaan API. AWS CLI AWS Cara ini lebih dianjurkan daripada menyimpan kunci akses dalam instans EC2. Untuk menetapkan AWS peran ke instans EC2 dan membuatnya tersedia untuk semua aplikasinya, Anda membuat profil instance yang dilampirkan ke instance. Profil instans berisi peran dan memungkinkan program yang berjalan di instans EC2 mendapatkan kredensial sementara.

Untuk informasi selengkapnya, lihat Menggunakan peran IAM untuk memberikan izin ke aplikasi yang berjalan di instans Amazon EC2 dalam Panduan Pengguna IAM.

Untuk mempelajari apakah harus menggunakan peran IAM, lihat Kapan harus membuat peran IAM (bukan pengguna) dalam Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke identitas atau sumber daya IAM. AWS Kebijakan adalah objek AWS yang, ketika dikaitkan dengan identitas atau sumber daya, menentukan izinnya. AWS mengevaluasi kebijakan ini ketika entitas (pengguna root, pengguna, atau peran IAM) membuat permintaan. Izin dalam kebijakan dapat menentukan permintaan yang diizinkan atau ditolak. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang struktur dan konten dokumen kebijakan JSON, lihat Gambaran umum kebijakan JSON dalam Panduan Pengguna IAM.

Administrator dapat menggunakan kebijakan untuk menentukan siapa yang memiliki akses ke AWS sumber daya, dan tindakan apa yang dapat mereka lakukan pada sumber daya tersebut. Setiap entitas IAM (set izin atau peran) dimulai tanpa izin. Dengan kata lain, secara default, pengguna tidak dapat melakukan apa pun, termasuk mengubah kata sandinya sendiri. Untuk memberikan izin kepada pengguna untuk melakukan sesuatu, administrator harus melampirkan kebijakan izin kepada pengguna. Atau administrator dapat menambahkan pengguna ke grup yang memiliki izin yang dimaksudkan. Ketika administrator memberikan izin untuk grup, semua pengguna dalam grup tersebut akan diberi izin tersebut.

Kebijakan IAM mendefinisikan izin untuk suatu tindakan terlepas dari metode yang Anda gunakan untuk melakukan operasinya. Sebagai contoh, anggap saja Anda memiliki kebijakan yang mengizinkan tindakan `iam:GetRole`. Pengguna dengan kebijakan tersebut bisa mendapatkan informasi peran dari AWS Management Console, API AWS CLI, atau AWS API.

Kebijakan berbasis identitas

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke identitas, seperti set izin atau peran. Kebijakan ini mengontrol tindakan apa yang bisa dilakukan oleh identitas tersebut, sumber daya yang mana, dan dalam kondisi apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Membuat kebijakan IAM dalam Panduan Pengguna IAM.

Kebijakan berbasis identitas dapat dikategorikan lebih lanjut sebagai kebijakan inline atau kebijakan terkelola. Kebijakan inline disematkan secara langsung ke satu set izin atau peran. Kebijakan

terkelola adalah kebijakan mandiri yang dapat Anda lampirkan ke beberapa set izin dan peran di AWS akun Anda. Kebijakan AWS terkelola mencakup kebijakan terkelola dan kebijakan yang dikelola pelanggan. Untuk mempelajari cara memilih antara kebijakan terkelola atau kebijakan inline, lihat Memilih antara kebijakan terkelola dan kebijakan inline dalam Panduan Pengguna IAM.

Untuk informasi tentang kebijakan AWS terkelola yang khusus untuk Amazon Aurora, lihat. AWS kebijakan terkelola untuk Amazon RDS

Jenis kebijakan lainnya

AWS mendukung jenis kebijakan tambahan yang kurang umum. Jenis kebijakan ini dapat mengatur izin maksimum yang diberikan kepada Anda oleh jenis kebijakan yang lebih umum.

- Batasan izin – Batasan izin adalah fitur lanjutan untuk menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM (set izin atau peran). Anda dapat menetapkan batasan izin untuk suatu entitas. Izin yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas milik entitas dan batasan izin tersebut. Kebijakan berbasis sumber daya yang menentukan set izin atau peran di bidang `Principal` tidak dibatasi oleh batasan izin. Penolakan eksplisit dalam salah satu kebijakan ini membatalkan izin. Untuk informasi selengkapnya tentang batasan izin, lihat Batasan izin untuk entitas IAM dalam Panduan Pengguna IAM.
- Kebijakan kontrol layanan (SCP) — SCP adalah kebijakan JSON yang menentukan izin maksimum untuk organisasi atau unit organisasi (OU) di. AWS Organizations AWS Organizations adalah layanan untuk mengelompokkan dan mengelola beberapa AWS akun secara terpusat yang dimiliki bisnis Anda. Jika Anda mengaktifkan semua fitur di sebuah organisasi, maka Anda dapat menerapkan kebijakan kontrol layanan (SCP) ke salah satu atau semua akun Anda. SCP membatasi izin untuk entitas di akun anggota, termasuk masing-masing. Pengguna root akun AWS Untuk informasi selengkapnya tentang Organisasi dan SCP, lihat Cara kerja SCP dalam Panduan Pengguna AWS Organizations .
- Kebijakan sesi – Kebijakan sesi adalah kebijakan lanjutan yang Anda berikan sebagai parameter saat Anda membuat sesi sementara secara programatis untuk peran atau pengguna terfederasi. Izin sesi yang dihasilkan adalah perpotongan antara set izin atau kebijakan berbasis identitas peran dan kebijakan sesi tersebut. Izin juga dapat berasal dari kebijakan berbasis sumber daya. Penolakan eksplisit dalam salah satu kebijakan ini membatalkan izin. Untuk informasi selengkapnya, lihat Kebijakan sesi dalam Panduan Pengguna IAM.

Beberapa jenis kebijakan

Ketika beberapa jenis kebijakan berlaku untuk sebuah permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan saat beberapa jenis kebijakan terlibat, lihat Logika evaluasi kebijakan di Panduan Pengguna IAM.

Cara kerja Amazon Aurora dengan IAM

Sebelum menggunakan IAM untuk mengelola akses ke Amazon Aurora, Anda harus memahami fitur IAM yang dapat digunakan dengan Aurora.

Fitur IAM yang dapat digunakan dengan Amazon Aurora

Fitur IAM	Dukungan Amazon Aurora
Kebijakan berbasis identitas	Ya
Kebijakan berbasis sumber daya	Tidak
Tindakan kebijakan	Ya
Sumber daya kebijakan	Ya
kunci-kunci persyaratan kebijakan (spesifik layanan)	Ya
ACL	Tidak
Kontrol akses berbasis atribut (ABAC) (tag dalam kebijakan)	Ya
Kredensial sementara	Ya
Teruskan sesi akses	Ya
Peran layanan	Ya
Peran terkait layanan	Ya

Untuk mendapatkan tampilan tingkat tinggi tentang cara Aurora dan layanan AWS lainnya bekerja dengan IAM, AWS lihat layanan yang bekerja dengan IAM di Panduan Pengguna IAM.

Topik

- Kebijakan berbasis identitas Aurora
- Kebijakan berbasis sumber daya dalam Aurora
- Tindakan kebijakan untuk Aurora
- Sumber daya kebijakan untuk Aurora
- Kunci kondisi kebijakan untuk Aurora
- Daftar kontrol akses (ACL) di Aurora
- Kontrol akses berbasis atribut (ABAC) dalam kebijakan dengan tag Aurora
- Menggunakan kredensial sementara dengan Aurora
- Teruskan sesi akses untuk
- Peran layanan untuk Aurora
- Peran terkait layanan untuk Aurora

Kebijakan berbasis identitas Aurora

Mendukung kebijakan berbasis identitas	Ya

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke sebuah identitas, seperti pengguna IAM, grup pengguna IAM, atau peran IAM. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan pengguna dan peran, di sumber daya mana, dan dengan ketentuan apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Membuat kebijakan IAM dalam Panduan Pengguna IAM.

Dengan kebijakan berbasis identitas IAM, Anda dapat menentukan tindakan dan sumber daya yang diizinkan atau ditolak, serta ketentuan terkait jenis tindakan yang diizinkan atau ditolak. Anda tidak dapat menentukan pengguna utama dalam kebijakan berbasis identitas karena kebijakan ini berlaku untuk pengguna atau peran yang dilampiri kebijakan. Untuk mempelajari semua elemen yang dapat digunakan dalam kebijakan JSON, lihat Referensi elemen kebijakan JSON IAM dalam Panduan Pengguna IAM.

Contoh kebijakan berbasis identitas untuk Aurora

Untuk melihat contoh kebijakan berbasis identitas Aurora, lihat Contoh kebijakan berbasis identitas untuk Amazon Aurora.

Kebijakan berbasis sumber daya dalam Aurora

Mendukung kebijakan berbasis sumber daya	Tidak

Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya yang dilampiri kebijakan tersebut, kebijakan ini menentukan jenis tindakan yang dapat dilakukan oleh pengguna utama tertentu di sumber daya tersebut dan apa ketentuannya. Anda harus menentukan pengguna utama dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan seluruh akun atau entitas IAM di akun lain sebagai pengguna utama dalam kebijakan berbasis sumber daya. Menambahkan pengguna utama lintas akun ke kebijakan berbasis sumber daya bagian dari membangun hubungan kepercayaan. Ketika prinsipal dan sumber daya berbeda Akun AWS, administrator IAM di akun tepercaya juga harus memberikan izin entitas utama (pengguna atau peran) untuk mengakses sumber daya. Izin diberikan dengan melampirkan kebijakan berbasis identitas ke entitas tersebut. Namun, jika kebijakan berbasis sumber daya memberikan akses kepada pengguna utama dalam akun yang sama, kebijakan berbasis identitas lainnya tidak diperlukan. Untuk informasi selengkapnya, lihat Perbedaan peran IAM dengan kebijakan berbasis sumber daya di Panduan Pengguna IAM.

Tindakan kebijakan untuk Aurora

Mendukung tindakan kebijakan	Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, pengguna utama mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Elemen `Action` dari kebijakan JSON menjelaskan tindakan yang dapat Anda gunakan untuk mengizinkan atau menolak akses dalam sebuah kebijakan. Tindakan kebijakan biasanya memiliki nama yang sama dengan operasi AWS API terkait. Ada beberapa pengecualian, misalnya tindakan

hanya izin yang tidak memiliki operasi API yang cocok. Ada juga beberapa operasi yang memerlukan beberapa tindakan dalam suatu kebijakan. Tindakan tambahan ini disebut tindakan dependen.

Menyertakan tindakan dalam suatu kebijakan untuk memberikan izin melakukan operasi terkait.

Tindakan kebijakan di Aurora menggunakan awalan berikut sebelum tindakan: `rds:`. Misalnya, untuk memberikan izin kepada seseorang untuk menjelaskan instans DB dengan operasi API `DescribeDBInstances` Amazon RDS, Anda menyertakan tindakan `rds:DescribeDBInstances` dalam kebijakan mereka. Pernyataan kebijakan harus memuat elemen `Action` atau `NotAction`. Aurora menentukan serangkaian tindakannya sendiri yang menjelaskan tugas yang dapat Anda lakukan dengan layanan ini.

Untuk menentukan beberapa tindakan dalam satu pernyataan, pisahkan tindakan dengan koma seperti berikut:

```
"Action": [
      "rds:action1",
      "rds:action2"
```

Anda juga dapat menentukan beberapa tindakan menggunakan wildcard (*). Misalnya, untuk menentukan semua tindakan yang dimulai dengan kata `Describe`, sertakan tindakan berikut.

```
"Action": "rds:Describe*"
```

Untuk melihat daftar tindakan Aurora, lihat Tindakan yang Ditentukan oleh Amazon RDS di Referensi Otorisasi Layanan.

Sumber daya kebijakan untuk Aurora

Mendukung sumber daya kebijakan	Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, pengguna utama mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Elemen kebijakan JSON `Resource` menentukan objek atau beberapa objek yang menjadi target penerapan tindakan. Pernyataan harus menyertakan elemen `Resource` atau `NotResource`. Praktik

terbaiknya, tentukan sumber daya menggunakan Amazon Resource Name (ARN). Anda dapat melakukan ini untuk tindakan yang mendukung jenis sumber daya tertentu, yang dikenal sebagai izin tingkat sumber daya.

Untuk tindakan yang tidak mendukung izin di tingkat sumber daya, misalnya operasi pencantuman, gunakan wildcard (*) untuk mengindikasikan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

```
"Resource": "*"
```

Sumber daya instans DB memiliki Amazon Resource Name (ARN) berikut.

```
arn:${Partition}:rds:${Region}:${Account}:{ResourceType}/${Resource}
```

Untuk informasi selengkapnya tentang format ARN, lihat Amazon Resource Names (ARN) dan ruang nama AWS layanan.

Misalnya, untuk menentukan instans DB `dbtest` dalam pernyataan Anda, gunakan ARN berikut.

```
"Resource": "arn:aws:rds:us-west-2:123456789012:db:dbtest"
```

Untuk menentukan semua instans DB milik akun tertentu, gunakan wildcard (*).

```
"Resource": "arn:aws:rds:us-east-1:123456789012:db:*"
```

Beberapa operasi API RDS, seperti operasi untuk membuat sumber daya, tidak dapat dilakukan pada sumber daya tertentu. Jika demikian, gunakan wildcard (*).

```
"Resource": "*"
```

Banyak operasi API Amazon RDS menggunakan beberapa sumber daya. Misalnya, `CreateDBInstance` membuat instans DB. Anda dapat menentukan bahwa seorang pengguna harus menggunakan grup keamanan dan grup parameter spesifik saat membuat instans DB. Untuk menentukan beberapa sumber daya dalam satu pernyataan, pisahkan ARN dengan koma.

```
"Resource": [
      "resource1",
```

```
    "resource2"
```

Untuk melihat daftar jenis sumber daya Aurora dan ARN-nya, lihat Sumber Daya yang Ditentukan oleh Amazon RDS di Referensi Otorisasi Layanan. Untuk mempelajari jenis tindakan yang dapat Anda tentukan dengan ARN di tiap sumber daya, lihat Tindakan yang Ditentukan oleh Amazon RDS.

Kunci kondisi kebijakan untuk Aurora

Mendukung kunci kondisi kebijakan spesifik layanan	Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, pengguna utama mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Elemen `Condition` (atau blok `Condition`) memungkinkan Anda menentukan kondisi di mana suatu pernyataan akan diterapkan. Elemen `Condition` bersifat opsional. Anda dapat membuat ekspresi kondisional yang menggunakan operator kondisi, misalnya sama dengan atau kurang dari, untuk mencocokkan kondisi dalam kebijakan dengan nilai-nilai yang diminta.

Jika Anda menentukan beberapa elemen `Condition` dalam satu pernyataan, atau beberapa kunci dalam satu elemen `Condition`, AWS akan mengevaluasinya dengan menggunakan operasi `AND` logis. Jika Anda menentukan beberapa nilai untuk satu kunci kondisi, AWS mengevaluasi kondisi menggunakan `OR` operasi logis. Semua kondisi harus dipenuhi sebelum izin pernyataan diberikan.

Anda juga dapat menggunakan variabel placeholder saat menentukan kondisi. Misalnya, Anda dapat memberikan izin kepada pengguna IAM untuk mengakses sumber daya hanya jika izin tersebut mempunyai tag yang sesuai dengan nama pengguna IAM mereka. Untuk informasi selengkapnya, lihat Elemen kebijakan IAM: variabel dan tag di Panduan Pengguna IAM.

AWS mendukung kunci kondisi global dan kunci kondisi khusus layanan. Untuk melihat semua kunci kondisi AWS global, lihat kunci konteks kondisi AWS global di Panduan Pengguna IAM.

Aurora menentukan set kunci kondisinya sendiri dan juga mendukung penggunaan beberapa kunci kondisi global. Untuk melihat semua kunci kondisi AWS global, lihat kunci konteks kondisi AWS global di Panduan Pengguna IAM.

Semua operasi API RDS mendukung kunci kondisi `aws:RequestedRegion`.

Untuk melihat daftar kunci kondisi Aurora, lihat Kunci Kondisi untuk Amazon RDS di Referensi Otorisasi Layanan. Untuk mempelajari tindakan dan sumber daya yang dapat Anda gunakan dengan kunci kondisi, lihat Tindakan yang Ditentukan oleh Amazon RDS.

Daftar kontrol akses (ACL) di Aurora

Mendukung daftar kontrol akses (ACL)	Tidak

Daftar kontrol akses (ACL) mengontrol pengguna utama (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACL sama dengan kebijakan berbasis sumber daya, meskipun tidak menggunakan format dokumen kebijakan JSON.

Kontrol akses berbasis atribut (ABAC) dalam kebijakan dengan tag Aurora

Mendukung tag kontrol akses berbasis atribut (ABAC) dalam kebijakan	Ya

Kontrol akses berbasis atribut (ABAC) adalah strategi otorisasi yang menentukan izin berdasarkan atribut. Dalam AWS, atribut ini disebut tag. Anda dapat melampirkan tag ke entitas IAM (pengguna atau peran) dan ke banyak AWS sumber daya. Pemberian tanda ke entitas dan sumber daya adalah langkah pertama dari ABAC. Kemudian, rancanglah kebijakan ABAC untuk mengizinkan operasi saat tag milik pengguna utama cocok dengan tag yang ada di sumber daya yang ingin diakses.

ABAC sangat berguna di lingkungan yang berkembang dengan cepat dan berguna di situasi di mana pengelolaan kebijakan menjadi rumit.

Untuk mengendalikan akses berdasarkan tag, berikan informasi tentang tag di elemen kondisi dari kebijakan dengan menggunakan kunci kondisi `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, atau `aws:TagKeys`.

Jika sebuah layanan mendukung ketiga kunci kondisi untuk setiap jenis sumber daya, nilainya adalah Ya untuk layanan tersebut. Jika suatu layanan mendukung ketiga kunci kondisi hanya untuk beberapa jenis sumber daya, nilainya adalah Parsial.

Untuk informasi selengkapnya tentang ABAC, lihat Apa itu ABAC? di Panduan Pengguna IAM. Untuk melihat tutorial terkait langkah-langkah penyiapan ABAC, lihat Menggunakan kontrol akses berbasis atribut (ABAC) di Panduan Pengguna IAM.

Untuk informasi selengkapnya cara memberi tag ke sumber daya Aurora, lihat Menentukan kondisi: Menggunakan tag kustom. Untuk melihat contoh kebijakan berbasis identitas untuk membatasi akses ke sumber daya berdasarkan tag pada sumber daya tersebut, lihat Berikan izin untuk tindakan atas suatu sumber daya dengan tag tertentu dengan dua nilai yang berbeda.

Menggunakan kredensial sementara dengan Aurora

Mendukung kredensial sementara	Ya

Beberapa Layanan AWS tidak berfungsi saat Anda masuk menggunakan kredensi sementara. Untuk informasi tambahan, termasuk yang Layanan AWS bekerja dengan kredensi sementara, lihat Layanan AWS yang bekerja dengan IAM di Panduan Pengguna IAM.

Anda menggunakan kredensi sementara jika Anda masuk AWS Management Console menggunakan metode apa pun kecuali nama pengguna dan kata sandi. Misalnya, ketika Anda mengakses AWS menggunakan tautan masuk tunggal (SSO) perusahaan Anda, proses tersebut secara otomatis membuat kredensi sementara. Anda juga akan membuat kredensial sementara secara otomatis saat masuk ke konsol sebagai pengguna dan kemudian beralih peran. Untuk informasi selengkapnya tentang cara beralih peran, lihat Beralih peran (konsol) di Panduan Pengguna IAM.

Anda dapat membuat kredenal sementara secara manual menggunakan API AWS CLI atau AWS . Anda kemudian dapat menggunakan kredensi sementara tersebut untuk mengakses. AWS AWS merekomendasikan agar Anda secara dinamis menghasilkan kredensi sementara alih-alih menggunakan kunci akses jangka panjang. Untuk informasi selengkapnya, lihat Kredensial keamanan sementara di IAM.

Teruskan sesi akses untuk

Mendukung sesi akses ke depan	Ya

Saat Anda menggunakan pengguna atau peran IAM untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Saat Anda menggunakan beberapa layanan, Anda mungkin melakukan tindakan yang kemudian memulai tindakan lain di layanan yang berbeda. FAS menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Permintaan FAS hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain Layanan AWS atau sumber daya untuk

menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan saat membuat permintaan FAS, lihat Meneruskan sesi akses.

Peran layanan untuk Aurora

Mendukung peran layanan	Ya

Peran layanan adalah peran IAM yang diambil oleh layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Membuat peran untuk mendelegasikan izin ke Layanan AWS di Panduan Pengguna IAM.

> **Warning**
>
> Mengubah izin untuk peran layanan dapat mengganggu fungsionalitas Aurora. Edit peran layanan hanya jika Aurora menyediakan panduan untuk melakukannya.

Peran terkait layanan untuk Aurora

Mendukung peran terkait layanan	Ya

Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan dapat menggunakan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.

Untuk detail tentang cara menggunakan peran terkait layanan Aurora, lihat Menggunakan peran tertaut layanan untuk Amazon Aurora.

Contoh kebijakan berbasis identitas untuk Amazon Aurora

Secara default, peran dan kumpulan izin tidak memiliki izin untuk membuat atau mengubah sumber daya Aurora. Mereka juga tidak dapat melakukan tugas menggunakan AWS Management Console, AWS CLI, atau AWS API. Administrator harus membuat kebijakan IAM yang memberikan izin kepada

peran atau kumpulan izin untuk menjalankan operasi API tertentu pada sumber daya tertentu yang diperlukan. Administrator kemudian dapat melampirkan kebijakan tersebut ke peran atau kumpulan izin yang memerlukan izin tersebut.

Untuk mempelajari cara membuat kebijakan berbasis identitas IAM menggunakan contoh dokumen kebijakan JSON ini, lihat Membuat kebijakan pada tab JSON dalam Panduan Pengguna IAM.

Topik

- Praktik terbaik kebijakan
- Menggunakan konsol Aurora
- Izinkan pengguna melihat izin mereka sendiri
- Izinkan pengguna untuk membuat instans DB di akun AWS
- Izin yang diperlukan untuk menggunakan konsol
- Mengizinkan pengguna melakukan setiap tindakan yang dijelaskan pada sumber daya RDS
- Mengizinkan pengguna membuat instans DB yang menggunakan grup parameter DB dan grup subnet yang telah ditentukan
- Berikan izin untuk tindakan atas suatu sumber daya dengan tag tertentu dengan dua nilai yang berbeda
- Mencegah pengguna menghapus instans DB
- Menolak semua akses ke sumber daya
- Contoh Kebijakan: Menggunakan kunci kondisi
- Menentukan kondisi: Menggunakan tag kustom

Praktik terbaik kebijakan

Kebijakan berbasis identitas menentukan apakah seseorang dapat membuat, mengakses, atau menghapus sumber daya Amazon RDS yang ada di akun Anda. Tindakan ini dikenai biaya untuk Akun AWS Anda. Ketika Anda membuat atau mengedit kebijakan berbasis identitas, ikuti panduan dan rekomendasi ini:

- Mulailah dengan kebijakan AWS terkelola dan beralih ke izin hak istimewa paling sedikit — Untuk mulai memberikan izin kepada pengguna dan beban kerja Anda, gunakan kebijakan AWS terkelola yang memberikan izin untuk banyak kasus penggunaan umum. Mereka tersedia di Anda Akun AWS. Kami menyarankan Anda mengurangi izin lebih lanjut dengan menentukan kebijakan

yang dikelola AWS pelanggan yang khusus untuk kasus penggunaan Anda. Untuk informasi selengkapnya, lihat kebijakan yang dikelola AWS atau kebijakan yang dikelola AWS untuk fungsi pekerjaan di Panduan Pengguna IAM.

- Menerapkan izin dengan hak akses paling rendah – Ketika Anda menetapkan izin dengan kebijakan IAM, hanya berikan izin yang diperlukan untuk melakukan tugas. Anda melakukan ini dengan menentukan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, juga dikenal sebagai izin hak akses paling rendah. Untuk informasi selengkapnya tentang cara menggunakan IAM untuk menerapkan izin, lihat Kebijakan dan izin di IAM di Panduan Pengguna IAM.
- Gunakan kondisi dalam kebijakan IAM untuk membatasi akses lebih lanjut – Anda dapat menambahkan kondisi ke kebijakan Anda untuk membatasi akses ke tindakan dan sumber daya. Sebagai contoh, Anda dapat menulis kondisi kebijakan untuk menentukan bahwa semua permintaan harus dikirim menggunakan SSL. Anda juga dapat menggunakan ketentuan untuk memberikan akses ke tindakan layanan jika digunakan melalui yang spesifik Layanan AWS, seperti AWS CloudFormation. Untuk informasi selengkapnya, lihat Elemen kebijakan JSON IAM: Syarat di Panduan Pengguna IAM.
- Menggunakan IAM Access Analyzer untuk memvalidasi kebijakan IAM Anda guna memastikan izin yang aman dan berfungsi – IAM Access Analyzer memvalidasi kebijakan baru dan yang sudah ada sehingga kebijakan tersebut mematuhi bahasa kebijakan IAM (JSON) dan praktik terbaik IAM. IAM Access Analyzer menyediakan lebih dari 100 pemeriksaan kebijakan dan rekomendasi yang dapat ditindaklanjuti untuk membantu Anda membuat kebijakan yang aman dan fungsional. Untuk informasi selengkapnya, lihat validasi kebijakan Analizer Akses IAM di Panduan Pengguna IAM.
- Memerlukan otentikasi multi-faktor (MFA) - Jika Anda memiliki skenario yang mengharuskan pengguna IAM atau pengguna root di Anda, Akun AWS aktifkan MFA untuk keamanan tambahan. Untuk mewajibkan MFA saat operasi API dipanggil, tambahkan kondisi MFA pada kebijakan Anda. Untuk informasi selengkapnya, lihat Mengonfigurasi akses API yang dilindungi MFA di Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang praktik terbaik dalam IAM, lihat Praktik terbaik keamanan di IAM di Panduan Pengguna IAM.

Menggunakan konsol Aurora

Untuk mengakses konsol Amazon Aurora, Anda harus memiliki kumpulan izin minimum. Izin ini harus memungkinkan Anda untuk membuat daftar dan melihat detail tentang sumber daya Amazon Aurora di Anda. Akun AWS Jika Anda membuat kebijakan berbasis identitas yang lebih ketat daripada

izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya untuk entitas (pengguna atau peran) dengan kebijakan tersebut.

Anda tidak perlu mengizinkan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke AWS CLI atau AWS API. Sebagai alternatif, hanya izinkan akses ke tindakan yang cocok dengan operasi API yang sedang Anda coba lakukan.

Untuk memastikan bahwa entitas tersebut masih dapat menggunakan konsol Aurora, lampirkan juga kebijakan terkelola AWS berikut ke entitas.

```
AmazonRDSReadOnlyAccess
```

Untuk informasi selengkapnya, lihat Menambahkan izin ke pengguna di Panduan Pengguna IAM.

Izinkan pengguna melihat izin mereka sendiri

Contoh ini menunjukkan cara membuat kebijakan yang mengizinkan para pengguna IAM melihat kebijakan inline dan terkelola yang dilampirkan ke identitas pengguna mereka. Kebijakan ini mencakup izin untuk menyelesaikan tindakan ini di konsol atau menggunakan API atau secara terprogram. AWS CLI AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
```

```
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Izinkan pengguna untuk membuat instans DB di akun AWS

Berikut ini adalah contoh kebijakan yang memungkinkan pengguna dengan ID `123456789012` untuk membuat instans DB untuk AWS akun Anda. Kebijakan ini mewajibkan nama instans DB baru dimulai dengan `test`. Instans DB yang baru juga harus menggunakan mesin basis data MySQL dan kelas instans DB `db.t2.micro`. Selain itu, instans DB baru harus menggunakan grup opsi dan grup parameter DB yang dimulai dengan `default`, dan harus menggunakan grup subnet `default`.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowCreateDBInstanceOnly",
         "Effect": "Allow",
         "Action": [
            "rds:CreateDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:db:test*",
            "arn:aws:rds:*:123456789012:og:default*",
            "arn:aws:rds:*:123456789012:pg:default*",
            "arn:aws:rds:*:123456789012:subgrp:default"
         ],
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql",
               "rds:DatabaseClass": "db.t2.micro"
            }
         }
      }
   ]
```

```
}
```

Kebijakan ini mencakup pernyataan tunggal yang menentukan izin berikut untuk pengguna:

- Kebijakan ini memungkinkan pengguna untuk membuat instans DB menggunakan operasi CreatedBInstance API (ini juga berlaku untuk perintah dan). create-db-instance AWS CLI AWS Management Console
- Elemen `Resource` menentukan bahwa pengguna dapat melakukan tindakan pada atau dengan sumber daya. Anda menentukan sumber daya menggunakan Amazon Resource Name (ARN). ARN ini mencakup nama layanan yang dimiliki sumber daya (`rds`), AWS Wilayah (*menunjukkan wilayah mana pun dalam contoh ini), nomor AWS akun (123456789012adalah nomor akun dalam contoh ini), dan jenis sumber daya. Untuk informasi selengkapnya tentang cara membuat ARN, lihat Bekerja dengan Amazon Resource Name (ARN) di Amazon RDS.

  Elemen `Resource` dalam contoh menentukan batasan kebijakan berikut pada sumber daya untuk pengguna:

  - ID instans DB untuk instans DB baru harus dimulai dengan `test` (misalnya, `testCustomerData1`, `test-region2-data`).
  - Grup opsi untuk instans DB baru harus dimulai dengan `default`.
  - Grup parameter DB opsi untuk instans DB baru harus dimulai dengan `default`.
  - Grup subnet untuk instans DB baru harus berupa grup subnet `default`.
- Elemen `Condition` menentukan bahwa mesin DB harus berupa MySQL dan kelas instans DB harus berupa `db.t2.micro`. Elemen `Condition` menentukan kondisi ketika kebijakan harus diberlakukan. Anda dapat menambahkan izin atau batasan tambahan dengan menggunakan elemen `Condition`. Untuk informasi selengkapnya tentang cara menentukan kondisi, lihat Kunci kondisi kebijakan untuk Aurora. Contoh ini menetapkan kondisi `rds:DatabaseEngine` dan `rds:DatabaseClass`. Untuk informasi tentang nilai kondisi yang valid untuk `rds:DatabaseEngine`, lihat daftar di bagian parameter `Engine` di CreateDBInstance. Untuk informasi tentang nilai kondisi yang valid untuk `rds:DatabaseClass`, Lihat Mesin DB yang didukung untuk kelas instans DB.

Kebijakan ini tidak menentukan elemen `Principal` karena dalam kebijakan berbasis identitas, Anda tidak menentukan pengguna utama yang mendapatkan izin. Saat Anda menyematkan kebijakan kepada pengguna, pengguna ini menjadi pengguna utama implisit. Saat Anda menyematkan kebijakan izin pada peran IAM, pengguna utama yang diidentifikasi dalam kebijakan kepercayaan peran tersebut akan mendapatkan izin.

Untuk melihat daftar tindakan Aurora, lihat Tindakan yang Ditentukan oleh Amazon RDS di Referensi Otorisasi Layanan.

Izin yang diperlukan untuk menggunakan konsol

Agar pengguna dapat bekerja dengan konsol, pengguna tersebut harus memiliki kumpulan izin minimum. Izin ini memungkinkan pengguna untuk mendeskripsikan sumber daya Amazon Aurora untuk akun AWS mereka dan untuk memberikan informasi terkait lainnya, termasuk keamanan Amazon EC2 dan informasi jaringan.

Jika Anda membuat kebijakan IAM yang lebih ketat daripada izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya untuk pengguna dengan kebijakan IAM. Untuk memastikan bahwa pengguna masih dapat menggunakan konsol, lampirkan juga kebijakan yang dikelola `AmazonRDSReadOnlyAccess` kepada pengguna, sebagaimana dijelaskan dalam Mengelola akses menggunakan kebijakan.

Anda tidak perlu memperbolehkan izin konsol minimum bagi pengguna yang hanya melakukan panggilan ke AWS CLI atau Amazon RDS API.

Kebijakan berikut memberikan akses penuh ke semua sumber daya Amazon Aurora untuk akun root: AWS

```
AmazonRDSFullAccess
```

Mengizinkan pengguna melakukan setiap tindakan yang dijelaskan pada sumber daya RDS

Kebijakan izin berikut memberikan izin kepada pengguna untuk menjalankan semua tindakan yang dimulai dengan `Describe`. Tindakan ini menunjukkan informasi tentang sumber daya RDS, seperti instans DB. Karakter wildcard (*) dalam elemen `Resource` menunjukkan bahwa tindakan diperbolehkan untuk semua sumber daya Amazon Aurora yang dimiliki akun tersebut.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowRDSDescribe",
         "Effect": "Allow",
```

```
         "Action": "rds:Describe*",
         "Resource": "*"
      }
   ]
}
```

Mengizinkan pengguna membuat instans DB yang menggunakan grup parameter DB dan grup subnet yang telah ditentukan

Kebijakan izin berikut memberikan izin untuk hanya memperbolehkan pengguna membuat instans DB yang harus menggunakan grup parameter DB `mydbpg` dan grup subnet DB `mydbsubnetgroup`.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": [
            "arn:aws:rds:*:*:pg:mydbpg",
            "arn:aws:rds:*:*:subgrp:mydbsubnetgroup"
         ]
      }
   ]
}
```

Berikan izin untuk tindakan atas suatu sumber daya dengan tag tertentu dengan dua nilai yang berbeda

Anda dapat menggunakan kondisi dalam kebijakan berbasis identitas untuk mengontrol akses ke sumber daya Aurora berdasarkan tag. Kebijakan berikut memungkinkan izin untuk melakukan operasi `CreateDBSnapshot` API pada instans DB dengan tag `stage` diatur ke `development` atau `test`.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
```

```
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

Kebijakan berikut memungkinkan izin untuk melakukan operasi `ModifyDBInstance` API pada instans DB dengan tag `stage` diatur ke `development` atau `test`.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource": [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
         ]
      },
      {
```

```
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

Mencegah pengguna menghapus instans DB

Kebijakan izin berikut memberikan izin untuk mencegah pengguna menghapus instans DB tertentu. Misalnya, Anda mungkin ingin menolak kemampuan untuk menghapus instans DB produksi Anda kepada setiap pengguna yang bukan administrator.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "DenyDelete1",
         "Effect": "Deny",
         "Action": "rds:DeleteDBInstance",
         "Resource": "arn:aws:rds:us-west-2:123456789012:db:my-mysql-instance"
      }
   ]
}
```

Menolak semua akses ke sumber daya

Anda juga dapat secara eksplisit menolak akses ke sumber daya. Kebijakan penolakan lebih diutamakan daripada kebijakan yang diizinkan. Kebijakan berikut secara eksplisit menolak kemampuan pengguna untuk mengelola sumber daya:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Deny",
         "Action": "rds:*",
         "Resource": "arn:aws:rds:us-east-1:123456789012:db:mydb"
      }
   ]
}
```

Contoh Kebijakan: Menggunakan kunci kondisi

Berikut ini adalah contoh cara menggunakan kunci kondisi dalam kebijakan izin IAM Amazon Aurora.

Contoh 1: Memberikan izin untuk membuat instans DB yang menggunakan mesin DB spesifik dan tidak berupa Multi-AZ

Kebijakan berikut menggunakan kunci kondisi RDS dan memungkinkan pengguna membuat instans DB yang menggunakan mesin basis data MySQL saja dan tidak menggunakan MultiAZ. Elemen `Condition` menunjukkan persyaratan bahwa mesin basis data adalah MySQL.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "AllowMySQLCreate",
         "Effect": "Allow",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseEngine": "mysql"
            },
            "Bool": {
```

```
                "rds:MultiAz": false
            }
         }
      }
   ]
}
```

Contoh 2: Secara eksplisit menolak izin untuk membuat instans DB untuk kelas instans DB tertentu dan membuat instans DB yang menggunakan IOPS yang Tersedia

Kebijakan berikut secara eksplisit menolak izin untuk membuat instans DB yang menggunakan kelas instans DB `r3.8xlarge` dan `m4.10xlarge`, yang merupakan kelas instans DB terbesar dan termahal. Kebijakan ini juga mencegah pengguna membuat instans DB yang menggunakan IOPS yang Tersedia, yang menimbulkan biaya tambahan.

Izin yang secara tegas menolak lebih diprioritaskan daripada izin lain yang diberikan. Ini memastikan bahwa identitas tidak akan secara kebetulan mendapatkan izin yang tidak pernah ingin Anda berikan.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "DenyLargeCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "rds:DatabaseClass": [
                  "db.r3.8xlarge",
                  "db.m4.10xlarge"
               ]
            }
         }
      },
      {
         "Sid": "DenyPIOPSCreate",
         "Effect": "Deny",
         "Action": "rds:CreateDBInstance",
         "Resource": "*",
         "Condition": {
            "NumericNotEquals": {
```

```
                "rds:Piops": "0"
            }
         }
      }
   ]
}
```

Contoh 3: Membatasi kumpulan kunci dan nilai tag yang dapat digunakan untuk menandai sumber daya

Kebijakan berikut menggunakan kunci kondisi RDS dan memungkinkan penambahan tag dengan kunci `stage` untuk ditambahkan ke sumber daya dengan nilai `test`, `qa`, dan `production`.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource": "*",
         "Condition": {
            "streq": {
               "rds:req-tag/stage": [
                  "test",
                  "qa",
                  "production"
               ]
            }
         }
      }
   ]
}
```

Menentukan kondisi: Menggunakan tag kustom

Amazon Aurora mendukung penentuan kondisi dalam kebijakan IAM menggunakan tag kustom.

Sebagai contoh, misalkan Anda menambahkan tag bernama `environment` ke instans DB Anda dengan nilai seperti `beta`, `staging`, `production`, dan sebagainya. Jika melakukannya, Anda

dapat membuat kebijakan yang membatasi pengguna tertentu pada instans DB berdasarkan nilai tag `environment`.

> **Note**
>
> ID tag kustom bersifat peka huruf besar-kecil.

Tabel berikut mencantumkan ID tag RDS yang dapat digunakan pada elemen `Condition`.

ID tag RDS	Berlaku untuk
`db-tag`	Instans DB, termasuk replika baca
`snapshot-tag`	Snapshot DB
`ri-tag`	Instans DB terpesan
`og-tag`	Grup opsi DB
`pg-tag`	Grup parameter DB
`subgrp-tag`	Grup subnet DB
`es-tag`	Langganan peristiwa
`cluster-tag`	Klaster DB
`cluster-pg-tag`	Grup parameter klaster DB
`cluster-snapshot-tag`	Snapshot klaster DB

Sintaks untuk kondisi tag kustom adalah sebagai berikut:

```
"Condition":{"StringEquals":{"rds:rds-tag-identifier/tag-name":
["value"]} }
```

Misalnya, elemen `Condition` berikut berlaku untuk instans DB dengan tag bernama `environment` dan nilai tag `production`.

```
"Condition":{"StringEquals":{"rds:db-tag/environment": ["production"]} }
```

Untuk informasi tentang membuat tag, lihat Memberi tag pada sumber daya Amazon RDS.

Important

Jika Anda mengelola akses ke sumber daya RDS Anda menggunakan pemberian tag, sebaiknya Anda mengamankan akses ke tag untuk sumber daya RDS Anda. Anda dapat mengelola akses ke tag dengan membuat kebijakan untuk tindakan `AddTagsToResource` dan `RemoveTagsFromResource`. Misalnya, kebijakan berikut menolak kemampuan pengguna untuk menambahkan atau menghapus tag untuk semua sumber daya. Anda kemudian dapat membuat kebijakan untuk mengizinkan pengguna tertentu menambahkan atau menghapus tag.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyTagUpdates",
         "Effect":"Deny",
         "Action":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource":"*"
      }
   ]
}
```

Untuk melihat daftar tindakan Aurora, lihat Tindakan yang Ditentukan oleh Amazon RDS di Referensi Otorisasi Layanan.

Contoh kebijakan: Menggunakan tag kustom

Contoh berikut menunjukkan cara menggunakan tag kustom dalam kebijakan izin IAM Amazon Aurora. Untuk informasi lebih lanjut tentang cara menambahkan tag ke sumber daya Amazon Aurora, lihat Bekerja dengan Amazon Resource Name (ARN) di Amazon RDS.

Note

Semua contoh menggunakan wilayah us-west-2 dan berisi ID akun fiktif.

Contoh 1: Memberikan izin untuk tindakan pada sumber daya dengan tag tertentu dengan dua nilai yang berbeda

Kebijakan berikut memungkinkan izin untuk melakukan operasi `CreateDBSnapshot` API pada instans DB dengan tag `stage` diatur ke `development` atau `test`.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowAnySnapshotName",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:snapshot:*"
      },
      {
         "Sid":"AllowDevTestToCreateSnapshot",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBSnapshot"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

Kebijakan berikut memungkinkan izin untuk melakukan operasi `ModifyDBInstance` API pada instans DB dengan tag `stage` diatur ke `development` atau `test`.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
```

```
         "Sid":"AllowChangingParameterOptionSecurityGroups",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":" [
            "arn:aws:rds:*:123456789012:pg:*",
            "arn:aws:rds:*:123456789012:secgrp:*",
            "arn:aws:rds:*:123456789012:og:*"
         ]
      },
      {
         "Sid":"AllowDevTestToModifyInstance",
         "Effect":"Allow",
         "Action":[
            "rds:ModifyDBInstance"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:*",
         "Condition":{
            "StringEquals":{
                "rds:db-tag/stage":[
                  "development",
                  "test"
               ]
            }
         }
      }
   ]
}
```

Contoh 2: Secara eksplisit menolak izin untuk membuat instans DB yang menggunakan grup parameter DB yang ditentukan

Kebijakan berikut secara eksplisit menolak izin untuk membuat instans DB yang menggunakan grup parameter DB dengan nilai tag spesifik. Anda dapat menerapkan kebijakan ini jika Anda mengharuskan grup parameter DB yang dibuat pengguna tertentu selalu digunakan saat membuat instans DB. Kebijakan yang menggunakan `Deny` paling sering digunakan untuk membatasi akses yang diberikan oleh kebijakan yang lebih luas.

Izin yang secara tegas menolak lebih diprioritaskan daripada izin lain yang diberikan. Ini memastikan bahwa identitas tidak akan secara kebetulan mendapatkan izin yang tidak pernah ingin Anda berikan.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"DenyProductionCreate",
         "Effect":"Deny",
         "Action":"rds:CreateDBInstance",
         "Resource":"arn:aws:rds:*:123456789012:pg:*",
         "Condition":{
            "StringEquals":{
               "rds:pg-tag/usage":"prod"
            }
         }
      }
   ]
}
```

Contoh 3: Memberikan izin untuk tindakan pada instans DB dengan nama instans yang diawali dengan nama pengguna

Kebijakan berikut memungkinkan izin untuk memanggil API apa pun (kecuali untuk `AddTagsToResource` atau `RemoveTagsFromResource`) pada instans DB yang memiliki nama instans DB yang diawali dengan nama pengguna dan memiliki tag bernama `stage` yang sama dengan `devo` atau yang tidak memiliki tag bernama `stage`.

Baris `Resource` dalam kebijakan mengidentifikasi sumber daya berdasarkan Amazon Resource Name (ARN). Untuk informasi selengkapnya tentang cara menggunakan ARN dengan sumber daya Amazon Aurora, lihat Bekerja dengan Amazon Resource Name (ARN) di Amazon RDS.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"AllowFullDevAccessNoTags",
         "Effect":"Allow",
         "NotAction":[
            "rds:AddTagsToResource",
            "rds:RemoveTagsFromResource"
         ],
         "Resource":"arn:aws:rds:*:123456789012:db:${aws:username}*",
         "Condition":{
            "StringEqualsIfExists":{
```

```
                "rds:db-tag/stage":"devo"
            }
         }
      }
   ]
}
```

AWS kebijakan terkelola untuk Amazon RDS

Untuk menambahkan izin ke set dan peran izin, lebih mudah menggunakan kebijakan AWS terkelola daripada menulis kebijakan sendiri. Dibutuhkan waktu dan keahlian untuk membuat kebijakan yang dikelola pelanggan IAM yang hanya memberi tim Anda izin yang mereka butuhkan. Untuk memulai dengan cepat, Anda dapat menggunakan kebijakan AWS terkelola kami. Kebijakan ini mencakup kasus penggunaan umum dan tersedia di Akun AWS Anda. Untuk informasi selengkapnya tentang kebijakan AWS AWS terkelola, lihat kebijakan terkelola di Panduan Pengguna IAM.

Layanan AWS memelihara dan memperbarui kebijakan AWS terkelola. Anda tidak dapat mengubah izin dalam kebijakan AWS terkelola. Layanan terkadang menambahkan izin tambahan ke kebijakan AWS terkelola untuk mendukung fitur baru. Jenis pembaruan ini akan memengaruhi semua identitas (pengguna, grup, dan peran) di mana kebijakan tersebut dilampirkan. Layanan kemungkinan besar akan memperbarui kebijakan AWS terkelola saat fitur baru diluncurkan atau saat operasi baru tersedia. Layanan tidak menghapus izin dari kebijakan AWS terkelola, sehingga pembaruan kebijakan tidak merusak izin yang ada.

Selain itu, AWS mendukung kebijakan terkelola untuk fungsi pekerjaan yang mencakup beberapa layanan. Misalnya, kebijakan `ReadOnlyAccess` AWS terkelola menyediakan akses hanya-baca ke semua Layanan AWS dan sumber daya. Saat layanan meluncurkan fitur baru, AWS tambahkan izin hanya-baca untuk operasi dan sumber daya baru. Untuk melihat daftar dan deskripsi dari kebijakan fungsi tugas, lihat kebijakan yang dikelola AWS untuk fungsi tugas di Panduan Pengguna IAM.

Topik

- AWS kebijakan terkelola: AmazonRDS ReadOnlyAccess
- AWS kebijakan terkelola: AmazonRDS FullAccess
- AWS kebijakan terkelola: AmazonRDS DataFullAccess
- AWS kebijakan terkelola: AmazonRDS EnhancedMonitoringRole
- AWS kebijakan terkelola: AmazonRDS PerformanceInsightsReadOnly
- AWS kebijakan terkelola: AmazonRDS PerformanceInsightsFullAccess
- AWS kebijakan terkelola: AmazonRDS DirectoryServiceAccess
- AWS kebijakan terkelola: AmazonRDS ServiceRolePolicy

AWS kebijakan terkelola: AmazonRDS ReadOnlyAccess

Kebijakan ini memungkinkan akses hanya-baca ke Amazon RDS melalui. AWS Management Console

Detail izin

Kebijakan ini mencakup izin berikut:

- `rds` – Mengizinkan pengguna utama mendeskripsikan sumber daya Amazon RDS dan mencantumkan tag untuk sumber daya Amazon RDS.
- `cloudwatch`— Memungkinkan kepala sekolah untuk mendapatkan statistik metrik Amazon CloudWatch .
- `ec2` – Mengizinkan pengguna utama mendeskripsikan Zona Ketersediaan dan sumber daya jaringan.
- `logs`— Memungkinkan prinsipal untuk menggambarkan aliran CloudWatch log Log dari grup log, dan mendapatkan CloudWatch peristiwa log Log.
- `devops-guru`— Memungkinkan prinsipal untuk mendeskripsikan sumber daya yang memiliki cakupan Amazon DevOps Guru, yang ditentukan baik oleh nama CloudFormation tumpukan atau tag sumber daya.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON, lihat AmazonRDS ReadOnlyAccess di Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AmazonRDS FullAccess

Kebijakan ini menyediakan akses penuh ke Amazon RDS melalui. AWS Management Console

Detail izin

Kebijakan ini mencakup izin berikut:

- `rds` – Mengizinkan pengguna utama memiliki akses penuh ke Amazon RDS.
- `application-autoscaling` – Mengizinkan pengguna utama mendeskripsikan dan mengelola target dan kebijakan penskalaan Application Auto Scaling.
- `cloudwatch`— Memungkinkan kepala sekolah mendapatkan statika CloudWatch metrik dan mengelola alarm. CloudWatch

- `ec2` – Mengizinkan pengguna utama mendeskripsikan Zona Ketersediaan dan sumber daya jaringan.
- `logs`— Memungkinkan prinsipal untuk menggambarkan aliran CloudWatch log Log dari grup log, dan mendapatkan CloudWatch peristiwa log Log.
- `outposts`— Memungkinkan prinsipal untuk mendapatkan AWS Outposts jenis instance.
- `pi` – Mengizinkan pengguna utama untuk mendapatkan metrik Wawasan Performa.
- `sns` – Mengizinkan pengguna utama untuk berlangganan dan topik Amazon Simple Notification Service (Amazon SNS), dan menerbitkan pesan Amazon SNS.
- `devops-guru`— Memungkinkan prinsipal untuk mendeskripsikan sumber daya yang memiliki cakupan Amazon DevOps Guru, yang ditentukan baik oleh nama CloudFormation tumpukan atau tag sumber daya.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON, lihat AmazonRDS FullAccess di Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AmazonRDS DataFullAccess

Kebijakan ini memungkinkan akses penuh untuk menggunakan Data API dan editor kueri pada Aurora Serverless klaster tertentu Akun AWS. Kebijakan ini memungkinkan Akun AWS untuk mendapatkan nilai rahasia dari AWS Secrets Manager.

Anda dapat melampirkan kebijakan `AmazonRDSDataFullAccess` ke identitas IAM Anda.

Detail izin

Kebijakan ini mencakup izin berikut:

- `dbqms` – Mengizinkan pengguna utama mengakses, menghapus, mendeskripsikan, dan memperbarui kueri. Layanan Metadata Kueri basis data (`dbqms`) adalah layanan khusus internal. Ini memberikan kueri terbaru dan tersimpan Anda untuk editor kueri di AWS Management Console untuk beberapa Layanan AWS, termasuk Amazon RDS.
- `rds-data` – Mengizinkan pengguna utama untuk menjalankan pernyataan SQL pada basis data Aurora Serverless.
- `secretsmanager`— Memungkinkan kepala sekolah untuk mendapatkan nilai rahasia dari. AWS Secrets Manager

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON, lihat AmazonRDS DataFullAccess di Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AmazonRDS EnhancedMonitoringRole

Kebijakan ini menyediakan akses ke Amazon CloudWatch Logs untuk Amazon RDS Enhanced Monitoring.

Detail izin

Kebijakan ini mencakup izin berikut:

- `logs`— Memungkinkan prinsipal untuk membuat grup CloudWatch log Log dan kebijakan retensi, dan untuk membuat dan mendeskripsikan aliran CloudWatch log log dari grup log. Hal ini juga memungkinkan prinsipal untuk menempatkan dan mendapatkan peristiwa CloudWatch log Log.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON, lihat AmazonRDS EnhancedMonitoringRole di Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AmazonRDS PerformanceInsightsReadOnly

Kebijakan ini menyediakan akses hanya-baca ke Wawasan Performa Amazon RDS untuk instans DB Amazon RDS dan klaster DB Amazon Aurora.

Kebijakan ini kini mencakup `Sid` (ID pernyataan) sebagai pengidentifikasi pernyataan kebijakan.

Detail izin

Kebijakan ini mencakup izin berikut:

- `rds` – Mengizinkan pengguna utama mendeskripsikan instans DB Amazon RDS dan klaster DB Amazon Aurora.
- `pi` – Mengizinkan pengguna utama melakukan panggilan ke API Wawasan Performa Amazon RDS dan mengakses metrik Wawasan Performa.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON, lihat AmazonRDS PerformanceInsightsReadOnly di Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AmazonRDS PerformanceInsightsFullAccess

Kebijakan ini menyediakan akses penuh ke Wawasan Performa Amazon RDS untuk instans DB Amazon RDS dan klaster DB Amazon Aurora.

Kebijakan ini kini mencakup `Sid` (ID pernyataan) sebagai pengidentifikasi pernyataan kebijakan.

Detail izin

Kebijakan ini mencakup izin berikut:

- `rds` – Mengizinkan pengguna utama mendeskripsikan instans DB Amazon RDS dan klaster DB Amazon Aurora.
- `pi` – Mengizinkan pengguna utama melakukan panggilan ke API Wawasan Performa Amazon RDS, serta membuat, melihat, dan menghapus laporan analisis performa.
- `cloudwatch`— Memungkinkan kepala sekolah untuk membuat daftar semua metrik Amazon, dan mendapatkan CloudWatch data metrik dan statistik.

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON, lihat AmazonRDS PerformanceInsightsFullAccess di Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AmazonRDS DirectoryServiceAccess

Kebijakan ini mengizinkan Amazon RDS untuk melakukan panggilan ke AWS Directory Service.

Detail izin

Kebijakan ini mencakup izin berikut:

- `ds`— Memungkinkan kepala sekolah untuk mendeskripsikan AWS Directory Service direktori dan mengontrol otorisasi ke direktori. AWS Directory Service

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON, lihat AmazonRDS DirectoryServiceAccess di Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AmazonRDS ServiceRolePolicy

Anda tidak dapat melampirkan kebijakan `AmazonRDSServiceRolePolicy` ke entitas IAM Anda. Kebijakan ini dilampirkan ke peran tertaut layanan yang memungkinkan Amazon RDS melakukan

tindakan atas nama Anda. Untuk informasi selengkapnya, lihat Izin peran tertaut layanan untuk Amazon Aurora.

Amazon RDS memperbarui kebijakan AWS terkelola

Lihat detail tentang pembaruan kebijakan AWS terkelola untuk Amazon RDS sejak layanan ini mulai melacak perubahan ini. Untuk menerima peringatan otomatis tentang perubahan pada halaman ini, berlangganan umpan RSS di halaman Riwayat dokumen Amazon RDS.

Perubahan	Deskripsi	Tanggal
AWS kebijakan terkelola untuk Amazon RDS – Kebijakan baru	Amazon RDS menambahkan kebijakan terkelola baru yang diberi nama `AmazonRDSCustomInstanceProfileRolePolicy` untuk memungkinkan RDS Custom melakukan tindakan otomatisasi dan tugas manajemen database melalui profil instans EC2. Untuk informasi selengkapnya, lihat AWS kebijakan terkelola untuk Amazon RDS.	Februari 27, 2024
Izin peran tertaut layanan untuk Amazon Aurora – Pembaruan pada kebijakan yang sudah ada	Amazon RDS menambahkan ID pernyataan baru ke `AmazonRDSServiceRolePolicy` peran `AWSServiceRoleForRDS` terkait layanan. Untuk informasi selengkapnya, lihat Izin peran tertaut layanan untuk Amazon Aurora.	Januari 19, 2024
AWS kebijakan terkelola untuk Amazon RDS – Pembaruan pada kebijakan yang ada	Kebijakan terkelola `AmazonRDSPerformanceInsightsReadOnly` dan `AmazonRDSPerformanceInsightsFullAcce`	23 Oktober 2023

Perubahan	Deskripsi	Tanggal
	`ss` kini menyertakan `Sid` (ID pernyataan) sebagai pengidentifikasi dalam pernyataan kebijakan. Untuk informasi selengkapnya, lihat AWS kebijakan terkelola : AmazonRDS PerformanceInsightsReadOnly dan AWS kebijakan terkelola : AmazonRDS PerformanceInsightsFullAccess	
AWS kebijakan terkelola untuk Amazon RDS – Pembaruan pada kebijakan yang sudah ada	Amazon RDS menambahkan izin baru ke kebijakan terkelola `AmazonRDSFullAccess` . Izin ini memungkinkan Anda membuat, melihat, dan menghapus laporan analisis performa selama periode waktu tertentu. Untuk informasi selengkapnya tentang konfigurasi kebijakan akses untuk Wawasan Performa, lihat Mengonfigurasi kebijakan akses untuk Wawasan Performa	17 Agustus 2023

Perubahan	Deskripsi	Tanggal
AWS kebijakan terkelola untuk Amazon RDS – Kebijakan baru dan pembaruan pada kebijakan yang sudah ada	Amazon RDS menambahkan izin baru ke kebijakan terkelola `AmazonRDSPerformanceInsightsReadOnly` dan kebijakan terkelola baru bernama `AmazonRDSPerformanceInsightsFullAccess`. Izin ini memungkinkan Anda menganalisis Wawasan Performa selama periode tertentu, melihat hasil analisis beserta rekomendasinya, dan menghapus laporan. Untuk informasi selengkapnya tentang konfigurasi kebijakan akses untuk Wawasan Performa, lihat Mengonfigurasi kebijakan akses untuk Wawasan Performa	16 Agustus 2023

Perubahan	Deskripsi	Tanggal
AWS kebijakan terkelola untuk Amazon RDS – Pembaruan pada kebijakan yang sudah ada	Amazon RDS menambahkan CloudWatch namespace `ListMetrics` Amazon baru ke dan. `AmazonRDSFullAccess` `AmazonRDSReadOnlyAccess` Nama ruang ini diperlukan agar Amazon RDS dapat mencantumkan metrik penggunaan sumber daya tertentu. Untuk informasi selengkapnya, lihat Ringkasan mengelola izin akses ke CloudWatch sumber daya Anda di Panduan CloudWatch Pengguna Amazon.	4 April 2023

Perubahan	Deskripsi	Tanggal
Izin peran tertaut layanan untuk Amazon Aurora – Pembaruan pada kebijakan yang sudah ada	Amazon RDS menambahkan izin baru ke `AmazonRDSServiceRolePolicy` peran `AWSServiceRoleForRDS` terkait layanan untuk integrasi. AWS Secrets Manager RDS perlu berintegrasi dengan Secrets Manager untuk mengelola kata sandi pengguna utama di Secrets Manager. Rahasia menggunakan konvensi penamaan terpesan dan membatasi pembaruan pelanggan. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan dan AWS Secrets Manager.	22 Desember 2022

Perubahan	Deskripsi	Tanggal
AWS kebijakan terkelola untuk Amazon RDS – Pembaruan pada kebijakan yang ada	Amazon RDS menambahkan izin baru ke `AmazonRDSFullAccess` dan kebijakan `AmazonRDSReadOnlyAccess` terkelola untuk memungkinkan Anda mengaktifkan Amazon DevOps Guru di konsol RDS. Izin ini diperlukan untuk memeriksa apakah DevOps Guru dihidupkan. Untuk informasi selengkapnya, lihat Mengkonfigurasi kebijakan akses IAM untuk DevOps Guru untuk RDS.	19 Desember 2022
Izin peran tertaut layanan untuk Amazon Aurora – Pembaruan pada kebijakan yang sudah ada	Amazon RDS menambahkan CloudWatch namespace Amazon baru ke for. `AmazonRDSPreviewServiceRolePolicy` `PutMetricData` Nama ruang ini diperlukan agar Amazon RDS dapat menerbitkan metrik penggunaan sumber daya. Untuk informasi selengkapnya, lihat Menggunakan tombol kondisi untuk membatasi akses ke CloudWatch ruang nama di CloudWatch Panduan Pengguna Amazon.	7 Juni 2022

Perubahan	Deskripsi	Tanggal
Izin peran tertaut layanan untuk Amazon Aurora – Pembaruan pada kebijakan yang sudah ada	Amazon RDS menambahkan CloudWatch namespace Amazon baru ke for. `AmazonRDSBetaServiceRolePolicy` `PutMetricData` Nama ruang ini diperlukan agar Amazon RDS dapat menerbitkan metrik penggunaan sumber daya. Untuk informasi selengkapnya, lihat Menggunakan tombol kondisi untuk membatasi akses ke CloudWatch ruang nama di CloudWatch Panduan Pengguna Amazon.	7 Juni 2022
Izin peran tertaut layanan untuk Amazon Aurora – Pembaruan pada kebijakan yang sudah ada	Amazon RDS menambahkan CloudWatch namespace Amazon baru ke for. `AWSServiceRoleForRDS` `PutMetricData` Nama ruang ini diperlukan agar Amazon RDS dapat menerbitkan metrik penggunaan sumber daya. Untuk informasi selengkapnya, lihat Menggunakan tombol kondisi untuk membatasi akses ke CloudWatch ruang nama di CloudWatch Panduan Pengguna Amazon.	22 April 2022

Perubahan	Deskripsi	Tanggal
AWS kebijakan terkelola untuk Amazon RDS – Kebijakan baru	Amazon RDS menambahkan kebijakan terkelola baru yang diberi nama `AmazonRDSPerformanceInsightsReadOnly` untuk mengizinkan Amazon RDS memanggil AWS layanan atas nama instans DB Anda. Untuk informasi selengkapnya tentang konfigurasi kebijakan akses untuk Wawasan Performa, lihat Mengonfigurasi kebijakan akses untuk Wawasan Performa	10 Maret 2022
Izin peran tertaut layanan untuk Amazon Aurora – Pembaruan pada kebijakan yang sudah ada	Amazon RDS menambahkan CloudWatch ruang nama Amazon baru ke for. `AWSServiceRoleForRDS` `PutMetricData` Ruang nama ini diperlukan untuk Amazon DocumentDB (dengan kompatibilitas MongoDB) dan Amazon Neptune untuk menerbitkan metrik. CloudWatch Untuk informasi selengkapnya, lihat Menggunakan tombol kondisi untuk membatasi akses ke CloudWatch ruang nama di CloudWatch Panduan Pengguna Amazon.	4 Maret 2022

Perubahan	Deskripsi	Tanggal
Amazon RDS mulai melacak perubahan	Amazon RDS mulai melacak perubahan untuk kebijakan yang AWS dikelola.	26 Oktober 2021

Mencegah masalah confused deputy lintas layanan

Masalah confused deputy adalah masalah keamanan saat entitas yang tidak memiliki izin untuk melakukan suatu tindakan dapat memaksa entitas yang lebih berhak untuk melakukan tindakan tersebut. Di AWS, peniruan identitas lintas layanan dapat mengakibatkan masalah confused deputy.

Peniruan identitas lintas layanan dapat terjadi ketika satu layanan (layanan pemanggil) memanggil layanan lain (layanan yang dipanggil). Layanan panggilan dapat dimanipulasi agar menggunakan izinnya untuk bertindak pada sumber daya pelanggan lain dengan cara yang seharusnya tidak diizinkan untuk mengakses. Untuk mencegah hal ini, AWS menyediakan alat yang dapat membantu Anda melindungi data untuk semua layanan dengan pengguna utama layanan yang telah diberi akses ke sumber daya di akun Anda. Lihat informasi selengkapnya di Masalah confused deputy dalam Panduan Pengguna IAM.

Untuk membatasi izin yang diberikan Amazon RDS kepada layanan lain untuk sumber daya tertentu, sebaiknya gunakan kunci konteks kondisi global `aws:SourceArn` dan `aws:SourceAccount` dalam kebijakan sumber daya.

Dalam beberapa kasus, nilai `aws:SourceArn` tidak berisi ID akun, misalnya saat Anda menggunakan Amazon Resource Name (ARN) untuk bucket Amazon S3. Dalam kasus ini, pastikan untuk menggunakan kedua kunci konteks kondisi global untuk membatasi izin. Dalam beberapa kasus, Anda menggunakan kunci konteks kondisi global dan nilai `aws:SourceArn` yang berisi ID akun. Dalam kasus ini, pastikan bahwa nilai `aws:SourceAccount` dan akun dalam nilai `aws:SourceArn` menggunakan ID akun yang sama ketika digunakan dalam pernyataan yang sama. Jika Anda ingin hanya satu sumber daya yang akan dikaitkan dengan akses lintas layanan, gunakan `aws:SourceArn`. Jika Anda ingin mengizinkan semua sumber daya di akun AWS yang ditentukan dikaitkan dengan penggunaan lintas layanan, gunakan `aws:SourceAccount`.

Pastikan nilai `aws:SourceArn` adalah ARN untuk jenis sumber daya Amazon RDS. Lihat informasi selengkapnya di Bekerja dengan Amazon Resource Name (ARN) di Amazon RDS.

Cara paling efektif untuk melindungi dari masalah confused deputy adalah dengan menggunakan kunci konteks kondisi global `aws:SourceArn` dengan ARN sumber daya penuh. Dalam beberapa kasus, Anda mungkin tidak mengetahui ARN lengkap sumber daya atau mungkin Anda menentukan beberapa sumber daya. Dalam kasus ini, gunakan kunci kondisi konteks global `aws:SourceArn` dengan wildcard (*) untuk bagian ARN yang tidak diketahui. Contohnya adalah `arn:aws:rds:*:123456789012:*`.

Contoh berikut menunjukkan bagaimana Anda dapat menggunakan `aws:SourceArn` dan kunci konteks kondisi global `aws:SourceAccount` di Amazon RDS untuk mencegah masalah confused deputy.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "rds.amazonaws.com"
    },
    "Action": "sts:AssumeRole",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:rds:us-east-1:123456789012:db:mydbinstance"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

Untuk contoh kebijakan lainnya yang menggunakan kunci konteks kondisi global `aws:SourceArn` dan `aws:SourceAccount`, lihat bagian berikut:

- Memberikan izin untuk menerbitkan pemberitahuan ke topik Amazon SNS
- Mengatur akses ke bucket Amazon S3 (Impor PostgreSQL)
- Menyiapkan akses ke bucket Amazon S3 (Ekspor PostgreSQL)

Autentikasi basis data IAM

Anda dapat mengautentikasi ke instans DB menggunakan autentikasi basis data AWS Identity and Access Management (IAM). Autentikasi basis data IAM bekerja dengan Aurora MySQL, dan Aurora PostgreSQL. Dengan metode autentikasi ini, Anda tidak perlu menggunakan kata sandi saat terhubung ke instans DB. Sebagai gantinya, Anda menggunakan token autentikasi.

Token autentikasi adalah string karakter unik yang dihasilkan Amazon Aurora atas permintaan. Token autentikasi dibuat menggunakan AWS Signature Versi 4. Setiap token memiliki masa pakai 15 menit. Anda tidak perlu menyimpan kredensial pengguna dalam basis data, karena autentikasi dikelola secara eksternal menggunakan IAM. Anda juga masih dapat menggunakan autentikasi basis data standar. Token hanya digunakan untuk autentikasi dan tidak memengaruhi sesi setelah dibuat.

Autentikasi basis data IAM memberikan manfaat berikut:

- Lalu lintas jaringan ke dan dari basis data dienkripsi menggunakan Lapisan Soket Aman (SSL) atau Keamanan Lapisan Pengangkutan (TLS). Untuk informasi selengkapnya tentang cara menggunakan SSL/TLS bersama Amazon Aurora, lihat .
- Anda dapat menggunakan IAM untuk mengelola akses ke sumber daya basis data Anda secara terpusat, bukan mengelola akses satu per satu pada instans DB.
- Untuk aplikasi yang berjalan di Amazon EC2, Anda dapat menggunakan kredensial profil khusus untuk instans EC2 untuk mengakses basis data, bukan menggunakan kata sandi, untuk keamanan yang lebih baik.

Secara umum, pertimbangkan untuk menggunakan autentikasi basis data IAM saat aplikasi Anda membuat kurang dari 200 koneksi per detik, dan Anda tidak ingin mengelola nama pengguna dan kata sandi secara langsung dalam kode aplikasi Anda.

Driver JDBC AWS untuk MySQL mendukung autentikasi basis data IAM. Untuk informasi selengkapnya, lihat Autentikasi Database AWS IAM di AWS Driver JDBC untuk repositori MySQL. GitHub

Topik

- Ketersediaan Wilayah dan versi
- Dukungan CLI dan SDK
- Batasan untuk autentikasi basis data IAM
- Rekomendasi untuk autentikasi basis data IAM

- Kunci konteks kondisi global AWS yang tidak didukung
- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM
- Menghubungkan ke instans DB menggunakan autentikasi IAM

Ketersediaan Wilayah dan versi

Ketersediaan fitur dan dukungan bervariasi di seluruh versi khusus dari setiap mesin basis data Aurora, dan di seluruh Wilayah AWS. Untuk informasi selengkapnya tentang ketersediaan versi dan Wilayah dengan autentikasi basis data IAM dan Aurora, lihat. Autentikasi basis data IAM di Aurora

Untuk Aurora MySQL, semua kelas instans DB yang didukung mendukung autentikasi basis data IAM, kecuali untuk db.t2.small dan db.t3.small. Untuk informasi tentang kelas instans DB yang didukung, lihat Mesin DB yang didukung untuk kelas instans DB.

Dukungan CLI dan SDK

Autentikasi basis data IAM tersedia untuk AWS CLI dan untuk SDK AWS khusus bahasa berikut:

- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java
- AWS SDK for JavaScript
- AWS SDK for PHP
- AWS SDK for Python (Boto3)
- AWS SDK for Ruby

Batasan untuk autentikasi basis data IAM

Saat menggunakan autentikasi basis data IAM, batasan berikut berlaku:

- Jumlah maksimum koneksi per detik untuk instans DB Anda mungkin akan dibatasi, bergantung pada kelas instans DB-nya dan beban kerja Anda. Otentikasi IAM dapat gagal jika terjadi kehabisan sumber daya selama beban DB puncak.

- Saat ini, autentikasi basis data IAM tidak mendukung kunci konteks kondisi global.

  Untuk informasi selengkapnya tentang kunci konteks kondisi global, lihat AWS kunci konteks kondisi global dalam Panduan Pengguna IAM.
- Untuk PostgreSQL, jika peran IAM (`rds_iam`) ditambahkan ke pengguna (termasuk pengguna master RDS), autentikasi IAM diprioritaskan atas autentikasi kata sandi, sehingga pengguna harus login sebagai pengguna IAM.
- Untuk Aurora PostgreSQL, Anda tidak dapat menggunakan autentikasi IAM untuk membuat koneksi replikasi.
- Anda tidak dapat menggunakan data DNS Route 53 kustom sebagai pengganti titik akhir instans DB untuk menghasilkan token autentikasi.

Rekomendasi untuk autentikasi basis data IAM

Kami merekomendasikan hal berikut saat menggunakan autentikasi basis data IAM:

- Gunakan autentikasi basis data IAM saat aplikasi Anda membutuhkan kurang dari 200 koneksi autentikasi basis data IAM baru per detik.

  Mesin basis data yang bekerja dengan Amazon Aurora tidak memberlakukan batasan apa pun pada upaya autentikasi per detik. Namun, saat Anda menggunakan autentikasi basis data IAM, aplikasi Anda harus membuat token autentikasi. Aplikasi Anda kemudian menggunakan token tersebut untuk terhubung ke instans DB. Jika Anda melebihi batas maksimum untuk koneksi baru per detik, maka overhead tambahan dari autentikasi basis data IAM dapat menyebabkan throttling koneksi.

  Pertimbangkan untuk menggunakan penyatuan koneksi di aplikasi Anda untuk memitigasi pembuatan koneksi yang konstan. Cara ini dapat mengurangi overhead dari autentikasi DB IAM dan memungkinkan aplikasi Anda menggunakan kembali koneksi yang ada. Atau, pertimbangkan untuk menggunakan Proksi RDS untuk kasus penggunaan ini. Proksi RDS memiliki biaya tambahan. Lihat Harga Proksi RDS.
- Ukuran token autentikasi basis data IAM bergantung pada banyak hal termasuk jumlah tag IAM, kebijakan layanan IAM, panjang ARN, serta properti basis data dan IAM lainnya. Ukuran minimum token ini umumnya sekitar 1 KB tetapi bisa saja lebih besar. Karena token ini digunakan sebagai kata sandi dalam string koneksi ke basis data yang menggunakan autentikasi IAM, Anda harus memastikan bahwa driver basis data Anda (misalnya, ODBC) dan/atau alat apa pun

tidak membatasi atau memotong token ini dikarenakan ukurannya. Token yang terpotong akan menyebabkan kegagalan validasi autentikasi oleh basis data dan IAM.

- Jika Anda menggunakan kredensial temporer saat membuat token autentikasi basis data IAM, kredensial temporer masih harus valid saat menggunakan token autentikasi basis data IAM untuk membuat permintaan koneksi.

Kunci konteks kondisi global AWS yang tidak didukung

Autentikasi basis data IAM tidak mendukung subset kunci konteks kondisi global AWS berikut.

- `aws:Referer`
- `aws:SourceIp`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserAgent`
- `aws:VpcSourceIp`

Untuk informasi selengkapnya, lihat kunci konteks kondisi global AWS dalam Panduan Pengguna IAM.

Mengaktifkan dan menonaktifkan autentikasi basis data IAM

Secara default, autentikasi basis data IAM dinonaktifkan di instans DB. Anda dapat mengaktifkan atau menonaktifkan autentikasi basis data IAM menggunakan AWS Management Console, AWS CLI, atau API.

Anda dapat mengaktifkan autentikasi basis data IAM saat Anda melakukan salah satu tindakan berikut:

- Untuk membuat klaster DB yang baru dengan autentikasi basis data IAM diaktifkan, lihat Membuat klaster DB Amazon Aurora.
- Untuk memodifikasi klaster DB untuk mengaktifkan autentikasi basis data IAM, lihat Memodifikasi klaster DB Amazon Aurora.
- Untuk memulihkan klaster DB dari snapshot dengan autentikasi basis data IAM diaktifkan, lihat Memulihkan dari snapshot klaster DB.

- Untuk memulihkan klaster DB ke titik waktu dengan autentikasi basis data IAM diaktifkan, lihat Memulihkan klaster DB ke waktu tertentu.

Konsol

Setiap alur kerja pembuatan atau modifikasi memiliki bagian Autentikasi basis data, tempat Anda dapat mengaktifkan atau menonaktifkan autentikasi basis data IAM. Di bagian tersebut, pilih Autentikasi kata sandi dan basis data IAM untuk mengaktifkan autentikasi basis data IAM.

Untuk mengaktifkan atau menonaktifkan autentikasi basis data IAM untuk instans DB yang ada

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Basis data.
3. Pilih instans DB yang ingin Anda modifikasi.

> **Note**
>
> Anda hanya dapat mengaktifkan autentikasi IAM jika semua instans DB dalam klaster DB kompatibel dengan IAM. Periksa persyaratan kompatibilitas di Ketersediaan Wilayah dan versi.

4. Pilih Modifikasi.
5. Di bagian Autentikasi basis data, pilih Autentikasi kata sandi dan basis data IAM untuk mengaktifkan autentikasi basis data IAM. Pilih Autentikasi kata sandi atau Autentikasi kata sandi dan Kerberos untuk menonaktifkan autentikasi IAM.
6. Pilih Lanjutkan.
7. Untuk menerapkan perubahan secara langsung, pilih Langsung di bagian Penjadwalan perubahan.
8. Pilih Ubah klaster.

AWS CLI

Untuk membuat klaster DB baru dengan autentikasi IAM menggunakan AWS CLI, gunakan perintah `create-db-cluster`. Tentukan opsi `--enable-iam-database-authentication`.

Untuk memperbarui klaster DB yang ada agar memiliki atau tidak memiliki autentikasi IAM, gunakan perintah AWS CLI `modify-db-cluster`. Tentukan opsi `--enable-iam-database-authentication` atau `--no-enable-iam-database-authentication`, sesuai kebutuhan.

> **Note**
>
> Anda hanya dapat mengaktifkan autentikasi IAM jika semua instans DB dalam klaster DB kompatibel dengan IAM. Periksa persyaratan kompatibilitas di Ketersediaan Wilayah dan versi.

Secara default, Aurora melakukan modifikasi selama periode pemeliharaan berikutnya. Jika Anda ingin menggantinya dan mengaktifkan autentikasi DB IAM sesegera mungkin, gunakan parameter `--apply-immediately`.

Jika Anda memulihkan instans DB, gunakan salah satu perintah AWS CLI berikut:

- `restore-db-cluster-to-point-in-time`
- `restore-db-cluster-from-db-snapshot`

Pengaturan autentikasi basis data IAM ditetapkan ke pengaturan snapshot sumber secara default. Untuk mengubah pengaturan ini, tetapkan opsi `--enable-iam-database-authentication` atau `--no-enable-iam-database-authentication` sebagaimana diperlukan.

API RDS

Untuk membuat instans DB baru dengan autentikasi IAM dengan menggunakan API, gunakan pengoperasian API `CreateDBCluster`. Tetapkan parameter `EnableIAMDatabaseAuthentication` ke `true`.

Untuk memperbarui klaster DB yang ada agar memiliki autentikasi IAM, gunakan pengoperasian API `ModifyDBCluster`. Tetapkan parameter `EnableIAMDatabaseAuthentication` ke `true` untuk mengaktifkan autentikasi IAM, atau `false` untuk menonaktifkannya.

> **Note**
>
> Anda hanya dapat mengaktifkan autentikasi IAM jika semua instans DB dalam klaster DB kompatibel dengan IAM. Periksa persyaratan kompatibilitas di Ketersediaan Wilayah dan versi.

Jika Anda memulihkan instans DB, gunakan salah satu pengoperasian API berikut:

- `RestoreDBClusterFromSnapshot`
- `RestoreDBClusterToPointInTime`

Pengaturan autentikasi basis data IAM ditetapkan ke pengaturan snapshot sumber secara default. Untuk mengubah pengaturan ini, tetapkan parameter `EnableIAMDatabaseAuthentication` ke `true` untuk mengaktifkan autentikasi IAM, atau `false` untuk menonaktifkannya.

Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM

Agar pengguna atau peran dapat terhubung ke instans DB, Anda harus membuat kebijakan IAM. Setelah itu, Anda melampirkan kebijakan ke peran atau kumpulan izin.

> **Note**
>
> Untuk mempelajari lebih lanjut tentang kebijakan IAM, lihat Manajemen identitas dan akses untuk Amazon Aurora.

Contoh kebijakan berikut memungkinkan pengguna untuk terhubung ke instans DB menggunakan autentikasi basis data IAM.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
```

```
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:1234567890:dbuser:cluster-ABCDEFGHIJKL01234/
db_user"
            ]
        }
    ]
}
```

Important

Pengguna dengan izin administrator dapat mengakses instans DB tanpa izin eksplisit dalam kebijakan IAM. Jika Anda ingin membatasi akses administrator ke instans DB, Anda dapat membuat peran IAM dengan izin istimewa minimum yang sesuai dan menetapkannya ke administrator.

Note

Jangan samakan awalan `rds-db:` dengan awalan pengoperasian API RDS lain yang dimulai dengan `rds:`. Anda menggunakan awalan `rds-db:` dan tindakan `rds-db:connect` hanya untuk autentikasi basis data IAM. Hal ini tidak berlaku dalam konteks lainnya.

Contoh kebijakan tersebut mencakup pernyataan tunggal dengan elemen berikut:

- `Effect` – Tentukan `Allow` untuk memberikan akses ke instans DB. Jika Anda tidak secara eksplisit mengizinkan akses, maka akses ditolak secara default.
- `Action` – Tentukan `rds-db:connect` untuk mengizinkan koneksi ke instans DB.
- `Resource` – Tentukan Amazon Resource Name (ARN) yang menjelaskan satu akun basis data dalam satu instans DB. Format ARN adalah sebagai berikut.

  ```
  arn:aws:rds-db:region:account-id:dbuser:DbClusterResourceId/db-user-name
  ```

  Dalam format ini, ganti hal berikut:

- *region* adalah Wilayah AWS untuk instans DB. Dalam contoh kebijakan, Wilayah AWS adalah `us-east-2`.
- *account-id* adalah nomor akun AWS untuk instans DB. Dalam contoh kebijakan, nomor akun adalah `1234567890`. Pengguna harus berada dalam akun yang sama dengan akun untuk klaster DB.

  Untuk melakukan akses lintas akun, buat peran IAM dengan kebijakan yang ditunjukkan di atas dalam akun untuk klaster DB dan izinkan akun Anda yang lain untuk mengambil peran tersebut.
- *DbClusterResourceId* adalah pengidentifikasi untuk instans DB. Pengidentifikasi ini bersifat unik untuk Wilayah AWS dan tidak pernah berubah. Dalam contoh kebijakan, pengidentifikasinya adalah `cluster-ABCDEFGHIJKL01234`.

  Untuk menemukan ID sumber daya instans DB di AWS Management Console untuk Amazon Aurora, pilih instans DB untuk melihat detailnya. Lalu pilih tab Konfigurasi. ID Sumber Daya ditampilkan di bagian Konfigurasi.

  Atau, Anda dapat menggunakan perintah AWS CLI untuk mencantumkan pengidentifikasi dan ID sumber daya untuk semua instans DB Anda di Wilayah AWS saat ini, seperti yang ditunjukkan berikut.

  ```
  aws rds describe-db-clusters --query "DBClusters[*].
  [DBClusterIdentifier,DbClusterResourceId]"
  ```

  > **Note**
  >
  > Jika Anda terhubung ke basis data melalui Proksi RDS, sebutkan ID sumber daya proksinya, seperti `prx-ABCDEFGHIJKL01234`. Untuk informasi tentang cara menggunakan autentikasi basis data IAM dengan Proksi RDS, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.
- *db-user-name* adalah nama akun basis data untuk dikaitkan dengan autentikasi IAM. Dalam contoh kebijakan, akun basis datanya adalah `db_user`.

Anda dapat membuat ARN lain untuk mendukung berbagai pola akses. Kebijakan berikut memungkinkan akses ke dua akun basis data yang berbeda dalam instans DB.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-ABCDEFGHIJKL01234/
jane_doe",
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-ABCDEFGHIJKL01234/
mary_roe"
         ]
      }
   ]
}
```

Kebijakan berikut menggunakan karakter "*" untuk mencocokkan semua instans DB dan akun basis data untuk akun AWS tertentu dan Wilayah AWS.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:1234567890:dbuser:*/*"
            ]
        }
    ]
}
```

Kebijakan berikut sesuai dengan semua instans DB untuk akun AWS tertentu dan Wilayah AWS. Namun, kebijakan hanya memberikan akses ke instans DB yang memiliki akun basis data `jane_doe`.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
             "rds-db:connect"
         ],
         "Resource": [
             "arn:aws:rds-db:us-east-2:123456789012:dbuser:*/jane_doe"
         ]
      }
   ]
}
```

Pengguna atau peran hanya memiliki akses ke basis data yang aksesnya dimiliki oleh pengguna basis data tersebut. Misalnya, instans DB Anda memiliki basis data bernama dev, dan basis data lain bernama test. Jika pengguna basis data `jane_doe` hanya memiliki akses ke dev, setiap pengguna atau peran yang mengakses instans DB tersebut dengan pengguna `jane_doe` juga hanya akan memiliki akses ke dev. Pembatasan akses ini juga berlaku untuk objek basis data lain, seperti tabel, tampilan, dan sebagainya.

Administrator harus membuat kebijakan IAM yang memberi izin kepada entitas untuk menjalankan pengoperasian API tertentu pada sumber daya tertentu yang diperlukan. Administrator kemudian dapat melampirkan kebijakan tersebut ke peran atau kumpulan izin yang memerlukan izin tersebut. Untuk contoh kebijakan, lihat Contoh kebijakan berbasis identitas untuk Amazon Aurora.

Melampirkan kebijakan IAM ke peran atau kumpulan izin

Setelah membuat kebijakan IAM untuk memungkinkan autentikasi basis data, Anda perlu melampirkan kebijakan ke peran atau kumpulan izin. Untuk tutorial tentang topik ini, lihat Membuat dan melampirkan kebijakan yang dikelola pelanggan pertama Anda dalam Panduan Pengguna IAM.

Saat mempelajari tutorial ini, Anda dapat menggunakan salah satu contoh kebijakan yang ditunjukkan dalam bagian ini sebagai titik awal dan menyesuaikannya dengan kebutuhan Anda. Di

akhir tutorial, Anda memiliki kumpulan izin dengan kebijakan terlampir yang dapat memanfaatkan tindakan `rds-db:connect`.

> **Note**
>
> Anda dapat memetakan beberapa peran dan kumpulan izin ke akun pengguna basis data yang sama. Misalnya, kebijakan IAM Anda telah menentukan ARN sumber daya berikut.
>
> ```
> arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-12ABC34DEFG5HIJ6KLMNOP78QR/
> jane_doe
> ```
>
> Jika Anda melampirkan kebijakan ke Jane, Bob, dan Diego, maka masing-masing pengguna tersebut dapat terhubung ke klaster DB yang telah ditentukan menggunakan akun basis data `jane_doe`.

Membuat akun basis data menggunakan autentikasi IAM

Dengan autentikasi basis data IAM, Anda tidak perlu menetapkan kata sandi basis data ke akun pengguna yang Anda buat. Jika Anda menghapus pengguna yang dipetakan ke akun basis data, Anda juga harus menghapus akun basis data dengan pernyataan `DROP USER`.

> **Note**
>
> Nama pengguna yang digunakan untuk autentikasi IAM harus sesuai dengan besar huruf/kecil nama pengguna dalam basis data.

Topik

- Menggunakan autentikasi IAM dengan Aurora MySQL
- Menggunakan autentikasi IAM dengan Aurora PostgreSQL

Menggunakan autentikasi IAM dengan Aurora MySQL

Dengan Aurora MySQL, autentikasi ditangani oleh `AWSAuthenticationPlugin`—plugin yang disediakan AWS yang bekerja secara lancar dengan IAM untuk mengautentikasi pengguna Anda.

Hubungkan ke instans DB sebagai pengguna master atau pengguna lain yang dapat membuat pengguna dan memberikan hak istimewa. Setelah terhubung, keluarkan pernyataan `CREATE USER`, seperti yang ditunjukkan pada contoh berikut.

```
CREATE USER jane_doe IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
```

Klausul `IDENTIFIED WITH` memungkinkan Aurora MySQL menggunakan `AWSAuthenticationPlugin` untuk mengautentikasi akun basis data (`jane_doe`). Klausul `AS 'RDS'` mengacu pada metode autentikasi. Pastikan nama pengguna basis data yang ditentukan sama dengan sumber daya dalam kebijakan IAM untuk akses basis data IAM. Untuk informasi selengkapnya, lihat Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM.

> **Note**
>
> Jika Anda melihat pesan berikut, artinya plugin yang disediakan AWS tidak tersedia untuk instans DB saat ini.
> `ERROR 1524 (HY000): Plugin 'AWSAuthenticationPlugin' is not loaded`
> Untuk memecahkan masalah kesalahan ini, verifikasi bahwa Anda menggunakan konfigurasi yang didukung dan telah mengaktifkan autentikasi basis data IAM di instans DB Anda. Untuk informasi selengkapnya, lihat Ketersediaan Wilayah dan versi dan Mengaktifkan dan menonaktifkan autentikasi basis data IAM.

Setelah membuat akun menggunakan `AWSAuthenticationPlugin`, Anda mengelolanya dengan cara yang sama seperti akun basis data lainnya. Misalnya, Anda dapat memodifikasi hak istimewa akun dengan pernyataan `GRANT` dan `REVOKE`, atau memodifikasi berbagai atribut akun dengan pernyataan `ALTER USER`.

Lalu lintas jaringan basis data dienkripsi menggunakan SSL/TLS saat menggunakan IAM. Untuk mengizinkan koneksi SSL, modifikasi akun pengguna dengan perintah berikut.

```
ALTER USER 'jane_doe'@'%' REQUIRE SSL;
```

Menggunakan autentikasi IAM dengan Aurora PostgreSQL

Untuk menggunakan autentikasi IAM dengan Aurora PostgreSQL, hubungkan ke instans DB sebagai pengguna master atau pengguna lain yang dapat membuat pengguna dan memberikan hak

istimewa. Setelah terhubung, buat pengguna basis data lalu beri mereka peran `rds_iam` seperti yang ditunjukkan pada contoh berikut.

```
CREATE USER db_userx;
GRANT rds_iam TO db_userx;
```

Pastikan nama pengguna basis data yang ditentukan sama dengan sumber daya dalam kebijakan IAM untuk akses basis data IAM. Untuk informasi selengkapnya, lihat Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM.

Perhatikan bahwa pengguna basis data PostgreSQL dapat menggunakan autentikasi IAM atau Kerberos tetapi tidak keduanya, sehingga pengguna ini juga tidak dapat memiliki peran `rds_ad`. Hal ini juga berlaku untuk keanggotaan bersarang. Untuk informasi selengkapnya, lihat Langkah 7: Buat pengguna PostgreSQL untuk pengguna utama Kerberos Anda .

Menghubungkan ke instans DB menggunakan autentikasi IAM

Dengan autentikasi basis data IAM, Anda menggunakan token autentikasi ketika Anda menghubungkan ke instans DB. Token autentikasi adalah string karakter yang Anda gunakan, bukannya kata sandi. Setelah Anda dibuat, token autentikasi berlaku selama 15 menit. Jika Anda mencoba menghubungkan menggunakan token yang tidak berlaku lagi, permintaan koneksi akan ditolak.

Setiap token autentikasi harus disertai dengan tanda tangan yang valid, menggunakan AWS signature versi 4. (Untuk informasi selengkapnya, lihat Proses penandatanganan Signature Versi 4 di Referensi Umum AWS.) AWS CLI dan SDK AWS, seperti AWS SDK for Java atau AWS SDK for Python (Boto3), dapat secara otomatis menandatangani setiap token yang Anda buat.

Anda dapat menggunakan token autentikasi saat menghubungkan ke Amazon Aurora dari layanan AWS lainnya, seperti AWS Lambda. Dengan menggunakan token, Anda tidak perlu memasukkan kata sandi dalam kode Anda. Atau, Anda dapat menggunakan SDK AWS untuk membuat dan menandatangani token autentikasi secara terprogram.

Setelah Anda memiliki token autentikasi IAM yang telah ditandatangani, Anda dapat terhubung ke klaster DB Aurora. Setelah itu, Anda dapat menemukan cara melakukannya menggunakan alat baris perintah atau SDK AWS, seperti AWS SDK for Java atau AWS SDK for Python (Boto3).

Untuk informasi selengkapnya, lihat postingan blog berikut ini:

- Gunakan autentikasi IAM untuk terhubung dengan SQL Workbench/J ke Aurora MySQL atau Amazon RDS for MySQL
- Menggunakan autentikasi IAM untuk terhubung dengan pgAdmin Amazon Aurora PostgreSQL atau Amazon RDS for PostgreSQL

Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke instans DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM

Topik

- Menghubungkan ke instans DB menggunakan autentikasi IAM dari baris perintah: AWS CLI dan klien mysql
- Menghubungkan ke instans DB Anda menggunakan autentikasi IAM dari baris perintah: AWS CLI dan klien psql
- Menghubungkan ke klaster DB menggunakan autentikasi IAM dan AWS SDK for .NET
- Menghubungkan ke instans DB menggunakan autentikasi IAM dan AWS SDK for Go
- Menghubungkan ke klaster DB menggunakan autentikasi IAM dan AWS SDK for Java
- Menghubungkan ke klaster DB menggunakan autentikasi IAM dan AWS SDK for Python (Boto3)

Menghubungkan ke instans DB menggunakan autentikasi IAM dari baris perintah: AWS CLI dan klien mysql

Anda dapat terhubung dari baris perintah ke klaster DB Aurora dengan alat baris perintah AWS CLI dan `mysql` seperti yang dijelaskan berikut ini.

Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke instans DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM

- Membuat akun basis data menggunakan autentikasi IAM

> **Note**
>
> Untuk informasi tentang cara menghubungkan ke basis data menggunakan SQL Workbench/J dengan autentikasi IAM, lihat postingan blog Menggunakan autentikasi IAM untuk terhubung dengan SQL Workbench/J ke Aurora MySQL atau Amazon RDS for MySQL.

Topik

- Membuat token autentikasi IAM
- Menghubungkan ke instans DB

Membuat token autentikasi IAM

Contoh berikut menunjukkan cara mendapatkan token autentikasi yang ditandatangani menggunakan AWS CLI.

```
aws rds generate-db-auth-token \
   --hostname rdsmysql.123456789012.us-west-2.rds.amazonaws.com \
   --port 3306 \
   --region us-west-2 \
   --username jane_doe
```

Dalam contoh, parameternya adalah sebagai berikut:

- `--hostname` – Nama host instans DB yang ingin Anda akses
- `--port` – Nomor port yang digunakan untuk menghubungkan ke instans DB Anda
- `--region` – Wilayah AWS tempat instans DB beroperasi
- `--username` – Akun basis data yang ingin Anda akses

Beberapa karakter pertama dari token terlihat seperti berikut.

```
rdsmysql.123456789012.us-west-2.rds.amazonaws.com:3306/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir instans DB untuk menghasilkan token autentikasi.

Menghubungkan ke instans DB

Format umum untuk terhubung ditampilkan sebagai berikut.

```
mysql --host=hostName --port=portNumber --ssl-ca=full_path_to_ssl_certificate --enable-cleartext-plugin --user=userName --password=authToken
```

Parameternya adalah sebagai berikut:

- `--host` – Nama host instans DB yang ingin Anda akses
- `--port` – Nomor port yang digunakan untuk menghubungkan ke instans DB Anda
- `--ssl-ca` – Jalur lengkap ke file sertifikat SSL yang berisi kunci publik

  Untuk informasi selengkapnya, lihat Menggunakan TLS dengan klaster DB Aurora MySQL.

  Untuk mengunduh sertifikat SSL, lihat .
- `--enable-cleartext-plugin` – Nilai yang menentukan bahwa `AWSAuthenticationPlugin` harus digunakan untuk koneksi ini

  Jika Anda menggunakan klien MariaDB, opsi `--enable-cleartext-plugin` tidak diperlukan.
- `--user` – Akun basis data yang ingin Anda akses
- `--password` – Token autentikasi IAM yang ditandatangani

Token autentikasi terdiri atas ratusan karakter. Ini dapat memberatkan baris perintah. Salah satu cara untuk mengatasi ini adalah dengan menyimpan token ke variabel lingkungan, lalu menggunakan variabel tersebut saat Anda terhubung. Contoh berikut menunjukkan satu cara untuk melakukan solusi ini. Pada contoh ini, */sample_dir/* adalah jalur lengkap ke file sertifikat SSL yang berisi kunci publik.

```
RDSHOST="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
```

```
TOKEN="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 3306 --region us-
west-2 --username jane_doe )"

mysql --host=$RDSHOST --port=3306 --ssl-ca=/sample_dir/global-bundle.pem --enable-
cleartext-plugin --user=jane_doe --password=$TOKEN
```

Saat Anda terhubung menggunakan `AWSAuthenticationPlugin`, koneksi diamankan menggunakan SSL. Untuk memverifikasi hal ini, ketik berikut ini di prompt perintah `mysql>`.

```
show status like 'Ssl%';
```

Baris berikut dalam output menampilkan lebih banyak detail.

```
+---------------+-------------+
| Variable_name | Value

                                                                    |

+---------------+-------------+
| ...           | ...
| Ssl_cipher    | AES256-SHA

                                                                    |

| ...           | ...
| Ssl_version   | TLSv1.1

                                                                    |

| ...           | ...
+-----------------------------+
```

Jika Anda ingin terhubung ke instans DB melalui proksi, lihat [Terhubung ke sebuah proksi menggunakan autentikasi IAM].

Menghubungkan ke instans DB Anda menggunakan autentikasi IAM dari baris perintah: AWS CLI dan klien psql

Anda dapat terhubung dari baris perintah ke klaster DB Aurora PostgreSQL dengan AWS CLI dan alat baris perintah psql seperti yang dijelaskan berikut.

Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke instans DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM

> **Note**
>
> Untuk informasi tentang cara menghubungkan ke basis data menggunakan pgAdmin dengan autentikasi IAM, lihat postingan blog Menggunakan autentikasi IAM untuk terhubung dengan pgAdmin Amazon Aurora PostgreSQL atau Amazon RDS for PostgreSQL.

Topik

- Membuat token autentikasi IAM
- Menghubungkan ke klaster Aurora PostgreSQL

Membuat token autentikasi IAM

Token autentikasi terdiri dari ratusan karakter sehingga kemungkinan menjadi sulit ditangani di baris perintah. Salah satu cara untuk mengatasi ini adalah dengan menyimpan token ke variabel lingkungan, lalu menggunakan variabel tersebut saat Anda terhubung. Contoh berikut menunjukkan cara menggunakan AWS CLI untuk mendapatkan token autentikasi yang ditandatangani menggunakan perintah `generate-db-auth-token` dan menyimpannya di variabel lingkungan `PGPASSWORD`.

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"
```

Dalam contoh, parameter untuk perintah `generate-db-auth-token` adalah sebagai berikut:

- `--hostname` – Nama host instans DB yang ingin Anda akses
- `--port` – Nomor port yang digunakan untuk menghubungkan ke instans DB Anda
- `--region` – Wilayah AWS tempat instans DB beroperasi
- `--username` – Akun basis data yang ingin Anda akses

Beberapa karakter pertama dari token yang dihasilkan terlihat sebagai berikut.

```
mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com:5432/?
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir instans DB untuk menghasilkan token autentikasi.

Menghubungkan ke klaster Aurora PostgreSQL

Format umum untuk menggunakan psql untuk terhubung ditampilkan sebagai berikut.

```
psql "host=hostName port=portNumber sslmode=verify-full
 sslrootcert=full_path_to_ssl_certificate dbname=DBName user=userName
 password=authToken"
```

Parameternya adalah sebagai berikut:

- `host` – Nama host instans DB yang ingin Anda akses
- `port` – Nomor port yang digunakan untuk menghubungkan ke instans DB Anda
- `sslmode` – Mode SSL yang akan digunakan

  Saat Anda menggunakan `sslmode=verify-full`, koneksi SSL memverifikasi titik akhir instans DB di sertifikat SSL.
- `sslrootcert` – Jalur lengkap ke file sertifikat SSL yang berisi kunci publik

  Untuk informasi selengkapnya, lihat Mengamankan data Aurora PostgreSQL dengan SSL/TLS.

  Untuk mengunduh sertifikat SSL, lihat .
- `dbname` – Basis data yang ingin Anda akses
- `user` – Akun basis data yang ingin Anda akses
- `password` – Token autentikasi IAM yang ditandatangani

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir instans DB untuk menghasilkan token autentikasi.

Contoh berikut menunjukkan cara menggunakan psql untuk terhubung. Dalam contoh, psql menggunakan variabel lingkungan `RDSHOST` untuk host dan variabel lingkungan `PGPASSWORD` untuk token yang dihasilkan. Selain itu, *`/sample_dir/`* adalah jalur penuh ke file sertifikat SSL yang berisi kunci publik.

```
export RDSHOST="mypostgres-cluster.cluster-123456789012.us-west-2.rds.amazonaws.com"
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname $RDSHOST --port 5432 --
region us-west-2 --username jane_doe )"

psql "host=$RDSHOST port=5432 sslmode=verify-full sslrootcert=/sample_dir/global-
bundle.pem dbname=DBName user=jane_doe password=$PGPASSWORD"
```

Jika Anda ingin terhubung ke klaster DB melalui proksi, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

Menghubungkan ke klaster DB menggunakan autentikasi IAM dan AWS SDK for .NET

Anda dapat menghubungkan ke klaster DB Aurora MySQL atau Aurora PostgreSQL dengan AWS SDK for .NET seperti yang dijelaskan berikut ini.

Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke instans DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM

Contoh

Contoh kode berikut ini menunjukkan cara membuat token autentikasi, lalu menggunakannya untuk menghubungkan ke instans DB.

Untuk menjalankan contoh kode ini, Anda memerlukan AWS SDK for .NET, yang ada di situs AWS. Paket `AWSSDK.CORE` dan `AWSSDK.RDS` diperlukan. Untuk terhubung ke instans DB, gunakan konektor basis data .NET untuk mesin DB, seperti MySqlConnector for MariaDB atau MySQL, atau Npgsql for PostgreSQL.

Kode ini terhubung ke klaster DB Aurora MySQL. Ubah nilai variabel berikut sesuai kebutuhan:

- `server` – Titik akhir instans DB yang ingin Anda akses
- `user` – Akun basis data yang ingin Anda akses
- `database` – Basis data yang ingin Anda akses
- `port` – Nomor port yang digunakan untuk menghubungkan ke instans DB Anda
- `SslMode` – Mode SSL yang akan digunakan

  Saat Anda menggunakan `SslMode=Required`, koneksi SSL memverifikasi titik akhir instans DB di sertifikat SSL.
- `SslCa` – Jalur lengkap ke sertifikat SSL untuk Amazon Aurora

  Untuk mengunduh sertifikat SSL, lihat .

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir instans DB untuk menghasilkan token autentikasi.

```
using System;
using System.Data;
using MySql.Data;
using MySql.Data.MySqlClient;
using Amazon;

namespace ubuntu
{
  class Program
  {
    static void Main(string[] args)
    {
```

```
      var pwd =
 Amazon.RDS.Util.RDSAuthTokenGenerator.GenerateAuthToken(RegionEndpoint.USEast1,
 "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com", 3306, "jane_doe");
      // for debug only Console.Write("{0}\n", pwd);  //this verifies the token is
 generated

      MySqlConnection conn = new
 MySqlConnection($"server=mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;user=jane_doe;database=mydB;port=3306;password={pwd};SslMode=Required;
      conn.Open();

      // Define a query
      MySqlCommand sampleCommand = new MySqlCommand("SHOW DATABASES;", conn);

      // Execute a query
      MySqlDataReader mysqlDataRdr = sampleCommand.ExecuteReader();

      // Read all rows and output the first column in each row
      while (mysqlDataRdr.Read())
        Console.WriteLine(mysqlDataRdr[0]);

      mysqlDataRdr.Close();
      // Close connection
      conn.Close();
    }
  }
}
```

Kode ini terhubung ke klaster DB Aurora PostgreSQL.

Ubah nilai variabel berikut sesuai kebutuhan:

- `Server` – Titik akhir instans DB yang ingin Anda akses
- `User ID` – Akun basis data yang ingin Anda akses
- `Database` – Basis data yang ingin Anda akses
- `Port` – Nomor port yang digunakan untuk menghubungkan ke instans DB Anda
- `SSL Mode` – Mode SSL yang akan digunakan

  Saat Anda menggunakan `SSL Mode=Required`, koneksi SSL memverifikasi titik akhir instans DB di sertifikat SSL.
- `Root Certificate` – Jalur lengkap ke sertifikat SSL untuk Amazon Aurora

Untuk mengunduh sertifikat SSL, lihat .

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir instans DB untuk menghasilkan token autentikasi.

```
using System;
using Npgsql;
using Amazon.RDS.Util;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            var pwd =
 RDSAuthTokenGenerator.GenerateAuthToken("postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com", 5432, "jane_doe");
// for debug only Console.Write("{0}\n", pwd);  //this verifies the token is generated

            NpgsqlConnection conn = new
 NpgsqlConnection($"Server=postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com;User Id=jane_doe;Password={pwd};Database=mydb;SSL
 Mode=Require;Root Certificate=full_path_to_ssl_certificate");
            conn.Open();

            // Define a query
                   NpgsqlCommand cmd = new NpgsqlCommand("select count(*) FROM
 pg_user", conn);

            // Execute a query
            NpgsqlDataReader dr = cmd.ExecuteReader();

            // Read all rows and output the first column in each row
            while (dr.Read())
                Console.Write("{0}\n", dr[0]);

            // Close connection
```

```
            conn.Close();
        }
    }
}
```

Jika Anda ingin terhubung ke klaster DB melalui proksi, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

Menghubungkan ke instans DB menggunakan autentikasi IAM dan AWS SDK for Go

Anda dapat menghubungkan ke klaster DB Aurora MySQL atau Aurora PostgreSQL dengan AWS SDK for Go seperti yang dijelaskan berikut ini.

Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke instans DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM

Contoh

Untuk menjalankan contoh kode ini, Anda memerlukan AWS SDK for Go, yang ada di situs AWS.

Ubah nilai variabel berikut sesuai kebutuhan:

- `dbName` – Basis data yang ingin Anda akses
- `dbUser` – Akun basis data yang ingin Anda akses
- `dbHost` – Titik akhir instans DB yang ingin Anda akses

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir instans DB untuk menghasilkan token autentikasi.

- `dbPort` – Nomor port yang digunakan untuk menghubungkan ke instans DB Anda
- `region` – Wilayah AWS tempat instans DB beroperasi

Selain itu, pastikan pustaka yang diimpor dalam kode sampel ada di sistem Anda.

> **Important**
>
> Contoh dalam bagian ini menggunakan kode berikut untuk menyediakan kredensial yang mengakses basis data dari lingkungan lokal:
> `creds := credentials.NewEnvCredentials()`
> Jika Anda mengakses basis data dari layanan AWS, seperti Amazon EC2 atau Amazon ECS, Anda dapat mengganti kode dengan kode berikut:
> `sess := session.Must(session.NewSession())`
> `creds := sess.Config.Credentials`
> Jika Anda membuat perubahan ini, pastikan Anda menambahkan impor berikut:
> `"github.com/aws/aws-sdk-go/aws/session"`

Topik

- Menghubungkan menggunakan autentikasi IAM dan AWS SDK for Go V2
- Menghubungkan menggunakan autentikasi IAM dan AWS SDK for Go V1.

Menghubungkan menggunakan autentikasi IAM dan AWS SDK for Go V2

Anda dapat menghubungkan ke instans DB menggunakan autentikasi IAM dan AWS SDK for Go V2.

Contoh kode berikut ini menunjukkan cara membuat token autentikasi, lalu menggunakannya untuk menghubungkan ke instans DB.

Kode ini terhubung ke klaster DB Aurora MySQL.

```
package main

import (
     "context"
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/go-sql-driver/mysql"
)
```

```
func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "mysqlcluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
     var dbPort int = 3306
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authenticationToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

Kode ini terhubung ke klaster DB Aurora PostgreSQL.

```
package main

import (
     "context"
```

```
     "database/sql"
     "fmt"

     "github.com/aws/aws-sdk-go-v2/config"
     "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
     _ "github.com/lib/pq"
)

func main() {

     var dbName string = "DatabaseName"
     var dbUser string = "DatabaseUser"
     var dbHost string = "postgresmycluster.cluster-123456789012.us-
east-1.rds.amazonaws.com"
     var dbPort int = 5432
     var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
     var region string = "us-east-1"

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
     panic("configuration error: " + err.Error())
    }

    authenticationToken, err := auth.BuildAuthToken(
     context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
    if err != nil {
     panic("failed to create authentication token: " + err.Error())
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authenticationToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

Jika Anda ingin terhubung ke instans DB melalui proksi, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

Menghubungkan menggunakan autentikasi IAM dan AWS SDK for Go V1.

Anda dapat menghubungkan ke instans DB menggunakan autentikasi IAM dan AWS SDK for Go V1

Contoh kode berikut ini menunjukkan cara membuat token autentikasi, lalu menggunakannya untuk menghubungkan ke instans DB.

Kode ini terhubung ke klaster DB Aurora MySQL.

```
package main

import (
    "database/sql"
    "fmt"
    "log"

    "github.com/aws/aws-sdk-go/aws/credentials"
    "github.com/aws/aws-sdk-go/service/rds/rdsutils"
    _ "github.com/go-sql-driver/mysql"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 3306
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
        dbUser, authToken, dbEndpoint, dbName,
    )

    db, err := sql.Open("mysql", dsn)
    if err != nil {
```

```
        panic(err)
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

Kode ini terhubung ke klaster DB Aurora PostgreSQL.

```
package main

import (
 "database/sql"
 "fmt"

 "github.com/aws/aws-sdk-go/aws/credentials"
 "github.com/aws/aws-sdk-go/service/rds/rdsutils"
 _ "github.com/lib/pq"
)

func main() {
    dbName := "app"
    dbUser := "jane_doe"
    dbHost := "postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
    dbPort := 5432
    dbEndpoint := fmt.Sprintf("%s:%d", dbHost, dbPort)
    region := "us-east-1"

    creds := credentials.NewEnvCredentials()
    authToken, err := rdsutils.BuildAuthToken(dbEndpoint, region, dbUser, creds)
    if err != nil {
        panic(err)
    }

    dsn := fmt.Sprintf("host=%s port=%d user=%s password=%s dbname=%s",
        dbHost, dbPort, dbUser, authToken, dbName,
    )

    db, err := sql.Open("postgres", dsn)
    if err != nil {
        panic(err)
```

```
    }

    err = db.Ping()
    if err != nil {
        panic(err)
    }
}
```

Jika Anda ingin terhubung ke instans DB melalui proksi, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

Menghubungkan ke klaster DB menggunakan autentikasi IAM dan AWS SDK for Java

Anda dapat menghubungkan ke klaster DB Aurora MySQL atau Aurora PostgreSQL dengan AWS SDK for Java seperti yang dijelaskan berikut ini.

Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke instans DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM
- Siapkan SDK AWS untuk Java

Topik

- Membuat token autentikasi IAM
- Membuat token autentikasi IAM secara manual
- Menghubungkan ke instans DB

Membuat token autentikasi IAM

Jika Anda menulis program menggunakan AWS SDK for Java, Anda dapat memperoleh token autentikasi yang ditandatangani dengan menggunakan kelas `RdsIamAuthTokenGenerator`. Penggunaan kelas ini mengharuskan Anda untuk memberikan kredensial AWS. Untuk melakukannya, Anda membuat instans kelas `DefaultAWSCredentialsProviderChain`. `DefaultAWSCredentialsProviderChain` menggunakan kunci akses AWS pertama dan kunci

rahasia yang ditemukan di rantai penyedia kredensial default. Untuk informasi selengkapnya tentang kunci akses AWS, lihat Mengelola kunci akses untuk pengguna.

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir instans DB untuk menghasilkan token autentikasi.

Setelah membuat instans `RdsIamAuthTokenGenerator`, Anda dapat memanggil metode `getAuthToken` untuk mendapatkan token yang ditandatangani. Berikan Wilayah AWS, nama host, nomor port, dan nama pengguna. Contoh kode berikut menunjukkan cara melakukannya.

```
package com.amazonaws.codesamples;

import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;

public class GenerateRDSAuthToken {

    public static void main(String[] args) {

     String region = "us-west-2";
     String hostname = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
     String port = "3306";
     String username = "jane_doe";

     System.out.println(generateAuthToken(region, hostname, port, username));
    }

    static String generateAuthToken(String region, String hostName, String port, String
 username) {

     RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
      .credentials(new DefaultAWSCredentialsProviderChain())
      .region(region)
      .build();

     String authToken = generator.getAuthToken(
      GetIamAuthTokenRequest.builder()
      .hostname(hostName)
```

```
      .port(Integer.parseInt(port))
      .userName(username)
      .build());

     return authToken;
    }

}
```

Membuat token autentikasi IAM secara manual

Di Java, cara termudah untuk menghasilkan token autentikasi adalah dengan menggunakan `RdsIamAuthTokenGenerator`. Kelas ini membuat token autentikasi untuk Anda, dan kemudian menandatanganinya menggunakan AWS signature versi 4. Untuk informasi selengkapnya, lihat Proses penandatanganan Signature versi 4 di Referensi Umum AWS.

Namun, Anda juga dapat membuat dan menandatangani token autentikasi secara manual, seperti ditunjukkan dalam contoh kode berikut.

```
package com.amazonaws.codesamples;

import com.amazonaws.SdkClientException;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.SigningAlgorithm;
import com.amazonaws.util.BinaryUtils;
import org.apache.commons.lang3.StringUtils;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.nio.charset.Charset;
import java.security.MessageDigest;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.SortedMap;
import java.util.TreeMap;

import static com.amazonaws.auth.internal.SignerConstants.AWS4_TERMINATOR;
import static com.amazonaws.util.StringUtils.UTF8;

public class CreateRDSAuthTokenManually {
    public static String httpMethod = "GET";
    public static String action = "connect";
    public static String canonicalURIParameter = "/";
```

```
    public static SortedMap<String, String> canonicalQueryParameters = new TreeMap();
    public static String payload = StringUtils.EMPTY;
    public static String signedHeader = "host";
    public static String algorithm = "AWS4-HMAC-SHA256";
    public static String serviceName = "rds-db";
    public static String requestWithoutSignature;

    public static void main(String[] args) throws Exception {

        String region = "us-west-2";
        String instanceName = "rdsmysql.123456789012.us-west-2.rds.amazonaws.com";
        String port = "3306";
        String username = "jane_doe";

        Date now = new Date();
        String date = new SimpleDateFormat("yyyyMMdd").format(now);
        String dateTimeStamp = new
SimpleDateFormat("yyyyMMdd'T'HHmmss'Z'").format(now);
        DefaultAWSCredentialsProviderChain creds = new
DefaultAWSCredentialsProviderChain();
    String awsAccessKey = creds.getCredentials().getAWSAccessKeyId();
    String awsSecretKey = creds.getCredentials().getAWSSecretKey();
        String expiryMinutes = "900";

        System.out.println("Step 1:  Create a canonical request:");
        String canonicalString = createCanonicalString(username, awsAccessKey, date,
dateTimeStamp, region, expiryMinutes, instanceName, port);
        System.out.println(canonicalString);
        System.out.println();

        System.out.println("Step 2:  Create a string to sign:");
        String stringToSign = createStringToSign(dateTimeStamp, canonicalString,
awsAccessKey, date, region);
        System.out.println(stringToSign);
        System.out.println();

        System.out.println("Step 3:  Calculate the signature:");
        String signature = BinaryUtils.toHex(calculateSignature(stringToSign,
newSigningKey(awsSecretKey, date, region, serviceName)));
        System.out.println(signature);
        System.out.println();

        System.out.println("Step 4:  Add the signing info to the request");
```

```
        System.out.println(appendSignature(signature));
        System.out.println();

    }

    //Step 1: Create a canonical request date should be in format YYYYMMDD and dateTime
 should be in format YYYYMMDDTHHMMSSZ
    public static String createCanonicalString(String user, String accessKey, String
 date, String dateTime, String region, String expiryPeriod, String hostName, String
 port) throws Exception {
        canonicalQueryParameters.put("Action", action);
        canonicalQueryParameters.put("DBUser", user);
        canonicalQueryParameters.put("X-Amz-Algorithm", "AWS4-HMAC-SHA256");
        canonicalQueryParameters.put("X-Amz-Credential", accessKey + "%2F" + date +
"%2F" + region + "%2F" + serviceName + "%2Faws4_request");
        canonicalQueryParameters.put("X-Amz-Date", dateTime);
        canonicalQueryParameters.put("X-Amz-Expires", expiryPeriod);
        canonicalQueryParameters.put("X-Amz-SignedHeaders", signedHeader);
        String canonicalQueryString = "";
        while(!canonicalQueryParameters.isEmpty()) {
            String currentQueryParameter = canonicalQueryParameters.firstKey();
            String currentQueryParameterValue =
 canonicalQueryParameters.remove(currentQueryParameter);
            canonicalQueryString = canonicalQueryString + currentQueryParameter + "=" +
 currentQueryParameterValue;
            if (!currentQueryParameter.equals("X-Amz-SignedHeaders")) {
                canonicalQueryString += "&";
            }
        }
        String canonicalHeaders = "host:" + hostName + ":" + port + '\n';
        requestWithoutSignature = hostName + ":" + port + "/?" + canonicalQueryString;

        String hashedPayload = BinaryUtils.toHex(hash(payload));
        return httpMethod + '\n' + canonicalURIParameter + '\n' + canonicalQueryString
 + '\n' + canonicalHeaders + '\n' + signedHeader + '\n' + hashedPayload;

    }

    //Step 2: Create a string to sign using sig v4
    public static String createStringToSign(String dateTime, String canonicalRequest,
 String accessKey, String date, String region) throws Exception {
        String credentialScope = date + "/" + region + "/" + serviceName + "/
aws4_request";
```

```
        return algorithm + '\n' + dateTime + '\n' + credentialScope + '\n' +
BinaryUtils.toHex(hash(canonicalRequest));

    }

    //Step 3: Calculate signature
    /**
     * Step 3 of the &AWS; Signature version 4 calculation. It involves deriving
     * the signing key and computing the signature. Refer to
     * http://docs.aws.amazon
     * .com/general/latest/gr/sigv4-calculate-signature.html
     */
    public static byte[] calculateSignature(String stringToSign,
                                            byte[] signingKey) {
        return sign(stringToSign.getBytes(Charset.forName("UTF-8")), signingKey,
                SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(byte[] data, byte[] key,
                          SigningAlgorithm algorithm) throws SdkClientException {
        try {
            Mac mac = algorithm.getMac();
            mac.init(new SecretKeySpec(key, algorithm.toString()));
            return mac.doFinal(data);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    public static byte[] newSigningKey(String secretKey,
                                   String dateStamp, String regionName, String
serviceName) {
        byte[] kSecret = ("AWS4" + secretKey).getBytes(Charset.forName("UTF-8"));
        byte[] kDate = sign(dateStamp, kSecret, SigningAlgorithm.HmacSHA256);
        byte[] kRegion = sign(regionName, kDate, SigningAlgorithm.HmacSHA256);
        byte[] kService = sign(serviceName, kRegion,
                SigningAlgorithm.HmacSHA256);
        return sign(AWS4_TERMINATOR, kService, SigningAlgorithm.HmacSHA256);
    }

    public static byte[] sign(String stringData, byte[] key,
                      SigningAlgorithm algorithm) throws SdkClientException {
```

```
        try {
            byte[] data = stringData.getBytes(UTF8);
            return sign(data, key, algorithm);
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to calculate a request signature: "
                            + e.getMessage(), e);
        }
    }

    //Step 4: append the signature
    public static String appendSignature(String signature) {
        return requestWithoutSignature + "&X-Amz-Signature=" + signature;
    }

    public static byte[] hash(String s) throws Exception {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(s.getBytes(UTF8));
            return md.digest();
        } catch (Exception e) {
            throw new SdkClientException(
                    "Unable to compute hash while signing request: "
                            + e.getMessage(), e);
        }
    }
}
```

Menghubungkan ke instans DB

Contoh kode berikut menunjukkan cara membuat token autentikasi, lalu menggunakannya untuk menghubungkan ke klaster yang menjalankan Aurora MySQL.

Untuk menjalankan contoh kode ini, Anda memerlukan AWS SDK for Java, yang ada di situs AWS. Selain itu, Anda memerlukan hal berikut:

- MySQL Connector/J. Contoh kode ini diuji dengan `mysql-connector-java-5.1.33-bin.jar`.
- Sertifikat menengah untuk Amazon Aurora yang khusus untuk Wilayah AWS. (Untuk informasi selengkapnya, lihat .) Saat runtime, pemuat kelas mencari sertifikat di direktori yang sama seperti contoh kode Java ini, sehingga pemuat kelas dapat menemukannya.
- Ubah nilai variabel berikut sesuai kebutuhan:
  - `RDS_INSTANCE_HOSTNAME` – Nama host instans DB yang ingin Anda akses.

- `RDS_INSTANCE_PORT` – Nomor port yang digunakan untuk menghubungkan ke instans DB PostgreSQL Anda.
- `REGION_NAME` – Wilayah AWS tempat instans DB beroperasi.
- `DB_USER` – Akun basis data yang ingin Anda akses.
- `SSL_CERTIFICATE` – Sertifikat menengah untuk Amazon Aurora yang khusus untuk Wilayah AWS.

  Untuk mengunduh sertifikat untuk Wilayah AWS Anda, lihat . Tempatkan sertifikat SSL di direktori yang sama dengan file program Java ini, sehingga pemuat kelas dapat menemukan sertifikat saat runtime.

Contoh kode ini memperoleh kredensial AWS dari rantai penyedia kredensial default.

> **Note**
>
> Tentukan kata sandi untuk `DEFAULT_KEY_STORE_PASSWORD` selain prompt yang ditampilkan di sini sebagai praktik terbaik keamanan.

```
package com.amazonaws.samples;

import com.amazonaws.services.rds.auth.RdsIamAuthTokenGenerator;
import com.amazonaws.services.rds.auth.GetIamAuthTokenRequest;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.security.KeyStore;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;
```

```
import java.net.URL;

public class IAMDatabaseAuthenticationTester {
    //&AWS; Credentials of the IAM user with policy enabling IAM Database Authenticated
 access to the db by the db user.
    private static final DefaultAWSCredentialsProviderChain creds = new
 DefaultAWSCredentialsProviderChain();
    private static final String AWS_ACCESS_KEY =
 creds.getCredentials().getAWSAccessKeyId();
    private static final String AWS_SECRET_KEY =
 creds.getCredentials().getAWSSecretKey();

    //Configuration parameters for the generation of the IAM Database Authentication
 token
    private static final String RDS_INSTANCE_HOSTNAME = "rdsmysql.123456789012.us-
west-2.rds.amazonaws.com";
    private static final int RDS_INSTANCE_PORT = 3306;
    private static final String REGION_NAME = "us-west-2";
    private static final String DB_USER = "jane_doe";
    private static final String JDBC_URL = "jdbc:mysql://" + RDS_INSTANCE_HOSTNAME +
 ":" + RDS_INSTANCE_PORT;

    private static final String SSL_CERTIFICATE = "rds-ca-2019-us-west-2.pem";

    private static final String KEY_STORE_TYPE = "JKS";
    private static final String KEY_STORE_PROVIDER = "SUN";
    private static final String KEY_STORE_FILE_PREFIX = "sys-connect-via-ssl-test-
cacerts";
    private static final String KEY_STORE_FILE_SUFFIX = ".jks";
    private static final String DEFAULT_KEY_STORE_PASSWORD = "changeit";

    public static void main(String[] args) throws Exception {
        //get the connection
        Connection connection = getDBConnectionUsingIam();

        //verify the connection is successful
        Statement stmt= connection.createStatement();
        ResultSet rs=stmt.executeQuery("SELECT 'Success!' FROM DUAL;");
        while (rs.next()) {
             String id = rs.getString(1);
            System.out.println(id); //Should print "Success!"
        }
```

```
        //close the connection
        stmt.close();
        connection.close();

        clearSslProperties();

    }

    /**
     * This method returns a connection to the db instance authenticated using IAM
 Database Authentication
     * @return
     * @throws Exception
     */
    private static Connection getDBConnectionUsingIam() throws Exception {
        setSslProperties();
        return DriverManager.getConnection(JDBC_URL, setMySqlConnectionProperties());
    }

    /**
     * This method sets the mysql connection properties which includes the IAM Database
 Authentication token
     * as the password. It also specifies that SSL verification is required.
     * @return
     */
    private static Properties setMySqlConnectionProperties() {
        Properties mysqlConnectionProperties = new Properties();
        mysqlConnectionProperties.setProperty("verifyServerCertificate","true");
        mysqlConnectionProperties.setProperty("useSSL", "true");
        mysqlConnectionProperties.setProperty("user",DB_USER);
        mysqlConnectionProperties.setProperty("password",generateAuthToken());
        return mysqlConnectionProperties;
    }

    /**
     * This method generates the IAM Auth Token.
     * An example IAM Auth Token would look like follows:
     * btusi123.cmz7kenwo2ye.rds.cn-north-1.amazonaws.com.cn:3306/?
Action=connect&DBUser=iamtestuser&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-
Date=20171003T010726Z&X-Amz-SignedHeaders=host&X-Amz-Expires=899&X-Amz-
Credential=AKIAPFXHGVDI5RNFO4AQ%2F20171003%2Fcn-north-1%2Frds-db%2Faws4_request&X-Amz-
Signature=f9f45ef96c1f770cdad11a53e33ffa4c3730bc03fdee820cfdf1322eed15483b
     * @return
     */
```

```
   private static String generateAuthToken() {
       BasicAWSCredentials awsCredentials = new BasicAWSCredentials(AWS_ACCESS_KEY,
AWS_SECRET_KEY);

       RdsIamAuthTokenGenerator generator = RdsIamAuthTokenGenerator.builder()
               .credentials(new
AWSStaticCredentialsProvider(awsCredentials)).region(REGION_NAME).build();
       return generator.getAuthToken(GetIamAuthTokenRequest.builder()

.hostname(RDS_INSTANCE_HOSTNAME).port(RDS_INSTANCE_PORT).userName(DB_USER).build());
   }

   /**
    * This method sets the SSL properties which specify the key store file, its type
and password:
    * @throws Exception
    */
   private static void setSslProperties() throws Exception {
       System.setProperty("javax.net.ssl.trustStore", createKeyStoreFile());
       System.setProperty("javax.net.ssl.trustStoreType", KEY_STORE_TYPE);
       System.setProperty("javax.net.ssl.trustStorePassword",
DEFAULT_KEY_STORE_PASSWORD);
   }

   /**
    * This method returns the path of the Key Store File needed for the SSL
verification during the IAM Database Authentication to
    * the db instance.
    * @return
    * @throws Exception
    */
   private static String createKeyStoreFile() throws Exception {
       return createKeyStoreFile(createCertificate()).getPath();
   }

   /**
    *  This method generates the SSL certificate
    * @return
    * @throws Exception
    */
   private static X509Certificate createCertificate() throws Exception {
       CertificateFactory certFactory = CertificateFactory.getInstance("X.509");
       URL url = new File(SSL_CERTIFICATE).toURI().toURL();
       if (url == null) {
```

```
            throw new Exception();
        }
        try (InputStream certInputStream = url.openStream()) {
            return (X509Certificate) certFactory.generateCertificate(certInputStream);
        }
    }

    /**
     * This method creates the Key Store File
     * @param rootX509Certificate - the SSL certificate to be stored in the KeyStore
     * @return
     * @throws Exception
     */
    private static File createKeyStoreFile(X509Certificate rootX509Certificate) throws
 Exception {
        File keyStoreFile = File.createTempFile(KEY_STORE_FILE_PREFIX,
 KEY_STORE_FILE_SUFFIX);
        try (FileOutputStream fos = new FileOutputStream(keyStoreFile.getPath())) {
            KeyStore ks = KeyStore.getInstance(KEY_STORE_TYPE, KEY_STORE_PROVIDER);
            ks.load(null);
            ks.setCertificateEntry("rootCaCertificate", rootX509Certificate);
            ks.store(fos, DEFAULT_KEY_STORE_PASSWORD.toCharArray());
        }
        return keyStoreFile;
    }

    /**
     * This method clears the SSL properties.
     * @throws Exception
     */
    private static void clearSslProperties() throws Exception {
           System.clearProperty("javax.net.ssl.trustStore");
           System.clearProperty("javax.net.ssl.trustStoreType");
           System.clearProperty("javax.net.ssl.trustStorePassword");
    }

}
```

Jika Anda ingin terhubung ke klaster DB melalui proksi, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

Menghubungkan ke klaster DB menggunakan autentikasi IAM dan AWS SDK for Python (Boto3)

Anda dapat menghubungkan ke klaster DB Aurora MySQL atau Aurora PostgreSQL dengan AWS SDK for Python (Boto3) seperti yang dijelaskan berikut ini.

Prasyarat

Berikut adalah prasyarat untuk menghubungkan ke instans DB menggunakan autentikasi IAM:

- Mengaktifkan dan menonaktifkan autentikasi basis data IAM
- Membuat dan menggunakan kebijakan IAM untuk akses basis data IAM
- Membuat akun basis data menggunakan autentikasi IAM

Selain itu, pastikan pustaka yang diimpor dalam kode sampel ada di sistem Anda.

Contoh

Contoh kode menggunakan profil untuk kredensial bersama. Untuk informasi tentang cara menentukan kredensial, lihat Kredensial dalam dokumentasi AWS SDK for Python (Boto3).

Contoh kode berikut ini menunjukkan cara membuat token autentikasi, lalu menggunakannya untuk menghubungkan ke instans DB.

Untuk menjalankan contoh kode ini, Anda memerlukan AWS SDK for Python (Boto3), yang ada di situs AWS.

Ubah nilai variabel berikut sesuai kebutuhan:

- `ENDPOINT` – Titik akhir instans DB yang ingin Anda akses
- `PORT` – Nomor port yang digunakan untuk menghubungkan ke instans DB Anda
- `USER` – Akun basis data yang ingin Anda akses
- `REGION` – Wilayah AWS tempat instans DB beroperasi
- `DBNAME` – Basis data yang ingin Anda akses
- `SSLCERTIFICATE` – Jalur lengkap ke sertifikat SSL untuk Amazon Aurora

  Untuk `ssl_ca`, tentukan sertifikat SSL. Untuk mengunduh sertifikat SSL, lihat .

> **Note**
>
> Anda tidak dapat menggunakan data DNS Route 53 kustom atau titik akhir kustom Aurora sebagai pengganti titik akhir instans DB untuk menghasilkan token autentikasi.

Kode ini terhubung ke klaster DB Aurora MySQL.

Sebelum menjalankan kode ini, instal driver PyMySQL dengan mengikuti petunjuk dalam Indeks Paket Python.

```
import pymysql
import sys
import boto3
import os

ENDPOINT="mysqlcluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="3306"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"
os.environ['LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN'] = '1'

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='default')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn =  pymysql.connect(host=ENDPOINT, user=USER, passwd=token, port=PORT,
 database=DBNAME, ssl_ca='SSLCERTIFICATE')
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

Kode ini terhubung ke klaster DB Aurora PostgreSQL.

Sebelum menjalankan kode ini, instal `psycopg2` dengan mengikuti petunjuk dalam dokumentasi Psycopg.

```
import psycopg2
import sys
import boto3
import os

ENDPOINT="postgresmycluster.cluster-123456789012.us-east-1.rds.amazonaws.com"
PORT="5432"
USER="jane_doe"
REGION="us-east-1"
DBNAME="mydb"

#gets the credentials from .aws/credentials
session = boto3.Session(profile_name='RDSCreds')
client = session.client('rds')

token = client.generate_db_auth_token(DBHostname=ENDPOINT, Port=PORT, DBUsername=USER,
 Region=REGION)

try:
    conn = psycopg2.connect(host=ENDPOINT, port=PORT, database=DBNAME, user=USER,
 password=token, sslrootcert="SSLCERTIFICATE")
    cur = conn.cursor()
    cur.execute("""SELECT now()""")
    query_results = cur.fetchall()
    print(query_results)
except Exception as e:
    print("Database connection failed due to {}".format(e))
```

Jika Anda ingin terhubung ke klaster DB melalui proksi, lihat Terhubung ke sebuah proksi menggunakan autentikasi IAM.

Memecahkan masalah identitas dan akses Amazon Aurora

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda hadapi saat bekerja dengan Aurora dan IAM.

Topik

- Saya tidak diberi otorisasi untuk melakukan tindakan di Aurora
- Saya tidak memiliki izin untuk melakukan iam:PassRole
- Saya ingin mengizinkan orang di luar akun AWS saya untuk mengakses sumber daya Aurora

Saya tidak diberi otorisasi untuk melakukan tindakan di Aurora

Jika AWS Management Console memberi tahu bahwa Anda tidak diberi otorisasi untuk melakukan tindakan, Anda harus menghubungi administrator untuk mendapatkan bantuan. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Contoh kesalahan berikut terjadi saat pengguna `mateojackson` mencoba menggunakan konsol untuk melihat detail tentang *`widget`*, tetapi tidak memiliki izin `rds:`*`GetWidget`*.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 rds:GetWidget on resource: my-example-widget
```

Dalam hal ini, Mateo meminta administratornya untuk memperbarui kebijakannya untuk mengizinkan dia mengakses sumber daya *`my-example-widget`* menggunakan tindakan `rds:`*`GetWidget`*.

Saya tidak memiliki izin untuk melakukan iam:PassRole

Jika Anda menerima kesalahan bahwa Anda tidak diberi otorisasi untuk melakukan tindakan `iam:PassRole`, Anda harus menghubungi administrator untuk mendapatkan bantuan. Administrator Anda adalah orang yang memberi Anda kredensial masuk. Minta orang tersebut untuk memperbarui kebijakan Anda agar Anda dapat meneruskan peran ke Aurora.

Beberapa layanan AWS mengizinkan Anda untuk melewati peran yang sudah ada ke layanan tersebut, alih-alih membuat peran layanan atau peran yang ditautkan ke layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh kesalahan berikut terjadi saat pengguna bernama `marymajor` mencoba menggunakan konsol untuk melakukan tindakan di Aurora. Namun, tindakan ini mengharuskan layanan memiliki izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut ke layanan.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dalam kasus ini, Mary meminta administrator untuk memperbarui kebijakannya agar dia dapat melakukan tindakan `iam:PassRole`.

Saya ingin mengizinkan orang di luar akun AWS saya untuk mengakses sumber daya Aurora

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau pengguna di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis sumber daya atau daftar kontrol akses (ACL), Anda dapat menggunakan kebijakan tersebut untuk memberi pengguna akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa hal berikut:

- Untuk mempelajari apakah Aurora mendukung fitur ini, lihat Cara kerja Amazon Aurora dengan IAM.
- Untuk mempelajari cara memberikan akses ke sumber daya Anda di seluruh akun AWS yang Anda miliki, lihat Memberikan akses kepada pengguna IAM di akun AWS lain yang Anda miliki dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses ke sumber daya Anda ke akun AWS pihak ke tiga, lihat Memberikan akses ke akun AWS milik pihak ketiga dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat Memberikan akses kepada pengguna eksternal yang sah (federasi identitas) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara penggunaan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Perbedaan antara peran IAM dan kebijakan berbasis sumber daya di Panduan Pengguna IAM.

Pencatatan dan pemantauan di Amazon Aurora

Pemantauan adalah bagian penting dari upaya memelihara keandalan, ketersediaan, dan performa Amazon Aurora dan solusi AWS Anda. Anda harus mengumpulkan data pemantauan dari semua bagian solusi AWS agar dapat dengan lebih mudah melakukan debug kegagalan multi-titik jika terjadi. AWS menyediakan beberapa alat untuk memantau sumber daya Amazon Aurora Anda dan merespons potensi insiden:

CloudWatch Alarm Amazon

Menggunakan CloudWatch alarm Amazon, Anda menonton satu metrik selama periode waktu yang Anda tentukan. Jika metrik melebihi ambang batas tertentu, pemberitahuan akan dikirim ke topik atau AWS Auto Scaling kebijakan Amazon SNS. CloudWatch alarm tidak memanggil tindakan karena mereka berada dalam keadaan tertentu. Sebaliknya, status harus diubah dan dipertahankan selama jangka waktu tertentu.

Log AWS CloudTrail

CloudTrail menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau AWS layanan di Aurora. CloudTrail menangkap semua panggilan API untuk Aurora sebagai peristiwa, termasuk panggilan dari konsol dan dari panggilan kode ke operasi Amazon RDS API. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat untuk Amazon Aurora, alamat IP dari mana permintaan dibuat, siapa yang membuat permintaan, kapan dibuat, dan detail tambahan. Untuk informasi selengkapnya, lihat Memantau panggilan API Amazon Aurora di AWS CloudTrail.

Pemantauan yang Ditingkatkan

Amazon Aurora menyediakan metrik secara waktu nyata untuk sistem operasi (OS) tempat klaster DB berjalan. Anda dapat melihat metrik untuk cluster DB menggunakan konsol, atau menggunakan output JSON Pemantauan yang Ditingkatkan dari Amazon CloudWatch Logs dalam sistem pemantauan pilihan Anda. Untuk informasi selengkapnya, lihat Memantau metrik OS dengan Pemantauan yang Disempurnakan.

Wawasan Performa Amazon RDS

Wawasan Performa memperluas fitur pemantauan Amazon Aurora yang ada untuk menggambarkan performa basis data Anda dan membantu Anda menganalisis masalah yang memengaruhinya. Dengan dasbor Wawasan Performa, Anda dapat memvisualisasikan beban basis data dan memfilter beban berdasarkan waktu tunggu, pernyataan SQL, host, atau pengguna. Untuk informasi selengkapnya, lihat Memantau muatan DB dengan Wawasan Performa di Amazon Aurora.

Log Basis Data

Anda dapat melihat, mengunduh, dan melihat log basis data menggunakan AWS Management Console, AWS CLI, atau RDS API. Untuk informasi selengkapnya, lihat Memantau file log Amazon Aurora.

Rekomendasi Amazon Aurora

Amazon Aurora memberikan rekomendasi otomatis untuk sumber daya basis data. Rekomendasi ini memberikan panduan praktik terbaik dengan menganalisis data konfigurasi, penggunaan, dan performa klaster DB. Untuk informasi selengkapnya, lihat Melihat dan menanggapi rekomendasi Amazon RDS.

Notifikasi Peristiwa Amazon Aurora

Amazon Aurora menggunakan Amazon Simple Notification Service (Amazon SNS) untuk memberikan notifikasi ketika peristiwa Amazon Aurora terjadi. Notifikasi ini bisa dalam bentuk apa pun yang didukung oleh Amazon SNS untuk Wilayah AWS, seperti email, pesan teks, atau panggilan ke titik akhir HTTP. Untuk informasi selengkapnya, lihat Bekerja dengan pemberitahuan peristiwa Amazon RDS.

AWS Trusted Advisor

Trusted Advisor mengacu pada praktik terbaik yang dipelajari dari melayani ratusan ribu pelanggan AWS. Trusted Advisor memeriksa lingkungan AWS Anda lalu membuat rekomendasi ketika ada peluang untuk menghemat uang, meningkatkan ketersediaan dan performa sistem, atau membantu menutup kesenjangan keamanan. Semua pelanggan AWS memiliki akses ke lima pemeriksaan Trusted Advisor. Pelanggan dengan paket dukungan Bisnis atau Perusahaan dapat melihat semua pemeriksaan Trusted Advisor.

Trusted Advisor memiliki pemeriksaan terkait Amazon Aurora berikut:

- Instans DB Diam Amazon Aurora
- Risiko Akses Grup Keamanan Amazon Aurora
- Pencadangan Amazon Aurora
- Multi-AZ Amazon Aurora
- Aksesibilitas Instans DB Aurora

Lihat informasi selengkapnya tentang beberapa pemeriksaan ini di Praktik terbaik (pemeriksaan) Trusted Advisor.

Aliran aktivitas basis data

Aliran aktivitas basis data dapat melindungi basis data Anda dari ancaman internal dengan mengontrol akses DBA ke aliran aktivitas basis data. Dengan demikian, pengumpulan, transmisi, penyimpanan, dan pemrosesan aliran aktivitas basis data selanjutnya berada di luar akses DBA yang mengelola basis data. Aliran aktivitas basis data dapat menyediakan sarana keamanan RDS

untuk basis data Anda dan memenuhi persyaratan kepatuhan dan peraturan. Untuk informasi selengkapnya, lihat Memantau Amazon Aurora dengan Aliran Aktivitas Basis Data.

Untuk informasi selengkapnya tentang cara memantau Aurora, lihat Memantau metrik di klaster Amazon Aurora.

Validasi kepatuhan untuk Amazon Aurora

Auditor pihak ketiga menilai keamanan dan kepatuhan Amazon Aurora sebagai bagian dari beberapa program kepatuhan AWS. Program ini mencakup SOC, PCI, FedRAMP, HIPAA, dan lainnya.

Untuk daftar layanan AWS dalam cakupan program kepatuhan khusus, lihat Layanan AWS yang dicakup oleh program kepatuhan. Untuk informasi umum, lihat Program kepatuhan AWS.

Anda dapat mengunduh laporan audit pihak ketiga menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat Mengunduh laporan di AWS Artifact.

Tanggung jawab kepatuhan Anda saat menggunakan Amazon Aurora ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, serta hukum dan peraturan yang berlaku. AWS menyediakan sumber daya berikut untuk memudahkan kepatuhan:

- Panduan mulai cepat keamanan dan kepatuhan – Panduan deployment ini membahas pertimbangan arsitektur dan berisi langkah-langkah untuk men-deploy lingkungan dasar yang berfokus pada keamanan dan kepatuhan di AWS.
- Membuat rancangan untuk Keamanan dan Kepatuhan HIPAA di Amazon Web Services – Laporan resmi ini menjelaskan cara perusahaan dapat menggunakan AWS untuk membuat aplikasi yang memenuhi HIPAA.
- Sumber daya kepatuhan AWS – Kumpulan petunjuk pengoperasian dan panduan ini mungkin berlaku untuk industri dan lokasi Anda.
- AWS Config – Layanan AWS ini menilai sejauh mana konfigurasi sumber daya Anda mematuhi praktik internal, pedoman industri, dan peraturan.
- AWS Security Hub – Layanan AWS berisi gambaran menyeluruh tentang status keamanan Anda dalam AWS. Security Hub menggunakan kontrol keamanan untuk mengevaluasi sumber daya AWS Anda dan memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik. Untuk daftar layanan dan kontrol yang didukung, lihat Referensi kontrol Security Hub.

Ketangguhan di Amazon Aurora

Infrastruktur global AWS dibangun di sekitar Wilayah dan Zona Ketersediaan AWS. AWS Wilayah menyediakan beberapa Zona Ketersediaan yang terpisah dan terisolasi secara fisik, yang terhubung dengan jaringan latensi rendah, throughput tinggi, dan sangat redundan. Dengan Zona Ketersediaan, Anda dapat merancang serta mengoperasikan aplikasi dan basis data yang secara otomatis melakukan fail over di antara Zona Ketersediaan tanpa gangguan. Zona Ketersediaan memiliki ketersediaan dan toleransi kesalahan yang lebih baik, dan dapat diskalakan dibandingkan infrastruktur pusat data tunggal atau multi tradisional.

Untuk informasi selengkapnya tentang Wilayah AWS dan Zona Ketersediaan, lihat Infrastruktur Global AWS.

Selain infrastruktur global AWS, Aurora menawarkan beberapa fitur untuk membantu mendukung kebutuhan ketangguhan dan pencadangan data Anda.

Pencadangan dan pemulihan

Aurora mencadangkan volume klaster Anda secara otomatis dan mempertahankan data yang dipulihkan selama periode retensi cadangan. Pencadangan Aurora bersifat kontinu dan inkremental, sehingga Anda dapat memulihkan dengan cepat ke titik mana pun dalam periode retensi cadangan. Tidak ada dampak performa atau gangguan layanan basis data saat data cadangan ditulis. Anda dapat menentukan periode retensi cadangan, dari 1 hingga 35 hari, saat Anda membuat atau memodifikasi klaster DB.

Jika Anda ingin mempertahankan cadangan di luar periode retensi cadangan, Anda juga dapat mengambil snapshot data itu di volume klaster Anda. Aurora mempertahankan data pemulihan inkremental selama masa retensi cadangan. Karena itu, Anda perlu membuat cuplikan hanya untuk data yang ingin Anda pertahankan setelah periode retensi cadangan. Anda dapat membuat klaster DB baru dari snapshot.

Anda dapat memulihkan data Anda dengan membuat klaster DB Aurora baru dari data cadangan yang dipertahankan Aurora, atau dari snapshot klaster DB yang telah Anda simpan. Anda dapat membuat dengan cepat salinan baru klaster DB dari data cadangan ke sebarang titik waktu selama masa retensi cadangan Anda. Sifat kontinu dan inkremental cadangan Aurora selama masa retensi cadangan berarti Anda tidak perlu sering-sering mengambil snapshot data untuk meningkatkan waktu pemulihan.

Untuk informasi selengkapnya, lihat Mencadangkan dan memulihkan klaster DB Amazon Aurora.

Replikasi

Aurora Replica adalah titik akhir independen dalam klaster DB Aurora, yang paling berguna untuk menskalakan operasi baca dan meningkatkan ketersediaan. Hingga 15 Aurora Replica dapat disebarkan ke seluruh Zona Ketersediaan yang dijangkau klaster DB di dalam sebuah Wilayah AWS. Volume klaster DB terdiri atas beberapa salinan data untuk klaster DB. Namun, data dalam volume klaster disajikan berupa volume tunggal logis ke instans DB utama dan ke Aurora Replica dalam klaster DB. Jika instans DB primer gagal, Aurora Replica dipromosikan menjadi instans DB primer.

Aurora juga mendukung opsi-opsi replikasi yang khusus untuk Aurora MySQL dan Aurora PostgreSQL.

Untuk informasi selengkapnya, lihat Replikasi dengan Amazon Aurora.

Failover

Aurora menyimpan salinan data dalam klaster DB di beberapa Zona Ketersediaan dalam satu Wilayah AWS. Penyimpanan ini terjadi terlepas dari apakah instans DB di klaster DB mencakup beberapa Zona Ketersediaan atau tidak. Saat Anda membuat Aurora Replica di beberapa Zona Ketersediaan, Aurora menyediakan dan memelihara secara otomatis semuanya secara sinkron. Instans DB utama direplikasi secara sinkron di beberapa Zona Ketersediaan ke Aurora Replica untuk menyediakan redundansi data, menghilangkan pembekuan I/O, dan meminimalkan lonjakan latensi selama pencadangan sistem. Menjalankan klaster DB dengan ketersediaan tinggi dapat meningkatkan ketersediaan selama pemeliharaan sistem terencana, dan membantu melindungi basis data Anda terhadap kegagalan dan gangguan Zona Ketersediaan.

Untuk informasi selengkapnya, lihat Ketersediaan yang tinggi untuk Amazon Aurora.

Keamanan infrastruktur dalam Amazon Aurora

Sebagai layanan terkelola, Amazon Relational Database Service dilindungi oleh keamanan jaringan global AWS. Untuk informasi tentang layanan keamanan AWS dan cara AWS melindungi infrastruktur, lihat Keamanan Cloud AWS. Guna mendesain lingkungan AWS Anda menggunakan praktik terbaik untuk keamanan infrastruktur, lihat Perlindungan Infrastruktur dalam Kerangka Kerja AWS Well-Architected Pilar Keamanan.

Anda dapat menggunakan panggilan API yang dipublikasikan AWS untuk mengakses Amazon RDS melalui jaringan. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Pengangkutan (TLS). Kami membutuhkan TLS 1.2 dan merekomendasikan TLS 1.3.
- Cipher cocok dengan perfect forward secrecy (PFS) seperti DHE (Ephemeral Diffie-Hellman) atau ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Sebagian besar sistem-sistem modern seperti Java 7 dan versi yang lebih baru mendukung mode-mode ini.

Selain itu, permintaan harus ditandatangani menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan pengguna utama IAM. Atau Anda dapat menggunakan AWS Security Token Service (AWS STS) untuk membuat kredensial keamanan sementara untuk menandatangani permintaan.

Selain itu, Aurora menawarkan fitur-fitur untuk membantu mendukung pemeliharaan infrastruktur keamanan.

Grup keamanan

Grup keamanan mengontrol akses lalu lintas masuk dan keluar dari klaster DB. Secara default, akses jaringan untuk klaster DB dinonaktifkan. Anda dapat menentukan aturan dalam grup keamanan yang mengizinkan akses dari rentang alamat IP, port, atau grup keamanan. Setelah aturan masuk dikonfigurasi, aturan yang sama berlaku untuk semua klaster DB yang terkait dengan grup keamanan tersebut.

Untuk informasi selengkapnya, lihat Mengontrol akses dengan grup keamanan.

Aksesibilitas publik

Saat Anda meluncurkan instans DB di dalam cloud privat virtual (VPC) berdasarkan layanan Amazon VPC, Anda dapat mengaktifkan atau menonaktifkan akses publik untuk instans tersebut. Untuk

menentukan apakah instans DB yang Anda buat memiliki nama DNS yang menyelesaikan ke alamat IP publik, Anda menggunakan parameter Aksesibilitas publik. Dengan menggunakan parameter ini, Anda dapat menetapkan apakah ada akses publik ke instans DB. Anda dapat memodifikasi instans DB untuk mengaktifkan atau menonaktifkan aksesibilitas publik dengan mengubah parameter Aksesibilitas publik.

Untuk informasi selengkapnya, lihat Menyembunyikan instans DB dalam VPC dari internet.

> **Note**
>
> Jika instans DB Anda ada dalam VPC, tetapi tidak dapat diakses publik, Anda juga dapat menggunakan koneksi AWS Site-to-Site VPN atau koneksi AWS Direct Connect untuk mengaksesnya dari jaringan pribadi. Untuk informasi selengkapnya, lihat Privasi lalu lintas antarjaringan.

API Amazon RDS dan titik akhir VPC antarmuka (AWS PrivateLink)

Anda dapat membuat koneksi privat antara titik akhir VPC dan API Amazon RDS dengan membuat titik akhir VPC antarmuka. Titik akhir antarmuka didukung oleh AWS PrivateLink.

AWS PrivateLink memungkinkan Anda mengakses operasi API Amazon RDS secara privat tanpa gateway internet, perangkat NAT, koneksi VPN, atau koneksi AWS Direct Connect. Instans DB dalam VPC Anda tidak memerlukan alamat IP publik untuk berkomunikasi dengan titik akhir API Amazon RDS untuk meluncurkan, memodifikasi, atau menghentikan instans DB dan klaster DB. Instans DB Anda juga tidak memerlukan alamat IP publik untuk menggunakan salah satu dari operasi API RDS yang tersedia. Lalu lintas antara VPC Anda dan Amazon RDS tidak keluar dari jaringan Amazon.

Setiap titik akhir antarmuka direpresentasikan oleh satu atau beberapa antarmuka jaringan elastis di subnet Anda. Untuk informasi selengkapnya tentang antarmuka jaringan elastis, lihat Antarmuka jaringan elastis dalam Panduan Pengguna Amazon EC2.

Untuk informasi selengkapnya tentang titik akhir VPC, lihat Titik akhir VPC Antarmuka () di AWS PrivateLink Panduan Pengguna Amazon VPC. Untuk informasi selengkapnya tentang operasi API RDS, lihat Referensi API Amazon RDS.

Anda tidak memerlukan titik akhir VPC antarmuka untuk terhubung ke klaster DB. Untuk informasi selengkapnya, lihat Skenario untuk mengakses klaster DB di VPC.

Pertimbangan untuk titik akhir VPC

Sebelum Anda menyiapkan titik akhir VPC antarmuka untuk titik akhir API Amazon RDS, pastikan Anda meninjau Properti dan batasan titik akhir antarmuka dalam Panduan Pengguna Amazon VPC.

Semua operasi API RDS yang relevan dengan pengelolaan sumber daya Amazon Aurora tersedia dari VPC Anda menggunakan AWS PrivateLink.

Kebijakan titik akhir VPC didukung untuk titik akhir API RDS. Secara default, akses penuh ke operasi API RDS diizinkan melalui titik akhir. Untuk informasi selengkapnya, lihat Mengontrol akses ke layanan dengan titik akhir VPC dalam Panduan Pengguna Amazon VPC.

Ketersediaan

API Amazon RDS saat ini mendukung titik akhir VPC di Wilayah AWS berikut:

- AS Timur (Ohio)
- AS Timur (Virginia Utara)
- AS Barat (California Utara)
- AS Barat (Oregon)
- Afrika (Cape Town)
- Asia Pasifik (Hong Kong)
- Asia Pasifik (Mumbai)
- Asia Pasifik (Osaka)
- Asia Pasifik (Seoul)
- Asia Pasifik (Singapura)
- Asia Pasifik (Sydney)
- Asia Pasifik (Tokyo)
- (Canada (Central)
- Kanada Barat (Calgary)
- China (Beijing)
- Tiongkok (Ningxia)
- Eropa (Frankfurt)
- Eropa (Zurich)
- Eropa (Irlandia)
- Eropa (London)
- Eropa (Paris)
- Eropa (Stockholm)
- Eropa (Milan)
- Israel (Tel Aviv)
- Timur Tengah (Bahrain)
- Amerika Selatan (Sao Paulo)
- AWS GovCloud (AS-Timur)
- AWS GovCloud (AS-Barat)

Membuat titik akhir VPC antarmuka untuk API Amazon RDS

Anda dapat membuat titik akhir VPC untuk API Amazon RDS menggunakan konsol Amazon VPC atau AWS Command Line Interface (AWS CLI). Untuk informasi selengkapnya, lihat Membuat titik akhir antarmuka dalam Panduan Pengguna Amazon VPC.

Buat titik akhir VPC untuk API Amazon RDS menggunakan nama layanan `com.amazonaws.region.rds`.

Kecuali Wilayah AWS di Tiongkok, jika Anda mengaktifkan DNS privat untuk titik akhir, Anda dapat membuat permintaan API ke Amazon RDS dengan titik akhir VPC menggunakan nama DNS default untuk Wilayah AWS, misalnya `rds.us-east-1.amazonaws.com`. Untuk Wilayah AWS Tiongkok (Beijing) dan Tiongkok (Ningxia), Anda dapat membuat permintaan API dengan titik akhir VPC menggunakan `rds-api.cn-north-1.amazonaws.com.cn` dan `rds-api.cn-northwest-1.amazonaws.com.cn`.

Untuk informasi selengkapnya, lihat Mengakses layanan melalui titik akhir antarmuka dalam Panduan Pengguna Amazon VPC.

Membuat kebijakan titik akhir VPC untuk API Amazon RDS

Anda dapat menyisipkan kebijakan titik akhir ke titik akhir VPC yang mengontrol akses ke API Amazon RDS. Kebijakan titik akhir menentukan informasi berikut:

- Prinsipal yang dapat melakukan tindakan.
- Tindakan yang dapat dilakukan.
- Sumber daya yang menjadi target tindakan.

Untuk informasi selengkapnya, lihat Mengontrol akses ke layanan dengan titik akhir VPC dalam Panduan Pengguna Amazon VPC.

Contoh: Kebijakan titik akhir VPC untuk tindakan API Amazon RDS

Berikut ini adalah contoh kebijakan titik akhir untuk API Amazon RDS. Jika dilampirkan ke sebuah titik akhir, kebijakan ini memberikan akses ke tindakan API Amazon RDS untuk semua prinsipal di semua sumber daya.

```
{
   "Statement":[
```

```
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "rds:CreateDBInstance",
            "rds:ModifyDBInstance",
            "rds:CreateDBSnapshot"
         ],
         "Resource":"*"
      }
   ]
}
```

Contoh: Kebijakan titik akhir VPC yang menolak semua akses dari akun AWS yang ditentukan

Kebijakan titik akhir VPC berikut menolak semua akses akun AWS `123456789012` ke sumber daya yang menggunakan titik akhir tersebut. Kebijakan ini memungkinkan semua tindakan dari akun lain.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": {
        "AWS": [
          "123456789012"
        ]
      }
   ]
}
```

Praktik terbaik keamanan untuk Amazon Aurora

Gunakan akun AWS Identity and Access Management (IAM) untuk mengontrol akses ke operasi API Amazon RDS, khususnya operasi yang membuat, memodifikasi, atau menghapus sumber daya

Amazon Aurora. Sumber daya tersebut termasuk klaster DB, grup keamanan, dan grup parameter. IAM juga dapat digunakan untuk mengontrol tindakan yang melakukan tindakan administratif umum seperti mencadangkan dan memulihkan kluster DB.

- Buat pengguna individual untuk setiap orang yang mengelola sumber daya Amazon Aurora, termasuk Anda sendiri. Jangan gunakan kredensial akar AWS untuk mengelola sumber daya Amazon Aurora.
- Beri setiap pengguna set izin minimum yang diperlukan untuk melakukan tugas-tugasnya.
- Gunakan grup IAM untuk mengelola izin secara efektif bagi beberapa pengguna.
- Putar kredensial IAM Anda secara rutin.
- Konfigurasikan AWS Secrets Manager untuk memutar rahasia untuk Amazon Aurora secara otomatis. Untuk informasi selengkapnya, lihat Memutar rahasia AWS Secrets Manager Anda di Panduan Pengguna AWS Secrets Manager. Anda juga dapat mengambil kredensial dari AWS Secrets Manager secara terprogram. Untuk informasi selengkapnya, lihat Mengambil nilai rahasia di Panduan Pengguna AWS Secrets Manager.

Untuk informasi selengkapnya tentang keamanan Amazon Aurora, lihat Keamanan dalam Amazon Aurora. Untuk informasi selengkapnya tentang IAM, lihat AWS Identity and Access Management. Untuk informasi tentang praktik terbaik IAM, lihat Praktik terbaik IAM.

AWS Security Hub menggunakan kontrol keamanan untuk mengevaluasi konfigurasi sumber daya dan standar keamanan untuk membantu Anda mematuhi berbagai kerangka kerja kepatuhan. Untuk informasi selengkapnya tentang penggunaan Security Hub guna mengevaluasi sumber daya RDS, lihat Kontrol Amazon Relational Database Service di Panduan Pengguna AWS Security Hub.

Anda dapat memantau penggunaan RDS yang berkaitan dengan praktik terbaik keamanan dengan menggunakan Security Hub. Untuk informasi selengkapnya tentang ini, silakan lihat Apa itu AWS Security Hub?.

Gunakan AWS Management Console, AWS CLI, atau RDS API untuk mengubah kata sandi bagi pengguna utama Anda. Jika Anda menggunakan alat lain, seperti klien SQL, untuk mengubah kata sandi pengguna utama, hak istimewa pengguna kemungkinan dapat terhapus secara tidak sengaja.

Mengontrol akses dengan grup keamanan

Grup keamanan VPC mengontrol akses lalu lintas masuk dan keluar dari klaster DB. Secara default, akses jaringan untuk klaster DB dinonaktifkan. Anda dapat menentukan aturan dalam grup

keamanan yang mengizinkan akses dari rentang alamat IP, port, atau grup keamanan. Setelah aturan masuk dikonfigurasi, aturan yang sama berlaku untuk semua klaster DB yang terkait dengan grup keamanan tersebut. Anda dapat menentukan hingga 20 aturan dalam satu grup keamanan.

Ikhtisar grup keamanan VPC

Setiap aturan grup keamanan VPC memungkinkan sumber tertentu untuk mengakses klaster DB dalam VPC yang terkait dengan grup keamanan VPC tersebut. Sumbernya dapat berupa rentang alamat (misalnya, 203.0.113.0/24), atau grup keamanan VPC lain. Dengan menentukan grup keamanan VPC sebagai sumber, Anda mengizinkan lalu lintas masuk dari semua instans (biasanya server aplikasi) yang menggunakan grup keamanan VPC sumber. Grup keamanan VPC dapat memiliki aturan yang mengatur lalu lintas masuk dan keluar. Namun, aturan lalu lintas keluar biasanya tidak berlaku untuk klaster DB. Aturan lalu lintas keluar hanya berlaku jika klaster DB bertindak sebagai klien. Anda harus menggunakan opsi API Amazon EC2 atau Grup Keamanan pada konsol VPC untuk membuat grup keamanan VPC.

Saat Anda membuat aturan untuk grup keamanan VPC yang memungkinkan akses ke klaster di VPC, Anda harus menentukan port untuk setiap rentang alamat yang diizinkan oleh aturan tersebut. Misalnya, jika Anda ingin mengaktifkan akses Secure Shell (SSH) untuk instans di VPC, buat aturan yang mengizinkan akses ke port 22 TCP untuk rentang alamat tertentu.

Anda dapat mengonfigurasi beberapa grup keamanan VPC yang mengizinkan akses ke port yang berbeda untuk instans yang berbeda di VPC Anda. Misalnya, Anda dapat membuat grup keamanan VPC yang memungkinkan akses ke port 80 TCP untuk server web di VPC Anda. Anda kemudian dapat membuat grup keamanan VPC lainnya yang memungkinkan akses ke port 3306 TCP untuk instans DB Aurora MySQL dalam VPC Anda.

> **Note**
>
> Dalam klaster DB Aurora, grup keamanan VPC yang terkait dengan klaster DB juga terkait dengan semua instans DB dalam klaster DB. Jika Anda mengubah grup keamanan VPC untuk klaster atau instans DB, perubahan akan diterapkan secara otomatis ke semua instans DB dalam klaster DB.

Untuk informasi selengkapnya tentang grup keamanan VPC, lihat Grup keamanan di Panduan Pengguna Amazon Virtual Private Cloud.

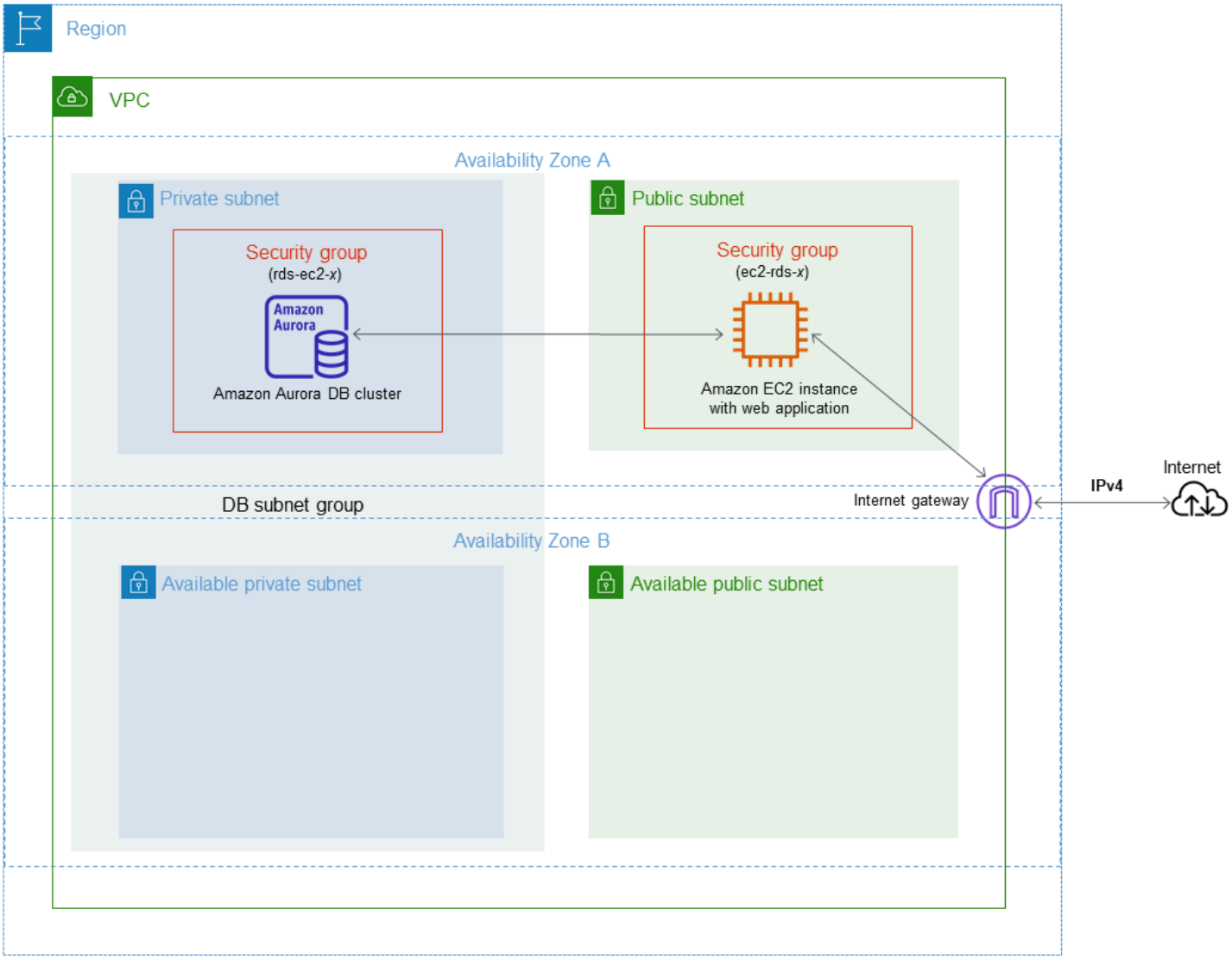
> **Note**
>
> Jika cluster DB Anda berada dalam VPC tetapi tidak dapat diakses publik, Anda juga dapat menggunakan koneksi AWS VPN Site-to-Site atau AWS Direct Connect koneksi untuk mengaksesnya dari jaringan pribadi. Untuk informasi selengkapnya, lihat Privasi lalu lintas antarjaringan.

Skenario grup keamanan

Penggunaan umum klaster DB di VPC adalah untuk berbagi data dengan server aplikasi yang dijalankan di instans Amazon EC2 dalam VPC yang sama, yang diakses oleh aplikasi klien di luar VPC. Untuk skenario ini, Anda menggunakan halaman RDS dan VPC di AWS Management Console atau operasi API RDS dan EC2 untuk membuat instans dan grup keamanan yang diperlukan:

1. Buat grup keamanan VPC (misalnya, `sg-0123ec2example`) dan tentukan aturan masuk yang menggunakan alamat IP aplikasi klien sebagai sumber. Grup keamanan ini memungkinkan aplikasi klien Anda untuk terhubung ke instans EC2 dalam VPC yang menggunakan grup keamanan ini.
2. Buat instans EC2 untuk aplikasi dan tambahkan instans EC2 ke grup keamanan VPC (`sg-0123ec2example`) yang Anda buat pada langkah sebelumnya.
3. Buat grup keamanan VPC kedua (misalnya, `sg-6789rdsexample`) dan buat aturan baru dengan menentukan grup keamanan VPC yang Anda buat di langkah 1 (`sg-0123ec2example`) sebagai sumbernya.
4. Buat klaster DB baru dan tambahkan klaster DB tersebut ke grup keamanan VPC (`sg-6789rdsexample`) yang telah Anda buat pada langkah sebelumnya. Saat Anda membuat klaster DB, gunakan nomor port yang sama dengan yang ditentukan untuk aturan grup keamanan VPC (`sg-6789rdsexample`) yang Anda buat pada langkah 3.

Diagram berikut menunjukkan skenario ini.



Untuk petunjuk lengkap tentang konfigurasi VPC untuk skenario ini, lihat Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4). Untuk informasi selengkapnya tentang penggunaan VPC, lihat Amazon VPC dan Amazon Aurora.

Membuat grup keamanan VPC

Anda dapat membuat grup keamanan VPC untuk instans DB menggunakan konsol VPC. Untuk informasi tentang pembuatan grup keamanan, lihat Berikan akses ke klaster DB dalam VPC dengan membuat grup keamanan dan Grup Keamanan dalam Panduan Pengguna Amazon Virtual Private Cloud.

Mengaitkan grup keamanan dengan klaster DB

Anda dapat mengaitkan grup keamanan dengan cluster DB menggunakan Modify cluster di konsol RDS, `ModifyDBCluster` Amazon RDS API, atau perintah. `modify-db-cluster` AWS CLI

Contoh CLI berikut mengaitkan grup VPC tertentu dan menghapus grup keamanan DB dari cluster DB

```
aws rds modify-db-cluster --db-cluster-identifier dbName --vpc-security-group-ids sg-ID
```

Untuk informasi tentang modifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Hak akses akun pengguna master

Saat Anda membuat klaster DB baru, pengguna master default yang Anda gunakan akan mendapatkan hak akses tertentu untuk klaster DB tersebut. Anda tidak dapat mengubah nama pengguna master setelah klaster DB dibuat.

> **Important**
>
> Kami sangat menyarankan agar Anda tidak menggunakan pengguna master secara langsung di aplikasi Anda. Sebagai gantinya, ikuti praktik terbaik menggunakan pengguna basis data yang dibuat dengan hak akses paling rendah yang diperlukan untuk aplikasi Anda.

> **Note**
>
> Jika Anda secara tidak sengaja menghapus izin bagi pengguna master, Anda dapat memulihkannya dengan memodifikasi klaster DB dan mengatur kata sandi pengguna master yang baru. Untuk informasi selengkapnya tentang cara memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Tabel berikut menunjukkan hak akses dan peran basis data yang diperoleh pengguna master untuk masing-masing mesin basis data.

Mesin basis data	Hak akses sistem	Peran basis data
Aurora MySQL	Versi 2: `ALTER, ALTER ROUTINE, CREATE, CREATE ROUTINE, CREATE TEMPORARY TABLES, CREATE USER, CREATE VIEW, DELETE, DROP, EVENT, EXECUTE, GRANT OPTION, INDEX, INSERT, LOAD FROM S3, LOCK TABLES, PROCESS, REFERENCES , RELOAD, REPLICATION CLIENT , REPLICATION SLAVE , SELECT, SELECT INTO S3, SHOW DATABASES , SHOW VIEW, TRIGGER, UPDATE`	—
	Versi 3: `ALTER, APPLICATION_PASSWORD_ADMIN , ALTER ROUTINE, CONNECTION_ADMIN , CREATE, CREATE ROLE, CREATE ROUTINE, CREATE TEMPORARY TABLES, CREATE USER, CREATE VIEW, DELETE, DROP, DROP ROLE, EVENT, EXECUTE, INDEX, INSERT, LOCK TABLES, PROCESS, REFERENCES , RELOAD, REPLICATION CLIENT , REPLICATION SLAVE , ROLE_ADMIN , SET_USER_ID , SELECT, SHOW DATABASES , SHOW_ROUTINE` (Aurora MySQL versi 3.04 dan yang lebih tinggi), `SHOW VIEW, TRIGGER, UPDATE, XA_RECOVER_ADMIN`	`rds_superuser_role` Untuk informasi selengkapnya tentang rds_superuser_role, lihat Model hak akses berbasis peran.
Aurora PostgreSQL	`LOGIN, NOSUPERUSER , INHERIT, CREATEDB, CREATEROLE , NOREPLICATION , VALID UNTIL 'infinity'`	`RDS_SUPERUSER` Untuk informasi selengkapnya tentang RDS_SUPERUSER, lihat Memahami peran dan izin PostgreSQL.

Menggunakan peran tertaut layanan untuk Amazon Aurora

Amazon Aurora menggunakan peran tertaut layanan AWS Identity and Access Management (IAM). Peran tertaut layanan adalah jenis peran IAM unik yang ditautkan langsung ke Amazon Aurora. Peran tertaut layanan telah ditetapkan sebelumnya oleh Amazon Aurora dan mencakup semua izin yang diperlukan layanan untuk memanggil layanan AWS lain atas nama Anda.

Peran tertaut layanan memudahkan penggunaan Amazon Aurora karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. Amazon Aurora menetapkan izin perannya yang tertaut layanan, dan kecuali ditetapkan lain, hanya Amazon Aurora yang dapat mengambil perannya. Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin, serta bahwa kebijakan izin tidak dapat dilampirkan ke entitas IAM lainnya.

Anda dapat menghapus peran hanya setelah terlebih dahulu menghapus sumber daya terkaitnya. Cara ini akan melindungi sumber daya Amazon Aurora karena Anda tidak dapat menghapus izin untuk mengakses sumber daya tersebut secara tidak sengaja.

Untuk informasi tentang layanan lain yang mendukung peran tertaut layanan, lihat Layanan AWS yang kompatibel dengan IAM dan cari layanan yang memiliki nilai Ya di dalam kolom Peran Tertaut Layanan. Pilih Ya dengan tautan untuk melihat dokumentasi peran tertaut layanan untuk layanan tersebut.

Izin peran tertaut layanan untuk Amazon Aurora

Amazon Aurora menggunakan peran tertaut layanan yang bernama AWSServiceRoleForRDS agar Amazon RDS dapat memanggil layanan AWS atas nama instans DB Anda.

Peran tertaut layanan AWSServiceRoleForRDS memercayakan layanan berikut untuk mengambil peran:

- `rds.amazonaws.com`

Peran tertaut layanan ini memiliki kebijakan izin yang menyertainya bernama `AmazonRDSServiceRolePolicy` yang memberikannya izin untuk beroperasi di akun Anda. Kebijakan izin peran memungkinkan Amazon Aurora menyelesaikan tindakan berikut pada sumber daya yang ditentukan:

Untuk informasi selengkapnya tentang kebijakan ini, termasuk dokumen kebijakan JSON, lihat AmazonRDSServiceRolePolicy dalam Panduan Referensi Kebijakan Terkelola AWS.

> **Note**
>
> Anda harus mengonfigurasi izin agar entitas IAM (seperti pengguna, grup, atau peran) dapat membuat, mengedit, atau menghapus peran tertaut layanan. Jika Anda menemukan pesan kesalahan berikut:
> Tidak dapat membuat sumber daya. Verifikasi bahwa Anda memiliki izin untuk membuat peran tertaut layanan. Jika tidak, tunggu dan coba lagi nanti.
> Pastikan Anda telah mengaktifkan izin berikut:
>
> ```
> {
>     "Action": "iam:CreateServiceLinkedRole",
>     "Effect": "Allow",
>     "Resource": "arn:aws:iam::*:role/aws-service-role/rds.amazonaws.com/
> AWSServiceRoleForRDS",
>     "Condition": {
>         "StringLike": {
>             "iam:AWSServiceName":"rds.amazonaws.com"
>         }
>     }
> }
> ```
>
> Untuk informasi selengkapnya, lihat Izin peran tertaut layanan dalam Panduan Pengguna IAM.

Membuat peran tertaut layanan untuk Amazon Aurora

Anda tidak perlu membuat peran tertaut layanan secara manual. Saat Anda membuat instans DB, Amazon Aurora membuat peran tertaut layanan untuk Anda.

> **Important**
>
> Jika Anda menggunakan layanan Amazon Aurora sebelum 1 Desember 2017, saat layanan tersebut mulai mendukung peran tertaut layanan, Amazon Aurora akan membuat peran AWSServiceRoleForRDS di akun Anda. Untuk mempelajari lebih lanjut, lihat Peran baru yang muncul di akun AWS saya.

Jika Anda menghapus peran tertaut layanan ini, dan ingin membuatnya lagi, Anda dapat mengulangi proses yang sama untuk membuat kembali peran tersebut di akun Anda. Saat Anda membuat instans DB, Amazon Aurora membuat peran tertaut layanan untuk Anda.

Mengedit peran tertaut layanan untuk Amazon Aurora

Amazon Aurora tidak mengizinkan Anda untuk mengedit peran tertaut layanan AWSServiceRoleForRDS. Setelah membuat peran tertaut layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin mereferensikan peran tersebut. Namun, Anda dapat mengedit deskripsi peran menggunakan IAM. Untuk informasi selengkapnya, lihat Mengedit peran tertaut layanan dalam Panduan Pengguna IAM.

Menghapus peran tertaut layanan untuk Amazon Aurora

Jika Anda tidak perlu lagi menggunakan fitur atau layanan yang memerlukan peran tertaut layanan, sebaiknya Anda menghapus peran tersebut. Dengan demikian, Anda tidak memiliki entitas tidak terpakai yang tidak dipantau atau dipelihara secara aktif. Namun, Anda harus menghapus semua klaster DB sebelum Anda dapat menghapus peran tertaut layanan.

Membersihkan peran tertaut layanan

Sebelum Anda dapat menggunakan IAM untuk menghapus peran tertaut layanan, Anda harus mengonfirmasi terlebih dahulu bahwa peran tersebut tidak memiliki sesi aktif dan menghapus sumber daya yang digunakan oleh peran tersebut.

Untuk memastikan peran tertaut layanan memiliki sesi aktif di konsol IAM

1. Masuk ke AWS Management Console dan buka konsol IAM di https://console.aws.amazon.com/iam/.
2. Di panel navigasi konsol IAM, pilih Peran. Kemudian, pilih nama (bukan kotak centang) peran AWSServiceRoleForRDS.
3. Di halaman Ringkasan untuk peran yang dipilih, pilih tab Penasihat Akses.
4. Pada tab Penasihat Akses, tinjau aktivitas terbaru untuk peran tertaut layanan tersebut.

> **Note**
>
> Jika Anda tidak yakin apakah Amazon Aurora menggunakan peran AWSServiceRoleForRDS tersebut, coba hapus peran tersebut. Jika layanan menggunakan peran tersebut, peran tidak dapat dihapus dan Anda dapat melihat

Wilayah AWS tempat peran tersebut digunakan. Jika peran tersebut sedang digunakan, Anda harus menunggu hingga sesi ini berakhir sebelum dapat menghapus peran tersebut. Anda tidak dapat mencabut sesi untuk peran tertaut layanan.

Jika Anda ingin menghapus peran AWSServiceRoleForRDS, Anda harus terlebih dahulu menghapus semua instans DB Anda.

Menghapus semua klaster

Gunakan salah satu prosedur berikut untuk menghapus satu klaster. Ulangi prosedur untuk setiap klaster Anda.

Untuk menghapus klaster (konsol)

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Dalam daftar Basis data pilih klaster yang ingin Anda hapus.
3. Untuk Tindakan Klaster, pilih Hapus.
4. Pilih Hapus.

Untuk menghapus klaster (CLI)

Lihat `delete-db-cluster` dalam Referensi Perintah AWS CLI.

Untuk menghapus klaster (API)

Lihat `DeleteDBCluster` dalam Referensi API Amazon RDS.

Anda dapat menggunakan konsol IAM, IAM CLI, atau IAM API untuk menghapus peran tertaut layanan AWSServiceRoleForRDS. Untuk informasi selengkapnya, lihat Menghapus peran tertaut layanan dalam Panduan Pengguna IAM.

Amazon VPC dan Amazon Aurora

Amazon Virtual Private Cloud (Amazon VPC) memungkinkan Anda meluncurkan sumber daya AWS, seperti klaster DB Aurora, ke cloud privat virtual (VPC).

Saat menggunakan VPC, Anda dapat mengontrol lingkungan jaringan virtual Anda. Anda dapat memilih rentang alamat IP Anda sendiri, membuat subnet, dan mengonfigurasi daftar kontrol akses dan perutean. Tidak ada biaya tambahan untuk menjalankan instans DB Anda dalam VPC.

Akun memiliki VPC default. Semua instans DB baru dibuat dalam VPC default kecuali Anda menentukan lain.

Topik

- Bekerja dengan instans DB dalam VPC
- Skenario untuk mengakses klaster DB di VPC
- Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4)
- Tutorial: Membuat VPC untuk digunakan dengan klaster DB (mode tumpukan ganda)

Berikut ini, Anda dapat menemukan diskusi tentang fungsi VPC yang relevan dengan klaster DB Amazon Aurora. Untuk informasi selengkapnya tentang Amazon VPC, lihat Panduan Memulai Amazon VPC dan Panduan Pengguna Amazon VPC.

Bekerja dengan instans DB dalam VPC

instans DB Anda berada di cloud privat virtual (VPC). VPC adalah jaringan virtual yang secara logis terisolasi dari jaringan virtual lain di Cloud AWS. Amazon VPC memungkinkan Anda meluncurkan sumber daya AWS, seperti instans DB Amazon Aurora atau instans Amazon EC2, ke VPC. VPC dapat berupa VPC default dari akun Anda atau VPC yang Anda buat. Semua VPC dikaitkan dengan akun AWS Anda.

VPC default memiliki tiga subnet yang dapat digunakan untuk mengisolasi sumber daya di dalam VPC. VPC default juga memiliki gateway internet yang dapat digunakan untuk memberikan akses ke sumber daya di dalam VPC dari luar VPC.

Untuk daftar skenario yang melibatkan instans DB Amazon Aurora di dalam VPC , lihat Skenario untuk mengakses klaster DB di VPC.

Topik

- Bekerja dengan instans DB dalam VPC
- Bekerja dengan grup subnet DB
- Subnet bersama
- Penentuan alamat IP Amazon Aurora
- Menyembunyikan instans DB dalam VPC dari internet
- Membuat instans DB dalam VPC

Dalam tutorial berikut, Anda dapat mempelajari cara membuat VPC yang dapat Anda gunakan untuk skenario Amazon Aurora yang umum:

- Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4)
- Tutorial: Membuat VPC untuk digunakan dengan klaster DB (mode tumpukan ganda)

Bekerja dengan instans DB dalam VPC

Berikut adalah beberapa tips cara bekerja dengan instans DB dalam VPC:

- VPC Anda harus memiliki minimal dua subnet. Subnet ini harus berada di dua Zona Ketersediaan yang berbeda di Wilayah AWS tempat Anda ingin men-deploy instans DB. Subnet adalah segmen dari rentang alamat IP VPC yang dapat Anda tentukan dan gunakan untuk mengelompokkan instans DB berdasarkan kebutuhan keamanan dan operasional Anda.
- Jika Anda ingin instans DB dalam VPC dapat diakses publik, pastikan untuk mengaktifkan nama host DNS dan resolusi DNS atribut VPC.
- VPC Anda harus memiliki grup subnet DB yang Anda buat. Anda membuat grup subnet DB dengan menentukan subnet yang telah Anda buat. Amazon Aurora memilih subnet dan alamat IP dalam subnet tersebut untuk dikaitkan dengan instans DB utama di klaster DB Anda. Instans DB utama menggunakan Zona Ketersediaan yang berisi subnet.
- VPC Anda harus memiliki grup keamanan VPC yang memungkinkan akses ke instans DB.

  Untuk informasi selengkapnya, lihat Skenario untuk mengakses klaster DB di VPC.
- Blok CIDR di setiap subnet Anda harus cukup besar untuk mengakomodasi alamat IP cadangan untuk Amazon Aurora untuk digunakan selama aktivitas pemeliharaan, termasuk failover dan penskalaan komputasi. Misalnya, rentang seperti 10.0.0.0/24 dan 10.0.1.0/24 biasanya memiliki kapasitas cukup besar.

- VPC dapat memiliki atribut penghunian instans, baik yang bersifat default atau khusus. Semua VPC default memiliki atribut penghunian instans yang ditetapkan ke default, dan VPC default dapat mendukung kelas instans DB mana pun.

  Jika Anda memilih untuk menempatkan instans DB dalam VPC khusus tempat atribut penghunian instans ditetapkan ke khusus, maka kelas instans DB untuk instans DB Anda harus merupakan salah satu jenis instans khusus Amazon EC2 yang disetujui. Misalnya, instans khusus EC2 r5.large sesuai dengan kelas instans DB db.r5.large. Untuk informasi tentang penghunian instans dalam VPC, lihat Instans khusus dalam Panduan Pengguna Amazon Elastic Compute Cloud.

  Untuk informasi selengkapnya tentang jenis instans yang dapat berada dalam instans khusus, lihat Instans khusus Amazon EC2 di halaman harga EC2.

  > **Note**
  >
  > Jika Anda menetapkan atribut penghunian instans ke khusus untuk instans DB, hal tersebut tidak menjamin bahwa instans DB akan berjalan di host khusus.

Bekerja dengan grup subnet DB

Subnet adalah segmen dari rentang alamat IP VPC yang Anda tetapkan untuk mengelompokkan sumber daya Anda berdasarkan kebutuhan keamanan dan operasional. Grup subnet DB adalah kumpulan subnet (biasanya privat) yang Anda buat dalam VPC dan kemudian Anda tetapkan untuk instans DB Anda. Dengan menggunakan grup subnet DB, Anda dapat menentukan VPC tertentu saat membuat instans menggunakan AWS CLI atau API RDS. Jika menggunakan konsol, Anda dapat memilih grup VPC dan subnet yang ingin Anda gunakan.

Setiap grup subnet DB harus memiliki subnet minimal di dua Zona Ketersediaan dalam Wilayah AWS tertentu. Saat membuat instans DB dalam VPC, Anda memilih grup subnet DB untuk klaster tersebut. Dari grup subnet DB, Amazon Aurora memilih subnet dan alamat IP dalam subnet tersebut untuk dikaitkan dengan instans DB utama di klaster DB Anda. DB menggunakan Zona Ketersediaan yang berisi subnet.

Subnet dalam grup subnet DB bersifat publik atau privat. Subnet bersifat umum atau privat, bergantung pada konfigurasi yang Anda tetapkan untuk daftar kontrol akses jaringan (network ACL) dan tabel perutean. Agar instans DB dapat diakses publik, semua subnet dalam grup subnet DB-nya harus bersifat publik. Jika subnet yang terkait dengan instans DB yang dapat diakses publik berubah dari publik menjadi pribadi, hal tersebut dapat memengaruhi ketersediaan instans DB.

Untuk membuat grup subnet DB yang mendukung mode dual-stack, pastikan setiap subnet yang Anda tambahkan ke grup subnet DB memiliki blok CIDR Internet Protocol versi 6 (IPv6) yang terkait dengannya. Untuk informasi selengkapnya, lihat Penentuan alamat IP Amazon Aurora dan Bermigrasi ke IPv6 dalam Panduan Pengguna Amazon VPC.

Saat Amazon Aurora membuat instans DB dalam VPC, antarmuka jaringan ditetapkan ke instans DB dengan menggunakan alamat IP dari grup subnet DB. Namun, kami sangat menyarankan agar Anda menggunakan nama Sistem Nama Domain (DNS) untuk terhubung ke instans DB. Kami merekomendasikan hal ini karena alamat IP pokok berubah selama failover.

> **Note**
>
> Untuk setiap instans DB yang Anda jalankan dalam VPC, pastikan untuk menyimpan minimal satu alamat di setiap subnet dalam grup subnet DB agar dapat digunakan oleh Amazon Aurora untuk tindakan pemulihan.

Subnet bersama

Anda dapat membuat instans DB dalam VPC bersama.

Beberapa pertimbangan yang perlu diingat saat menggunakan VPC bersama:

- Anda dapat memindahkan instans DB dari subnet VPC bersama ke subnet VPC privat dan sebaliknya.
- Peserta dalam VPC bersama harus membuat grup keamanan dalam VPC agar mereka dapat membuat instans DB.
- Pemilik dan peserta dalam VPC bersama dapat mengakses basis data dengan menggunakan kueri SQL. Namun, hanya pembuat sumber daya yang dapat melakukan panggilan API di sumber daya.

Penentuan alamat IP Amazon Aurora

Penentuan alamat IP memungkinkan sumber daya dalam VPC Anda berkomunikasi satu sama lain, dan dengan sumber daya melalui internet. Amazon Aurora mendukung protokol penentuan alamat IPv4 dan IPv6. Secara default, Amazon Aurora dan Amazon VPC menggunakan protokol penentuan alamat IPv4. Anda tidak dapat menonaktifkan perilaku ini. Saat Anda membuat VPC, pastikan untuk menentukan blok CIDR IPv4 (rentang alamat IPv4 privat). Atau, Anda dapat menetapkan blok CIDR

IPv6 ke VPC dan subnet Anda, serta menetapkan alamat IPv6 dari blok tersebut ke instans DB di subnet Anda.

Dukungan untuk protokol IPv6 memperbesar jumlah alamat IP yang didukung. Dengan menggunakan protokol IPv6, Anda memastikan bahwa Anda memiliki ketersediaan alamat yang memadai untuk menghadapi pertumbuhan internet di masa mendatang. Sumber daya RDS baru dan yang sudah ada dapat menggunakan alamat IPv4 dan IPv6 dalam VPC Anda. Mengonfigurasi, mengamankan, dan menerjemahkan lalu lintas jaringan di antara dua protokol yang digunakan di berbagai bagian aplikasi dapat menimbulkan overhead operasional. Anda dapat melakukan standardisasi pada protokol IPv6 bagi sumber daya Amazon RDS untuk menyederhanakan konfigurasi jaringan Anda.

Topik

- Alamat IPv4
- Alamat IPv6
- Mode dual-stack

Alamat IPv4

Saat membuat VPC, Anda harus menentukan rentang alamat IPv4 untuk VPC dalam bentuk blok CIDR, seperti `10.0.0.0/16`. Grup subnet DB mendefinisikan rentang alamat IP di blok CIDR ini yang dapat digunakan oleh instans DB. Alamat IP ini bisa bersifat privat atau publik.

Alamat IPv4 privat adalah alamat IP yang tidak dapat diakses melalui internet. Anda dapat menggunakan alamat IPv4 privat untuk komunikasi di antara instans DB Anda dan sumber daya lainnya, seperti instans Amazon EC2, dalam VPC yang sama. Setiap instans DB memiliki alamat IP privat untuk komunikasi dalam VPC.

Alamat IP publik adalah alamat IPv4 yang dapat diakses dari internet. Anda dapat menggunakan alamat publik untuk komunikasi di antara instans DB dan sumber daya di internet, seperti klien SQL. Anda mengontrol apakah instans DB Anda menerima alamat IP publik.

Untuk tutorial cara membuat VPC hanya dengan alamat IPv4 privat yang dapat Anda gunakan untuk skenario Amazon Aurora yang umum, lihat Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4).

Alamat IPv6

Atau, Anda dapat memilih mengaitkan blok CIDR IPv6 ke VPC dan subnet, serta menetapkan alamat IPv6 dari blok tersebut ke sumber daya dalam VPC Anda. Setiap alamat IPv6 bersifat unik secara global.

Blok CIDR IPv6 untuk VPC Anda secara otomatis ditetapkan dari kumpulan alamat IPv6 Amazon. Anda tidak dapat memilih sendiri rentang tersebut.

Saat menghubungkan ke alamat IPv6, pastikan kondisi berikut terpenuhi:

- Klien dikonfigurasi, sehingga lalu lintas klien ke basis data melalui IPv6 dimungkinkan.
- Grup keamanan RDS yang digunakan oleh instans DB dikonfigurasi dengan benar, sehingga lalu lintas klien ke basis data melalui IPv6 dimungkinkan.
- Stack sistem operasi klien memungkinkan lalu lintas pada alamat IPv6, dan driver serta pustaka sistem operasi dikonfigurasi untuk memilih titik akhir instans DB default yang benar (yaitu, IPv4 atau IPv6).

Untuk informasi selengkapnya tentang IPv6, lihat Penentuan alamat IP di Panduan Pengguna Amazon VPC.

Mode dual-stack

Ketika instans DB dapat berkomunikasi melalui protokol penentuan alamat IPv4 dan IPv6, mode dual-stack berarti digunakan. Jadi, sumber daya dapat berkomunikasi dengan instans DB melalui IPv4, IPv6, atau keduanya. RDS menonaktifkan akses Gateway Internet untuk titik akhir IPv6 dari instans DB mode dual-stack privat. RDS melakukan hal ini untuk memastikan titik akhir IPv6 Anda bersifat privat dan hanya dapat diakses dari dalam VPC Anda.

Topik

- Mode dual-stack dan grup subnet DB
- Bekerja dengan instans DB mode dual-stack
- Memodifikasi instans DB khusus IPv4 untuk menggunakan mode dual-stack
- Ketersediaan instans DB jaringan dual-stack
- Batasan untuk instans DB jaringan dual-stack

Untuk tutorial cara membuat VPC dengan alamat IPv4 dan IPv6 yang dapat Anda gunakan untuk skenario Amazon Aurora yang umum, lihat Tutorial: Membuat VPC untuk digunakan dengan klaster DB (mode tumpukan ganda).

Mode dual-stack dan grup subnet DB

Untuk menggunakan mode dual-stack, pastikan setiap subnet dalam grup subnet DB yang Anda kaitkan dengan instans DB memiliki blok CIDR IPv6 yang terkait dengannya. Anda dapat membuat grup subnet DB baru atau memodifikasi grup subnet DB yang ada untuk memenuhi persyaratan ini. Setelah instans DB berada dalam mode dual-stack, klien dapat terhubung secara normal. Pastikan firewall keamanan klien dan grup keamanan instans RDS DB dikonfigurasi secara akurat untuk memungkinkan lalu lintas melalui IPv6. Untuk terhubung, klien menggunakan Titik akhir instans utama klaster DB. Aplikasi klien dapat menentukan protokol mana yang lebih disukai saat terhubung ke basis data. Dalam mode dual-stack, instans DB mendeteksi protokol jaringan pilihan klien, baik IPv4 maupun IPv6, dan menggunakan protokol tersebut untuk koneksi.

Jika grup subnet DB berhenti mendukung mode dual-stack karena penghapusan subnet atau pembatalan pengaitan CIDR, ada risiko status jaringan menjadi tidak kompatibel untuk instans DB yang terkait dengan grup subnet DB. Selain itu, Anda tidak dapat menggunakan grup subnet DB saat membuat instans DB mode dual-stack yang baru.

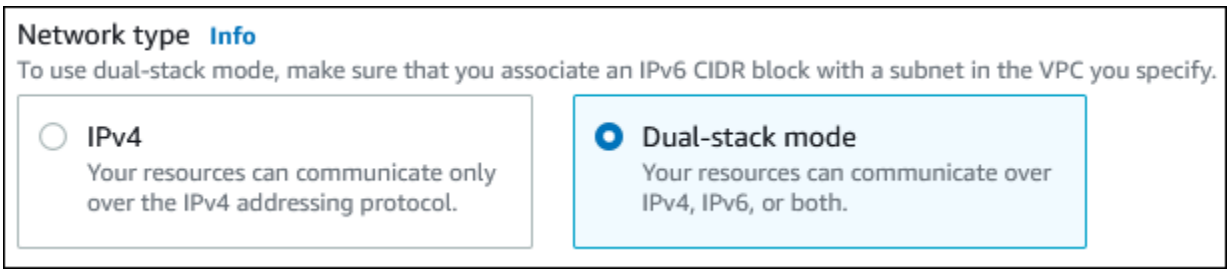
Untuk menentukan apakah grup subnet DB mendukung mode dual-stack dengan menggunakan AWS Management Console, lihat Jenis jaringan pada halaman detail grup subnet DB. Untuk menentukan apakah grup subnet DB mendukung mode dual-stack dengan menggunakanAWS CLI, jalankan describe-db-subnet-groupsperintah dan lihat `SupportedNetworkTypes` di output.

Replika baca dianggap sebagai instans DB independen dan dapat memiliki jenis jaringan yang berbeda dengan instans DB utama. Jika Anda mengubah jenis jaringan untuk instans DB utama replika baca, replika baca tidak akan terpengaruh. Saat memulihkan instans DB, Anda dapat memulihkannya ke jenis jaringan apa pun yang didukung.

Bekerja dengan instans DB mode dual-stack

Saat membuat atau memodifikasi instans DB, Anda dapat menentukan mode dual-stack agar sumber daya Anda dapat berkomunikasi dengan instans DB melalui IPv4, IPv6, atau keduanya.

Jika Anda menggunakan AWS Management Console untuk membuat atau memodifikasi instans DB, Anda dapat menentukan mode dual-stack di bagian Jenis jaringan. Gambar berikut menampilkan bagian Jenis jaringan di konsol.



Saat Anda menggunakan AWS CLI untuk membuat atau memodifikasi instans DB, tetapkan opsi `--network-type` ke `DUAL` untuk menggunakan mode dual-stack. Saat Anda menggunakan API RDS untuk membuat atau memodifikasi instans DB, tetapkan parameter `NetworkType` ke `DUAL` untuk menggunakan mode dual-stack. Saat Anda memodifikasi jenis jaringan instans DB, mungkin terjadi waktu henti. Jika mode dual-stack tidak didukung oleh versi mesin DB atau grup subnet DB yang ditentukan, kesalahan `NetworkTypeNotSupported` akan ditampilkan.

Untuk informasi selengkapnya tentang cara membuat klaster DB, lihat Membuat klaster DB Amazon Aurora. Untuk informasi selengkapnya tentang cara memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Untuk menentukan apakah instans DB berada di mode dual-stack dengan menggunakan konsol, lihat Jenis jaringan di tab Konektivitas & keamanan untuk instans DB.

Memodifikasi instans DB khusus IPv4 untuk menggunakan mode dual-stack

Anda dapat memodifikasi instans DB khusus IPv4 untuk menggunakan mode dual-stack. Untuk melakukannya, ubah jenis jaringan instans DB. Modifikasi dapat mengakibatkan waktu henti.

Sebaiknya Anda mengubah jenis jaringan klaster DB Amazon Aurora selama periode pemeliharaan. Saat ini, pengaturan jenis jaringan instans baru ke mode dual-stack tidak didukung. Anda dapat mengatur jenis jaringan secara manual dengan menggunakan perintah `modify-db-cluster`.

Sebelum memodifikasi instans DB untuk menggunakan mode dual-stack, pastikan grup subnet DB-nya mendukung mode dual-stack. Jika grup subnet DB yang terkait dengan instans DB tidak mendukung mode dual-stack, tentukan grup subnet DB yang berbeda yang mendukungnya saat Anda memodifikasi instans DB. Memodifikasi grup subnet DB dari instans DB dapat menyebabkan waktu henti.

Jika Anda memodifikasi grup subnet DB dari instans DB sebelum mengubah instans DB untuk menggunakan mode dual-stack, pastikan grup subnet DB valid untuk instans DB sebelum dan sesudah perubahan.

Kami menyarankan Anda menjalankan modify-db-clusterAPI hanya dengan `--network-type` parameter dengan nilai `DUAL` untuk mengubah jaringan cluster Amazon Aurora ke mode dual-stack. Menambahkan parameter lain bersama dengan parameter `--network-type` dalam panggilan API yang sama dapat mengakibatkan waktu henti.

Jika Anda tidak dapat terhubung ke instans DB setelah perubahan, pastikan firewall keamanan klien dan basis data serta tabel rute dikonfigurasi secara akurat agar lalu lintas ke basis data dimungkinkan pada jaringan yang dipilih (baik IPv4 maupun IPv6). Anda mungkin juga perlu memodifikasi parameter sistem operasi, pustaka, atau driver untuk terhubung menggunakan alamat IPv6.

Anda dapat memodifikasi instans DB khusus IPv4 untuk menggunakan mode dual-stack

1. Modifikasi grup subnet DB untuk mendukung mode dual-stack, atau buat grup subnet DB yang mendukung mode dual-stack:

   a. Kaitkan blok CIDR IPv6 dengan VPC Anda.

      Untuk mendapatkan petunjuk, lihat Menambahkan blok CIDR IPv6 ke VPC di Panduan Pengguna Amazon VPC.

   b. Tambahkan blok CIDR IPv6 ke semua subnet di grup subnet DB Anda.

      Untuk mendapatkan petunjuk, lihat Menambahkan blok CIDR IPv6 ke subnet di Panduan Pengguna Amazon VPC.

   c. Konfirmasikan bahwa grup subnet DB mendukung mode dual-stack.

      Jika Anda menggunakan AWS Management Console, pilih grup subnet DB, dan pastikan bahwa nilai Jenis jaringan yang didukung adalah Dual, IPv4.

      Jika Anda menggunakanAWS CLI, jalankan describe-db-subnet-groupsperintah, dan pastikan bahwa `SupportedNetworkType` nilai untuk instans DB adalah`Dual, IPv4`.

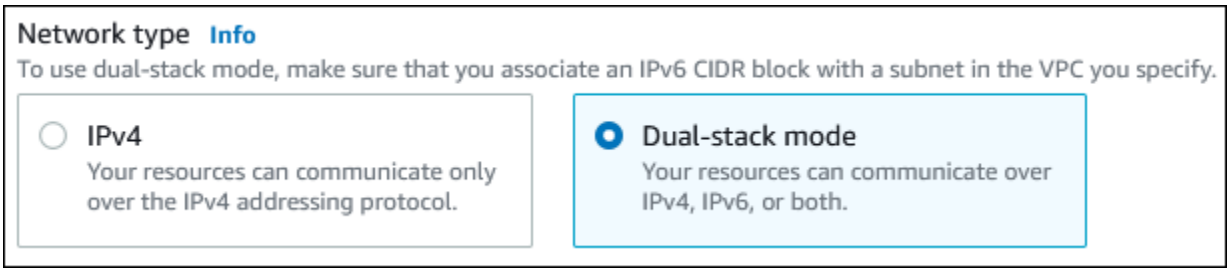
2. Modifikasi grup keamanan yang terkait dengan instans DB agar IPv6 dapat dihubungkan ke basis data, atau buat grup keamanan baru agar IPv6 dapat terhubung.

   Untuk mendapatkan petunjuk, lihat Aturan grup keamanan di Panduan Pengguna Amazon VPC.

3. Modifikasi instans DB untuk mendukung mode dual-stack. Untuk melakukannya, tetapkan Jenis jaringan ke Mode dual-stack.

   Jika Anda menggunakan konsol, pastikan pengaturan berikut sudah benar:

- Jenis jaringan – Mode dual-stack



- Grup subnet DB – Grup subnet DB yang telah Anda konfigurasikan pada langkah sebelumnya
- Grup keamanan — Keamanan yang telah Anda konfigurasikan pada langkah sebelumnya

Jika Anda menggunakan AWS CLI, pastikan pengaturan berikut sudah benar:

- `--network-type` – `dual`
- `--db-subnet-group-name` – Grup subnet DB yang telah Anda konfigurasikan pada langkah sebelumnya
- `--vpc-security-group-ids` – Grup keamanan VPC yang telah Anda konfigurasikan pada langkah sebelumnya

Contoh:

```
aws rds modify-db-cluster --db-cluster-identifier my-cluster --network-type "DUAL"
```

4. Konfirmasikan bahwa instans DB mendukung mode dual-stack.

   Jika Anda menggunakan konsol, pilih tab Konfigurasi untuk instans DB. Pada tab tersebut, pastikan nilai Jenis jaringan adalah Mode dual-stack.

   Jika Anda menggunakanAWS CLI, jalankan describe-db-clustersperintah, dan pastikan bahwa `NetworkType` nilai untuk cluster DB adalah`dual`.

   Jalankan perintah `dig` pada titik akhir instans DB penulis untuk mengidentifikasi alamat IPv6 yang terkait dengannya.

   ```
   dig db-instance-endpoint AAAA
   ```

   Gunakan titik akhir instans DB penulis, bukan alamat IPv6, untuk terhubung ke instans DB.

Ketersediaan instans DB jaringan dual-stack

Versi mesin DB berikut mendukung kluster DB jaringan dual-stack, kecuali di Asia Pasifik (Hyderabad), Asia Pasifik (Melbourne), Kanada Barat (Calgary), Eropa (Spanyol), Eropa (Zurich), Israel (Tel Aviv), dan Timur Tengah (UEA) Wilayah:

- Versi Aurora MySQL:
  - Versi 3.02 dan versi 3 yang lebih tinggi
  - Versi 2.09.1 dan versi 2 yang lebih tinggi

  Untuk informasi selengkapnya tentang versi Aurora MySQL, lihat Catatan Rilis untuk Aurora MySQL.
- Versi Aurora PostgreSQL:
  - Versi 14.3 dan versi 14 yang lebih tinggi
  - Versi 13.7 dan versi 13 yang lebih tinggi

  Untuk informasi selengkapnya tentang versi Aurora PostgreSQL, lihat Catatan Rilis untuk Aurora PostgreSQL.

Batasan untuk instans DB jaringan dual-stack

Batasan berikut berlaku untuk instans DB jaringan dual-stack:

- instans DB tidak dapat menggunakan protokol IPv6 secara eksklusif. Klaster tersebut dapat menggunakan IPv4 secara eksklusif, atau menggunakan protokol IPv4 dan IPv6 (mode dual-stack).
- Amazon RDS tidak mendukung subnet IPv6 asli.
- instans DB yang menggunakan mode dual-stack harus privat. Klaster tersebut tidak dapat diakses publik.
- Mode dual-stack tidak mendukung kelas instans DB db.r3.
- Anda tidak dapat menggunakan Proksi RDS dengan instans DB mode dual-stack.

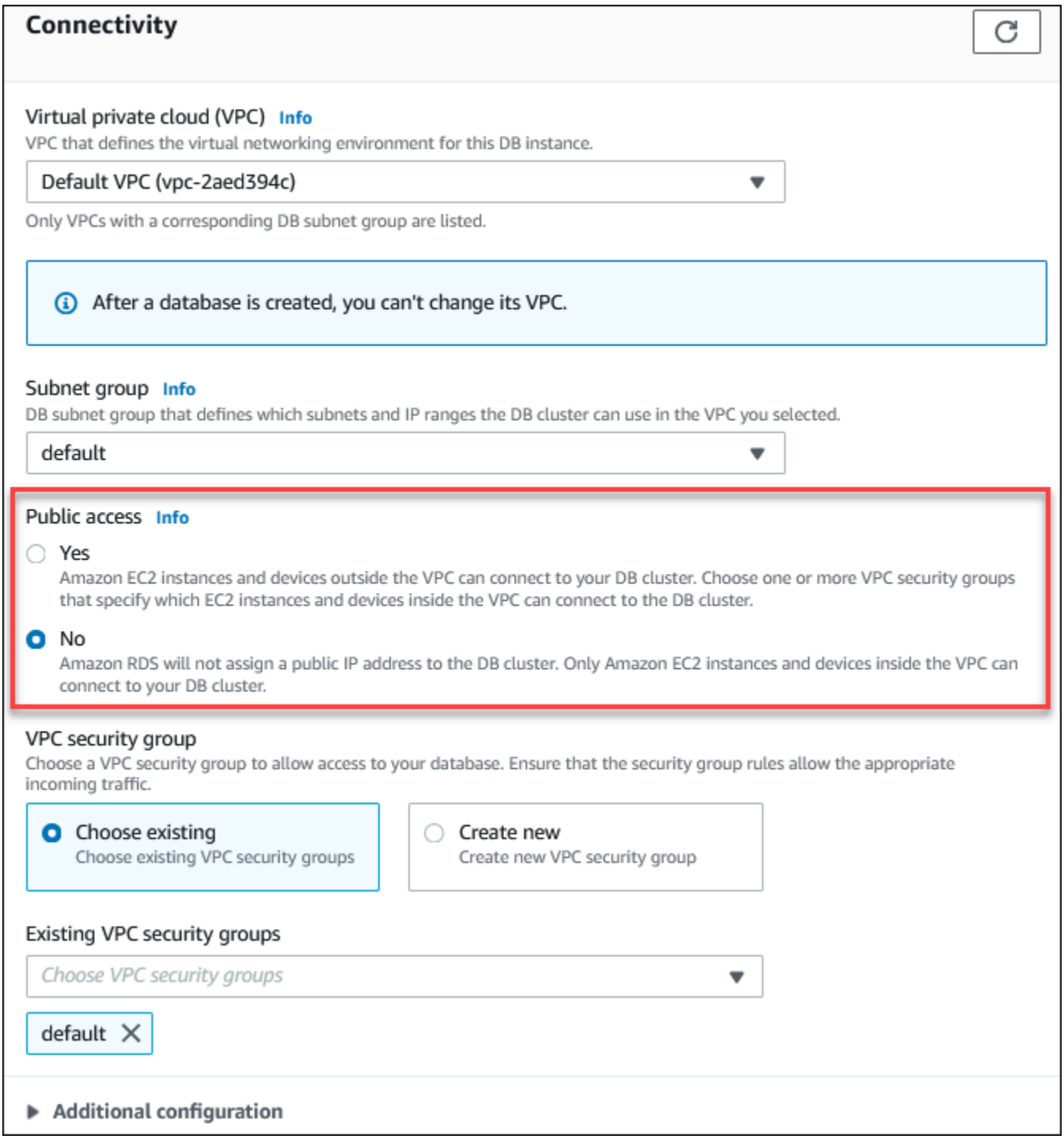
Menyembunyikan instans DB dalam VPC dari internet

Satu skenario Amazon Aurora yang umum adalah memiliki VPC yang di dalamnya ada instans EC2 dengan aplikasi web untuk publik dan instans DB dengan basis data yang tidak dapat diakses publik. Misalnya, Anda dapat membuat VPC yang memiliki subnet publik dan subnet privat. Instans Amazon

EC2 yang berfungsi sebagai server web dapat di-deploy di subnet publik. instans DB di-deploy di subnet privat. Dalam deployment seperti itu, hanya server web yang memiliki akses ke instans DB. Untuk ilustrasi skenario ini, lihat klaster DB di VPC yang diakses oleh instans EC2 dalam VPC yang sama.

Saat Anda meluncurkan instans DB di dalam VPC, instans DB memiliki alamat IP privat untuk lalu lintas di dalam VPC. Alamat IP privat ini tidak dapat diakses publik. Anda dapat menggunakan opsi Akses publik untuk menetapkan apakah instans DB juga memiliki alamat IP publik selain alamat IP privat. Jika instans DB ditetapkan sebagai dapat diakses publik, titik akhir DNS-nya akan diselesaikan ke alamat IP privat dari dalam VPC. Serta ke alamat IP publik dari luar VPC. Akses ke instans DB pada akhirnya dikontrol oleh grup keamanan yang digunakannya. Akses publik tersebut tidak diizinkan jika grup keamanan yang ditetapkan ke instans DB tidak mencakup aturan masuk yang mengizinkannya. Selain itu, agar instans DB dapat diakses publik, subnet dalam grup subnet DB-nya harus memiliki gateway internet. Lihat informasi yang lebih lengkap di Tidak dapat terhubung ke instans DB Amazon RDS

Anda dapat memodifikasi instans DB untuk mengaktifkan atau menonaktifkan aksesibilitas publik dengan memodifikasi opsi Akses publik. Ilustrasi berikut ini menunjukkan opsi Akses publik di bagian Konfigurasi konektivitas tambahan. Untuk menetapkan opsi tersebut, buka bagian Konfigurasi konektivitas tambahan di bagian Konektivitas.



Untuk informasi tentang cara memodifikasi instans DB untuk menetapkan opsi Akses publik, lihat Memodifikasi instans DB dalam klaster DB.

Membuat instans DB dalam VPC

Prosedur berikut membantu Anda membuat instans DB dalam VPC. Untuk menggunakan VPC default, Anda dapat memulai dengan langkah 2, dan menggunakan grup subnet VPC dan DB yang sudah dibuat untuk Anda. Jika ingin membuat VPC tambahan, Anda dapat membuat VPC baru.

> **Note**
>
> Jika Anda ingin instans DB dalam VPC dapat diakses publik, Anda harus memperbarui informasi DNS untuk VPC dengan mengaktifkan nama host DNS dan resolusi DNS atribut VPC. Untuk informasi tentang cara memperbarui informasi DNS untuk instans VPC, lihat Memperbarui dukungan DNS untuk VPC Anda.

Ikuti langkah-langkah berikut untuk membuat instans DB dalam VPC:

- Langkah 1: Buat VPC
- Langkah 2: Buat grup subnet DB
- Langkah 3: Buat grup keamanan VPC
- Langkah 4: Buat instans DB dalam VPC

Langkah 1: Buat VPC

Buat VPC dengan subnet di minimal dua Zona Ketersediaan. Anda menggunakan subnet ini saat membuat grup subnet DB. Jika Anda memiliki VPC default, subnet secara otomatis dibuat untuk Anda dalam setiap Zona Ketersediaan di Wilayah AWS.

Untuk informasi selengkapnya, lihat Membuat VPC dengan subnet publik dan privat, atau lihat Membuat VPC dalam Panduan Pengguna Amazon VPC.

Langkah 2: Buat grup subnet DB

Grup subnet DB adalah kumpulan subnet (biasanya privat) yang Anda buat untuk VPC dan kemudian ditetapkan untuk instans DB Anda. Grup subnet DB memungkinkan Anda menetapkan VPC tertentu saat Anda membuat instans DB menggunakan AWS CLI atau API RDS. Jika menggunakan konsol, Anda dapat memilih grup VPC dan subnet yang ingin digunakan. Setiap grup subnet DB harus memiliki minimal satu subnet di minimal dua Zona Ketersediaan di Wilayah AWS. Sebagai praktik

terbaik, setiap grup subnet DB harus memiliki minimal satu subnet untuk setiap Zona Ketersediaan di Wilayah AWS.

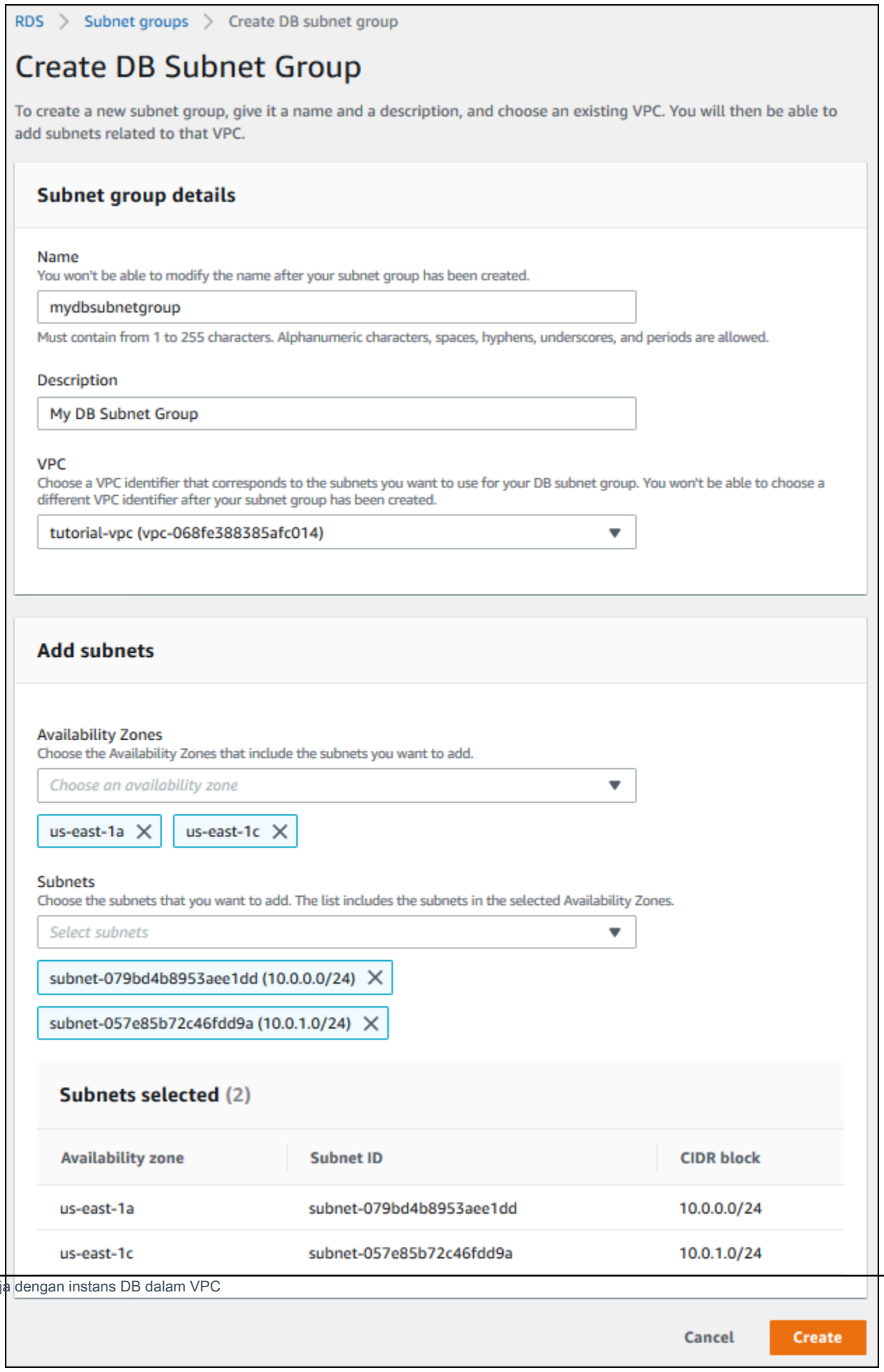
Agar instans DB dapat diakses publik, subnet dalam grup subnet DB harus memiliki gateway internet. Untuk informasi selengkapnya tentang gateway internet untuk subnet, lihat Terhubung ke internet menggunakan gateway internet dalam Panduan Pengguna Amazon VPC.

Saat membuat instans DB dalam VPC, Anda dapat memilih grup subnet DB. Amazon Aurora memilih subnet dan alamat IP dalam subnet tersebut untuk dikaitkan dengan instans DB Anda. Jika tidak ada grup subnet DB, Amazon Aurora membuat grup subnet default saat Anda membuat instans DB. Amazon Aurora membuat dan mengaitkan Antarmuka Jaringan Elastis ke instans DB Anda dengan alamat IP tersebut. instans DB menggunakan Zona Ketersediaan yang berisi subnet.

Pada langkah ini, Anda membuat grup subnet DB dan menambahkan subnet yang telah dibuat untuk VPC Anda.

Untuk membuat grup subnet DB

1. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di panel navigasi, pilih Grup subnet.
3. Pilih Buat Grup Subnet DB.
4. Untuk Nama, ketik nama grup subnet DB Anda.
5. Untuk Deskripsi, ketik deskripsi untuk grup subnet DB Anda.
6. Untuk VPC, pilih VPC default atau VPC yang Anda buat.
7. Di bagian Tambahkan subnet, pilih Zona Ketersediaan yang mencakup subnet dari Zona Ketersediaan, lalu pilih subnet dari Subnet.


RDS > Subnet groups > Create DB subnet group
Create DB Subnet Group
To create a new subnet group, give it a name and a description, and choose an existing VPC. You will then be able to add subnets related to that VPC.
Subnet group details
Name
You won't be able to modify the name after your subnet group has been created.
mydbsubnetgroup
Must contain from 1 to 255 characters. Alphanumeric characters, spaces, hyphens, underscores, and periods are allowed.
Description
My DB Subnet Group
VPC
Choose a VPC identifier that corresponds to the subnets you want to use for your DB subnet group. You won't be able to choose a different VPC identifier after your subnet group has been created.
tutorial-vpc (vpc-068fe388385afc014)
Add subnets
Availability Zones
Choose the Availability Zones that include the subnets you want to add.
Choose an availability zone
us-east-1a
us-east-1c
Subnets
Choose the subnets that you want to add. The list includes the subnets in the selected Availability Zones.
Select subnets
subnet-079bd4b8953aee1dd (10.0.0.0/24)
subnet-057e85b72c46fdd9a (10.0.1.0/24)
Subnets selected (2)
Availability zone
Subnet ID
CIDR block
us-east-1a
subnet-079bd4b8953aee1dd
10.0.0.0/24
us-east-1c
subnet-057e85b72c46fdd9a
10.0.1.0/24
Cancel
Create

8. Pilih Buat.

   Grup subnet DB baru Anda akan muncul dalam daftar grup subnet DB di konsol RDS. Anda dapat memilih grup subnet DB untuk melihat detail, termasuk semua subnet yang dikaitkan dengan grup, dalam panel detail di bagian bawah jendela.

Langkah 3: Buat grup keamanan VPC

Sebelum membuat instans DB, Anda dapat membuat grup keamanan VPC untuk dikaitkan dengan instans DB. Jika belum membuat grup keamanan VPC, Anda dapat menggunakan grup keamanan default saat membuat instans DB. Untuk petunjuk cara membuat grup keamanan untuk instans DB, lihat Membuat grup keamanan VPC untuk klaster DB privat, atau lihat Mengontrol lalu lintas ke sumber daya menggunakan grup keamanan dalam Panduan Pengguna Amazon VPC.

Langkah 4: Buat instans DB dalam VPC

Pada langkah ini, Anda membuat instans DB dan menggunakan nama VPC, grup subnet DB, dan grup keamanan VPC yang telah Anda buat pada langkah sebelumnya.

> **Note**
>
> Jika ingin instans DB dalam VPC dapat diakses publik, Anda harus mengaktifkan nama host DNS dan resolusi DNS atribut VPC. Untuk informasi selengkapnya, lihat Atribut DNS untuk VPC Anda dalam Panduan Pengguna Amazon VPC.

Untuk detail tentang cara membuat instans DB, lihat Membuat klaster DB Amazon Aurora.

Saat diminta di bagian Konektivitas, masukkan nama VPC, grup subnet DB, dan grup keamanan VPC.

> **Note**
>
> Memperbarui VPC saat ini belum didukung untuk klaster DB Aurora.

Skenario untuk mengakses klaster DB di VPC

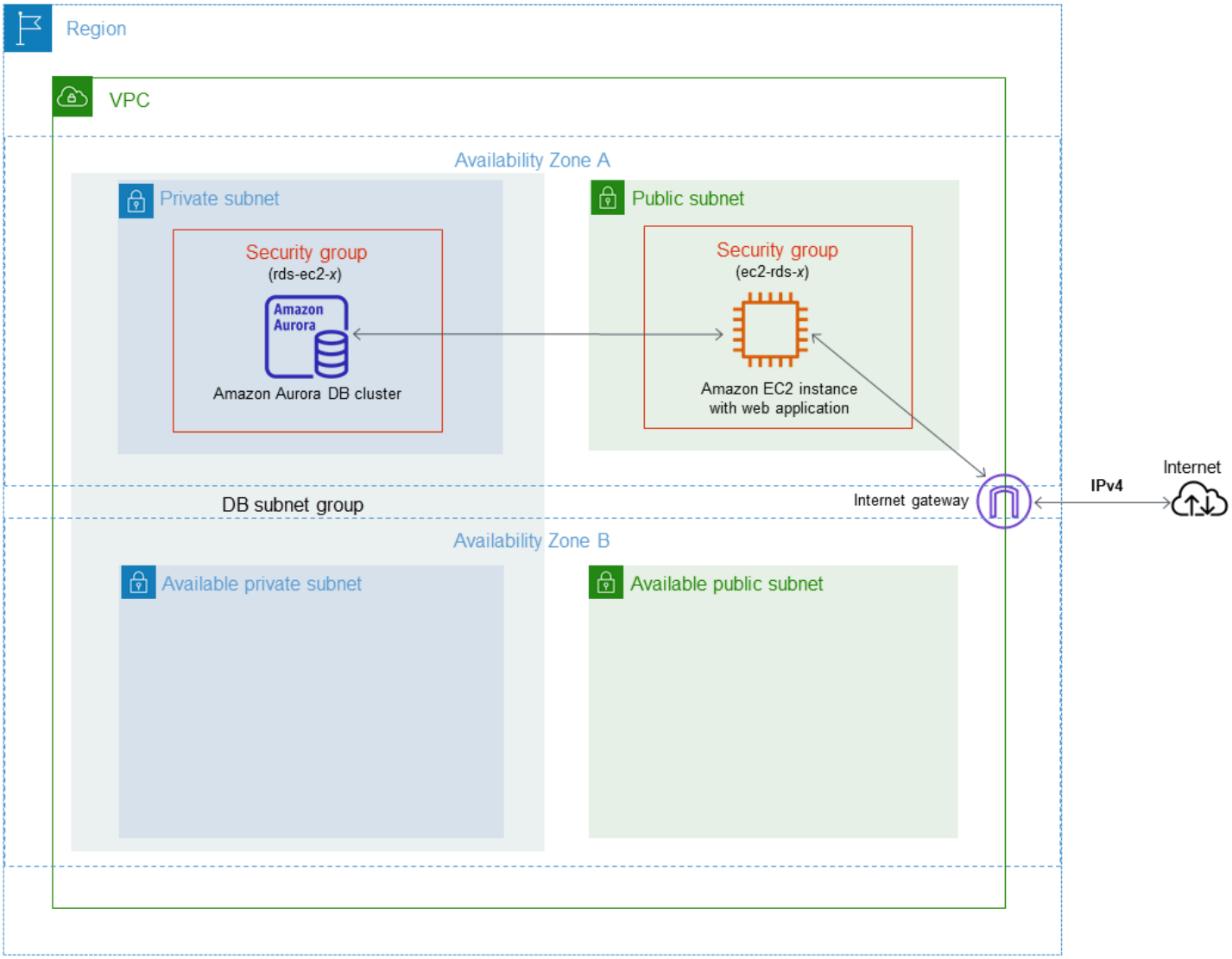
Amazon Aurora mendukung skenario berikut untuk mengakses klaster DB di VPC:

- Instans EC2 dalam VPC yang sama
- Instans EC2 di VPC yang berbeda
- Aplikasi klien melalui internet
- Jaringan pribadi

klaster DB di VPC yang diakses oleh instans EC2 dalam VPC yang sama

Penggunaan umum dari klaster DB di VPC adalah untuk berbagi data dengan server aplikasi yang dijalankan di instans EC2 dalam VPC yang sama.

Diagram berikut menunjukkan skenario ini.



Cara paling sederhana untuk mengelola akses antara instans EC2 dan klaster DB di VPC yang sama adalah dengan melakukan tindakan berikut:

- Buat grup keamanan VPC untuk memasukkan klaster DB. Grup keamanan ini dapat digunakan untuk membatasi akses ke klaster DB. Misalnya, Anda dapat membuat aturan kustom untuk grup keamanan ini. Hal ini dapat mengizinkan akses TCP menggunakan port yang Anda tetapkan ke klaster DB saat Anda membuatnya dan alamat IP yang Anda gunakan untuk mengakses klaster DB untuk pengembangan atau kepentingan lainnya.
- Buat grup keamanan VPC untuk memasukkan instans EC2 (server dan klien web) Anda. Grup keamanan ini dapat, jika diperlukan, mengizinkan akses ke instans EC2 dari internet dengan menggunakan tabel perutean VPC. Sebagai contoh, Anda dapat menetapkan aturan pada grup keamanan ini untuk mengizinkan akses TCP ke instans EC2 melalui port 22.
- Buat aturan kustom di grup keamanan untuk klaster DB Anda yang memungkinkan koneksi dari grup keamanan yang Anda buat untuk instans EC2 Anda. Aturan ini dapat mengizinkan anggota grup keamanan untuk mengakses klaster DB.

Tutorial ini mengonfigurasi subnet publik dan privat tambahan di Zona Ketersediaan terpisah. Grup subnet DB RDS membutuhkan subnet, setidaknya di dua Zona Ketersediaan. Subnet tambahan memudahkan Anda untuk beralih ke deployment instans DB Multi-AZ di masa mendatang.

Untuk tutorial yang menunjukkan cara membuat VPC dengan subnet publik maupun privat untuk skenario ini, lihat Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4).

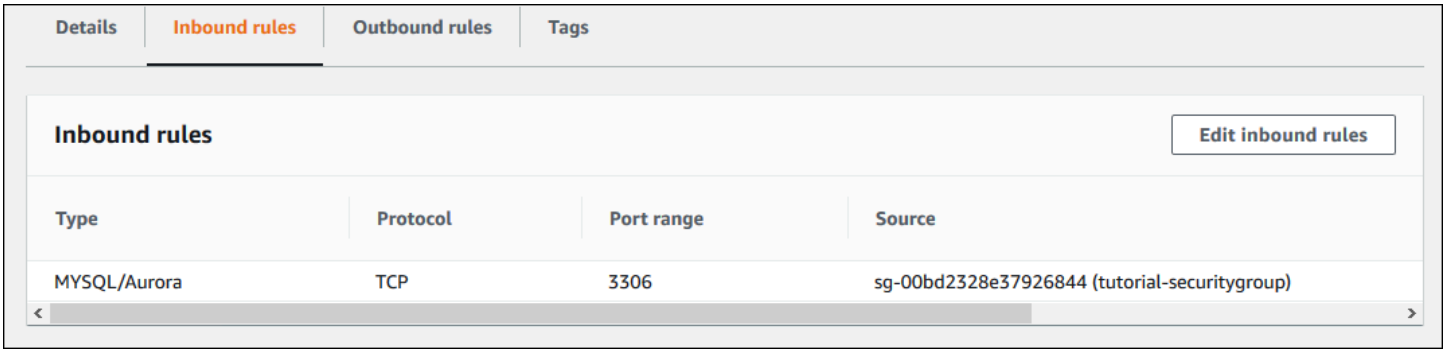
> **Tip**
>
> Anda dapat menyiapkan konektivitas jaringan antara instans Amazon EC2 dan klaster DB secara otomatis saat Anda membuat klaster DB. Untuk informasi selengkapnya, lihat Konfigurasikan konektivitas jaringan otomatis dengan instans EC2.

Untuk membuat aturan dalam grup keamanan VPC yang memungkinkan koneksi dari grup keamanan lain, ikuti langkah berikut:

1. Masuk ke AWS Management Console dan buka konsol Amazon VPC di https://console.aws.amazon.com/vpc.
2. Di panel navigasi, pilih Grup Keamanan.

3. Pilih atau buat grup keamanan yang ingin Anda berikan aksesnya ke anggota kelompok keamanan lain. Dalam skenario sebelumnya, ini adalah grup keamanan yang Anda gunakan untuk klaster DB Anda. Pilih tab Aturan masuk, lalu pilih Edit aturan masuk.
4. Di halaman Edit aturan masuk, pilih Tambahkan Aturan.
5. Untuk Jenis, pilih entri yang sesuai dengan port yang Anda gunakan saat membuat klaster DB Anda, seperti MySQL/Aurora.
6. Di kotak Sumber, mulai ketikkan ID dari grup keamanan, yang mencantumkan grup keamanan yang sesuai. Pilih grup keamanan dengan anggota yang ingin diberi akses ke sumber daya yang dilindungi oleh grup keamanan ini. Dalam skenario sebelumnya, ini adalah grup keamanan yang Anda gunakan untuk instans EC2.
7. Jika perlu, ulangi langkah-langkah untuk protokol TCP dengan membuat aturan Semua TCP sebagai Jenis dan grup keamanan Anda di kotak Sumber. Jika Anda ingin menggunakan protokol UDP, buat aturan dengan Semua UDP sebagai Jenis dan grup keamanan Anda di Sumber.
8. Pilih Simpan Aturan.

Layar berikut menunjukkan aturan masuk dengan grup keamanan sebagai sumbernya.

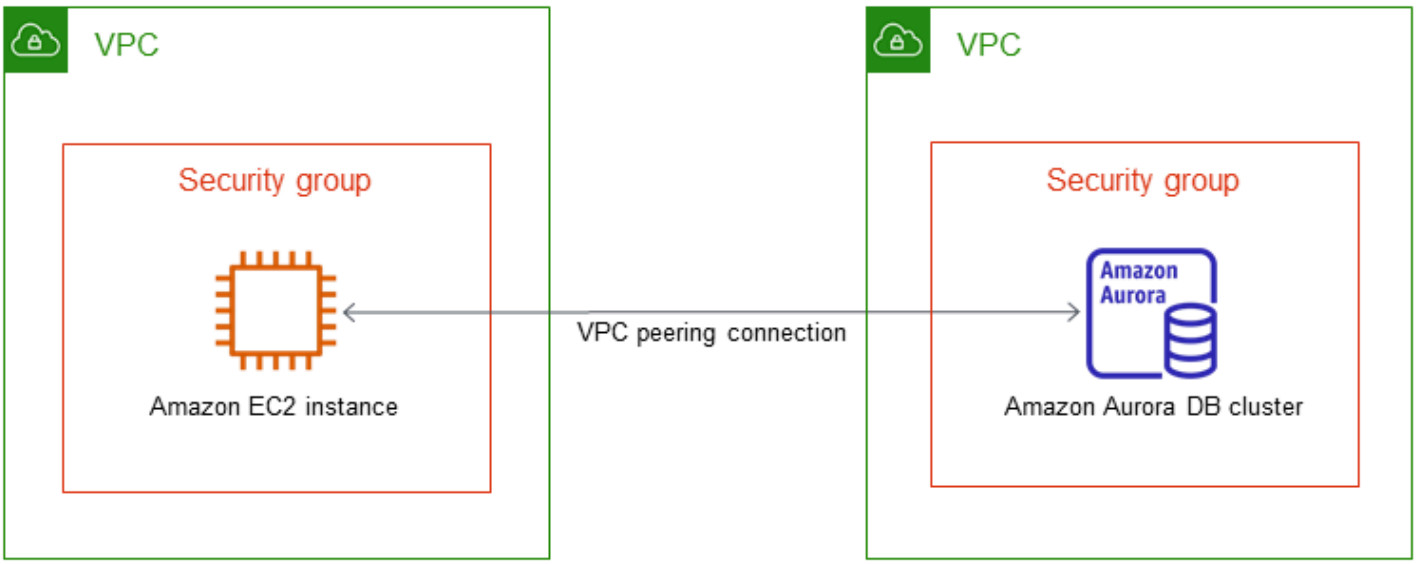


Untuk informasi selengkapnya tentang menghubungkan ke klaster DB dari instans EC2 Anda, lihat Menghubungkan ke klaster DB Amazon Aurora.

klaster DB di VPC yang diakses oleh instans EC2 di VPC yang berbeda

Jika klaster DB Anda ada dalam VPC yang berbeda dengan instans EC2 yang Anda gunakan untuk mengaksesnya, Anda dapat menggunakan peering VPC untuk mengakses klaster DB.

Diagram berikut menunjukkan skenario ini.

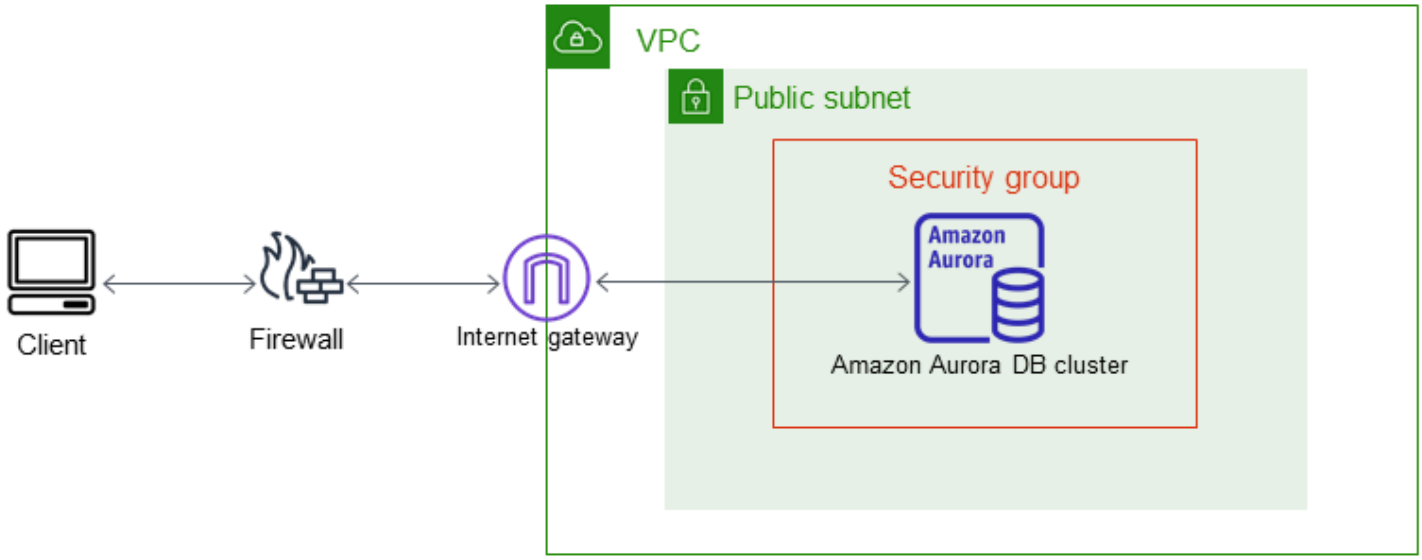


Koneksi peering VPC adalah koneksi jaringan antara dua VPC yang memungkinkan Anda merutekan lalu lintas di antara keduanya menggunakan alamat IP privat. Sumber daya di VPC yang mana pun dapat berkomunikasi satu sama lain seolah-olah mereka ada di jaringan yang sama. Anda dapat membuat koneksi peering VPC antara VPC Anda sendiri dengan VPC di akun AWS lain, atau dengan VPC di Wilayah AWS yang berbeda. Untuk mempelajari selengkapnya tentang peering VPC, lihat Peering VPC di Panduan Pengguna Amazon Virtual Private Cloud.

klaster DB di VPC yang diakses oleh aplikasi klien melalui internet

Untuk mengakses klaster DB di VPC dari aplikasi klien melalui internet, Anda dapat mengonfigurasi VPC dengan subnet publik tunggal, dan gateway internet untuk mengaktifkan komunikasi melalui internet.

Diagram berikut menunjukkan skenario ini.



Kami merekomendasikan konfigurasi berikut:

- VPC ukuran /16 (misalnya CIDR: 10.0.0.0/16). Ukuran ini menyediakan 65.536 alamat IP pribadi.
- Subnet ukuran /24 (misalnya CIDR: 10.0.0.0/24). Ukuran ini menyediakan 256 alamat IP pribadi.
- klaster DB Amazon Aurora yang dikaitkan dengan VPC dan subnet. Amazon RDS menetapkan alamat IP dalam subnet ke klaster DB Anda.
- Gateway internet yang menghubungkan VPC ke internet dan ke produk AWS lainnya.
- Grup keamanan yang terkait dengan klaster DB. Aturan masuk grup keamanan memungkinkan aplikasi klien Anda mengakses klaster DB Anda.

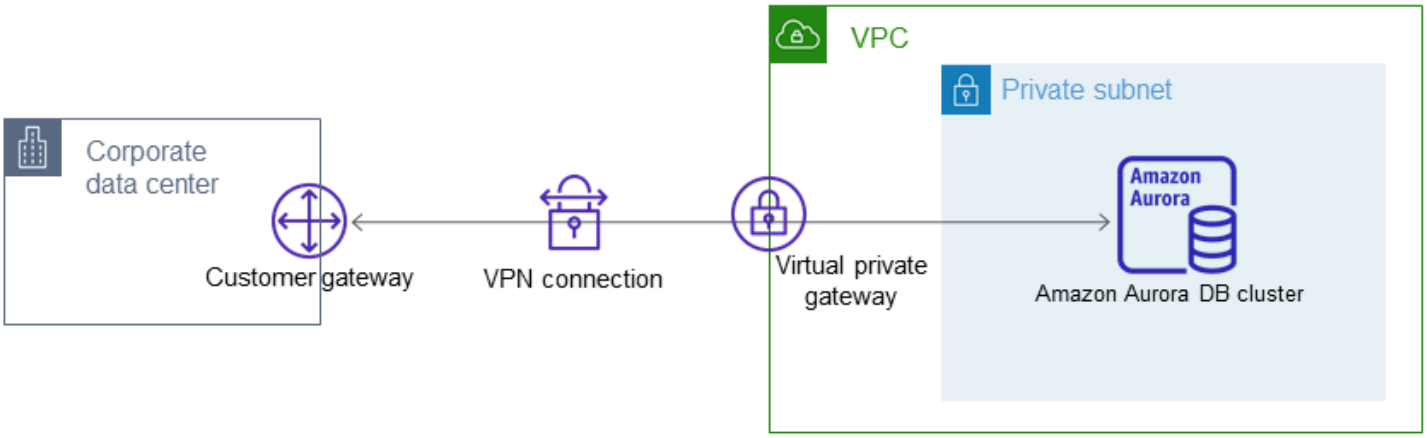
Untuk informasi tentang cara membuat klaster DB di VPC, lihat Membuat instans DB dalam VPC.

klaster DB di VPC yang diakses oleh jaringan pribadi

Jika klaster DB Anda tidak dapat diakses secara publik, Anda memiliki opsi berikut untuk mengaksesnya dari jaringan pribadi:

- Koneksi VPN Situs ke Situs AWS. Untuk informasi selengkapnya, lihat Apa itu AWS Site-to-Site VPN?
- Koneksi AWS Direct Connect. Untuk informasi selengkapnya, lihat Apa itu AWS Direct Connect?
- Koneksi AWS Client VPN. Untuk informasi selengkapnya, lihat Apa itu AWS Client VPN?

Diagram berikut menunjukkan skenario dengan koneksi VPN Situs ke Situs AWS.

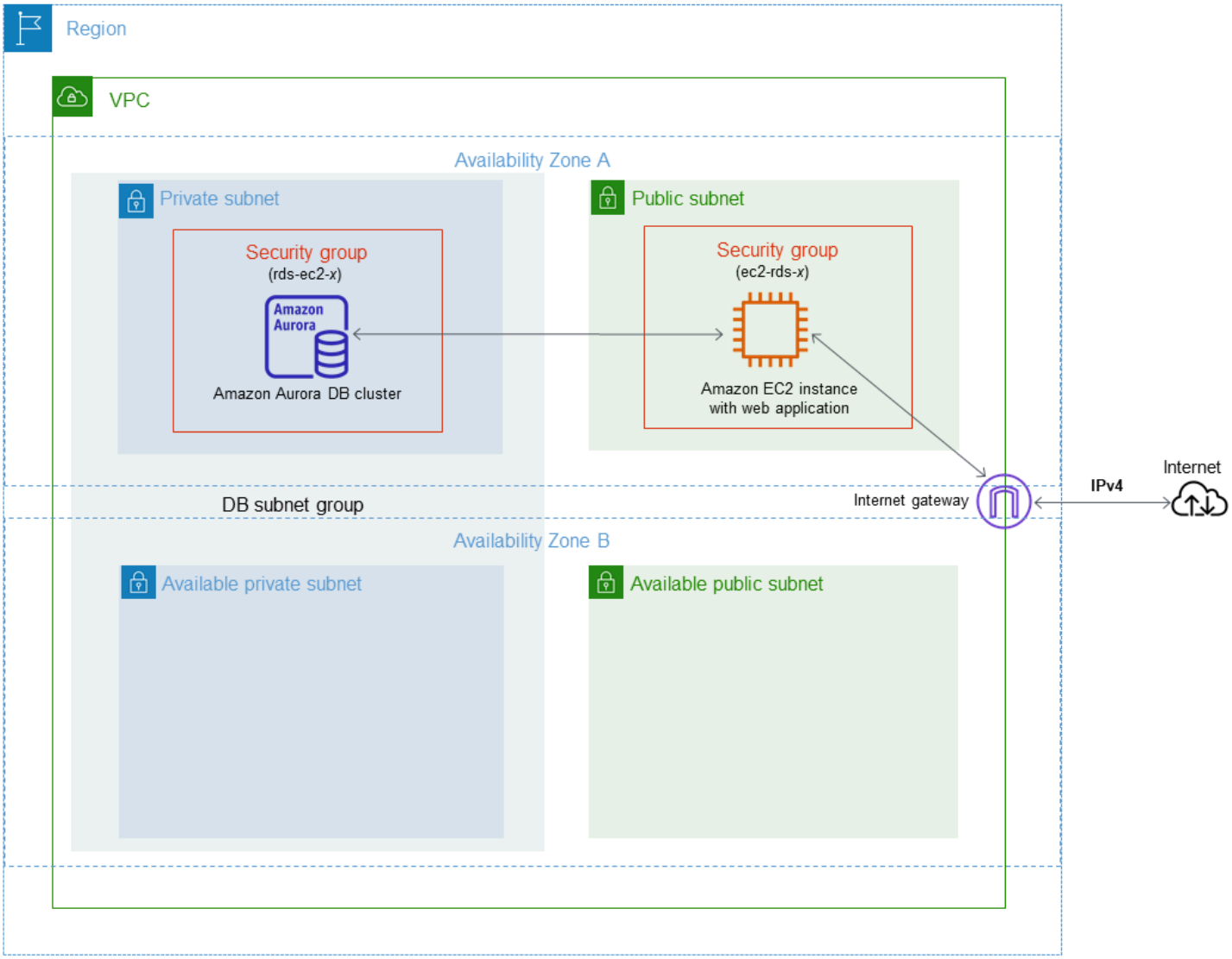


Untuk informasi selengkapnya, lihat Privasi lalu lintas antarjaringan.

Tutorial: Membuat VPC untuk digunakan dengan klaster DB (khusus IPv4)

Skenario umum mencakup klaster DB cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. VPC ini berbagi data dengan server web yang berjalan di VPC yang sama. Dalam tutorial ini, Anda akan membuat VPC untuk skenario ini.

Diagram berikut menunjukkan skenario ini. Untuk informasi tentang skenario lain, lihat Skenario untuk mengakses klaster DB di VPC.



klaster DB Anda harus tersedia hanya untuk server web Anda, dan bukan untuk internet publik. Dengan demikian, Anda akan membuat VPC dengan subnet publik dan privat. Server web di-hosting di subnet publik agar dapat menjangkau internet publik. Klaster DB di-hosting di subnet privat. Server web dapat terhubung ke klaster DB karena di-hosting dalam VPC yang sama. Namun, klaster DB tidak tersedia untuk internet publik, sehingga memberikan keamanan yang lebih baik.

Tutorial ini mengonfigurasi subnet publik dan privat tambahan di Zona Ketersediaan yang terpisah. Subnet ini tidak digunakan oleh tutorial. Grup subnet DB RDS membutuhkan subnet, setidaknya di dua Zona Ketersediaan. Subnet tambahan memudahkan untuk mengonfigurasi lebih dari satu instans DB Aurora.

Tutorial ini menjelaskan konfigurasi VPC untuk klaster DB Amazon Aurora. Untuk tutorial yang menunjukkan cara membuat server web untuk skenario VPC ini, lihat Tutorial: Membuat server web dan klaster DB Amazon Aurora. Untuk informasi selengkapnya tentang Amazon VPC, lihat Panduan Memulai Amazon VPC dan Panduan Pengguna Amazon VPC.

> **Tip**
>
> Anda dapat mengatur konektivitas jaringan antara instans Amazon EC2 dan klaster DB secara otomatis saat membuat klaster DB. Konfigurasi jaringannya mirip dengan yang dijelaskan dalam tutorial ini. Untuk informasi selengkapnya, lihat Konfigurasikan konektivitas jaringan otomatis dengan instans EC2.

Membuat VPC dengan subnet publik dan privat

Gunakan prosedur berikut untuk membuat VPC dengan subnet publik dan privat.

Membuat VPC dan subnet

1. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
2. Di pojok kanan atas AWS Management Console, pilih Wilayah untuk membuat VPC Anda. Contoh ini menggunakan Wilayah AS Barat (Oregon).
3. Di pojok kiri atas, pilih Dasbor VPC. Untuk mulai membuat VPC, pilih Buat VPC.
4. Untuk Sumber daya yang akan dibuat di bagian Pengaturan VPC, pilih VPC dan lainnya.
5. Untuk Pengaturan VPC, atur nilai-nilai ini:

   - Pembuatan otomatis tag nama – **`tutorial`**
   - Blok CIDR IPv4: – **`10.0.0.0/16`**
   - Blok CIDR IPv6 – Tidak ada blok CIDR IPv6
   - Penghunian – Default
   - Jumlah Zona Ketersediaan (AZ) – 2
   - Sesuaikan AZ – Pertahankan nilai default.

- Jumlah subnet publik – 2
- Jumlah subnet privat – 2
- Sesuaikan subnet blok CIDR – Pertahankan nilai default.
- Gateway NAT ($) – Tidak ada
- Titik akhir VPC – Tidak ada
- Opsi DNS – Pertahankan nilai default.

6. Pilih Buat VPC.

Membuat grup keamanan VPC untuk server web publik

Selanjutnya, Anda akan membuat grup keamanan untuk akses publik. Untuk terhubung ke instans EC2 publik di VPC Anda, tambahkan aturan masuk ke grup keamanan VPC Anda. Ini memungkinkan lalu lintas untuk terhubung dari internet.

Membuat grup keamanan VPC

1. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
2. Pilih Dasbor VPC, pilih Grup Keamanan, lalu pilih Buat grup keamanan.
3. Di halaman Membuat grup keamanan, atur nilai-nilai ini:
   - Nama grup keamanan: **`tutorial-securitygroup`**
   - Deskripsi: **`Tutorial Security Group`**
   - VPC: Pilih VPC yang Anda buat sebelumnya, misalnya: vpc-***`identifier`*** (tutorial-vpc)
4. Tambahkan aturan masuk ke grup keamanan.
   a. Tentukan alamat IP yang akan digunakan untuk terhubung ke instans EC2 di VPC Anda menggunakan Secure Shell (SSH). Untuk menentukan alamat IP publik Anda, Anda dapat menggunakan layanan di https://checkip.amazonaws.com di jendela atau tab browser lain. Contoh alamat IP adalah `203.0.113.25/32`.

      Dalam banyak kasus, Anda dapat menghubungkan melalui penyedia layanan Internet (ISP) atau dari belakang firewall Anda tanpa alamat IP statis. Jika demikian, temukan rentang alamat IP yang digunakan oleh komputer klien.

> **Warning**
>
> Jika menggunakan `0.0.0.0/0` untuk akses SSH, Anda memungkinkan semua alamat IP untuk mengakses instans publik Anda menggunakan SSH. Pendekatan ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, Anda hanya dapat memberikan otorisasi pada alamat IP atau rentang alamat tertentu saja untuk mengakses instans Anda menggunakan SSH.

b. Di bagian Aturan masuk, pilih Tambahkan aturan.

c. Atur nilai berikut untuk aturan masuk baru Anda yang akan mengizinkan akses SSH ke instans Amazon EC2 Anda. Tindakan ini memungkinkan Anda terhubung ke instans Amazon EC2 Anda untuk menginstal server web dan utilitas lainnya. Anda juga terhubung ke instans EC2 Anda untuk mengunggah konten untuk server web Anda.

- Jenis: **`SSH`**
- Sumber: Rentang atau alamat IP dari Langkah a, misalnya: **`203.0.113.25/32`**.

d. Pilih Tambahkan aturan.

e. Atur nilai berikut untuk aturan masuk baru yang akan mengizinkan akses HTTP ke server web Anda:

- Jenis: **`HTTP`**
- Sumber: **`0.0.0.0/0`**

5. Pilih Buat grup keamanan untuk membuat grup keamanan.

   Catat ID grup keamanan karena Anda membutuhkannya nanti dalam tutorial ini.

Membuat grup keamanan VPC untuk klaster DB privat

Agar klaster DB tetap privat, buat grup keamanan kedua untuk akses privat. Untuk terhubung ke klaster DB privat di VPC, tambahkan aturan masuk ke grup keamanan VPC Anda yang mengizinkan lalu lintas dari server web Anda saja.

Membuat grup keamanan VPC

1. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.

2. Pilih Dasbor VPC, pilih Grup Keamanan, lalu pilih Buat grup keamanan.
3. Di halaman Membuat grup keamanan, atur nilai-nilai ini:
   - Nama grup keamanan: **`tutorial-db-securitygroup`**
   - Deskripsi: **`Tutorial DB Instance Security Group`**
   - VPC: Pilih VPC yang Anda buat sebelumnya, misalnya: vpc-***`identifier`*** (tutorial-vpc)
4. Tambahkan aturan masuk ke grup keamanan.
   a. Di bagian Aturan masuk, pilih Tambahkan aturan.
   b. Atur nilai berikut untuk aturan masuk baru Anda yang akan mengizinkan lalu lintas MySQL di port 3306 dari instans Amazon EC2 Anda. Tindakan ini memungkinkan Anda terhubung dari server web Anda ke klaster DB Anda. Dengan demikian, Anda dapat menyimpan dan mengambil data dari aplikasi web Anda ke basis data Anda.
      - Jenis: **`MySQL/Aurora`**
      - Sumber: Pengidentifikasi grup keamanan tutorial-securitygroup yang Anda buat sebelumnya dalam tutorial ini, misalnya: sg-9edd5cfb.
5. Pilih Buat grup keamanan untuk membuat grup keamanan.

Membuat grup subnet DB

Grup subnet DB adalah kumpulan subnet yang Anda buat dalam VPC dan yang Anda tetapkan untuk klaster DB. Grup subnet DB memungkinkan Anda untuk menentukan VPC tertentu saat membuat klaster DB.

Membuat grup subnet DB

1. Identifikasi subnet privat untuk basis data Anda di VPC.
   a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
   b. Pilih Dasbor VPC, lalu pilih Subnet.
   c. Perhatikan ID subnet dari subnet bernama tutorial-subnet-private1-us-west-2a dan tutorial-subnet-private2-us-west-2b.

      Anda memerlukan ID subnet saat membuat grup subnet DB.
2. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

Pastikan Anda terhubung ke konsol Amazon RDS, bukan konsol Amazon VPC.

3. Di panel navigasi, pilih Grup subnet.
4. Pilih Buat grup subnet DB.
5. Di halaman Membuat grup subnet DB, atur nilai-nilai ini di Detail grup subnet:
   - Nama: **`tutorial-db-subnet-group`**
   - Deskripsi: **`Tutorial DB Subnet Group`**
   - VPC: tutorial-vpc (vpc-***`identifier`***)
6. Di bagian Tambahkan subnet, pilih Zona Ketersediaan dan Subnet.

   Untuk tutorial ini, pilih us-west-2a dan us-west-2b untuk Zona Ketersediaan. Untuk Subnet, pilih subnet privat yang Anda identifikasi pada langkah sebelumnya.
7. Pilih Buat.

   Grup subnet DB baru Anda akan muncul dalam daftar grup subnet DB di konsol RDS. Anda dapat memilih grup subnet DB untuk melihat detail di panel detail di bagian bawah jendela. Detail ini mencakup semua subnet yang dikaitkan dengan grup tersebut.

> **Note**
>
> Jika Anda membuat VPC ini untuk menyelesaikan Tutorial: Membuat server web dan klaster DB Amazon Aurora, buat klaster DB dengan mengikuti petunjuk di Membuat klaster DB Amazon Aurora.

Menghapus VPC

Setelah membuat VPC dan sumber daya lainnya untuk tutorial ini, Anda dapat menghapusnya jika tidak membutuhkannya lagi.

> **Note**
>
> Jika Anda menambahkan sumber daya di VPC yang Anda buat untuk tutorial ini, Anda mungkin perlu menghapus sumber daya tersebut sebelum menghapus VPC. Misalnya,

sumber daya ini mungkin menyertakan instans Amazon EC2 atau klasterDB Amazon RDS. Untuk informasi selengkapnya, lihat Menghapus VPC di Panduan Pengguna Amazon VPC.

Menghapus VPC dan sumber daya terkait

1. Hapus grup subnet DB.
    a. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
    b. Di panel navigasi, pilih Grup subnet.
    c. Pilih grup subnet DB yang ingin Anda hapus, seperti tutorial-db-subnet-group.
    d. Pilih Hapus, lalu pilih Hapus di jendela konfirmasi.
2. Catat ID VPC.
    a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
    b. Pilih Dasbor VPC, lalu pilih VPC.
    c. Dalam daftar, identifikasi VPC yang Anda buat, seperti tutorial-vpc.
    d. Catat ID VPC dari VPC yang Anda buat. Anda memerlukan ID VPC di langkah berikutnya.
3. Hapus grup keamanan.
    a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
    b. Pilih Dasbor VPC, lalu pilih Grup Keamanan.
    c. Pilih grup keamanan untuk instans DB Amazon RDS, seperti tutorial-db-securitygroup.
    d. Untuk Tindakan, pilih Hapus grup keamanan, lalu pilih Hapus di halaman konfirmasi.
    e. Di halaman Grup Keamanan, pilih grup keamanan untuk instans Amazon EC2, seperti tutorial-securitygroup.
    f. Untuk Tindakan, pilih Hapus grup keamanan, lalu pilih Hapus di halaman konfirmasi.
4. Hapus VPC.
    a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
    b. Pilih Dasbor VPC, lalu pilih VPC.
    c. Pilih VPC yang ingin Anda hapus, seperti tutorial-vpc.
    d. Untuk Tindakan, pilih Hapus VPC.

Halaman konfirmasi menunjukkan sumber daya lain yang dikaitkan dengan VPC yang juga akan dihapus, termasuk subnet terkait.

e. Di halaman konfirmasi, masukkan **`delete`** lalu pilih Hapus.

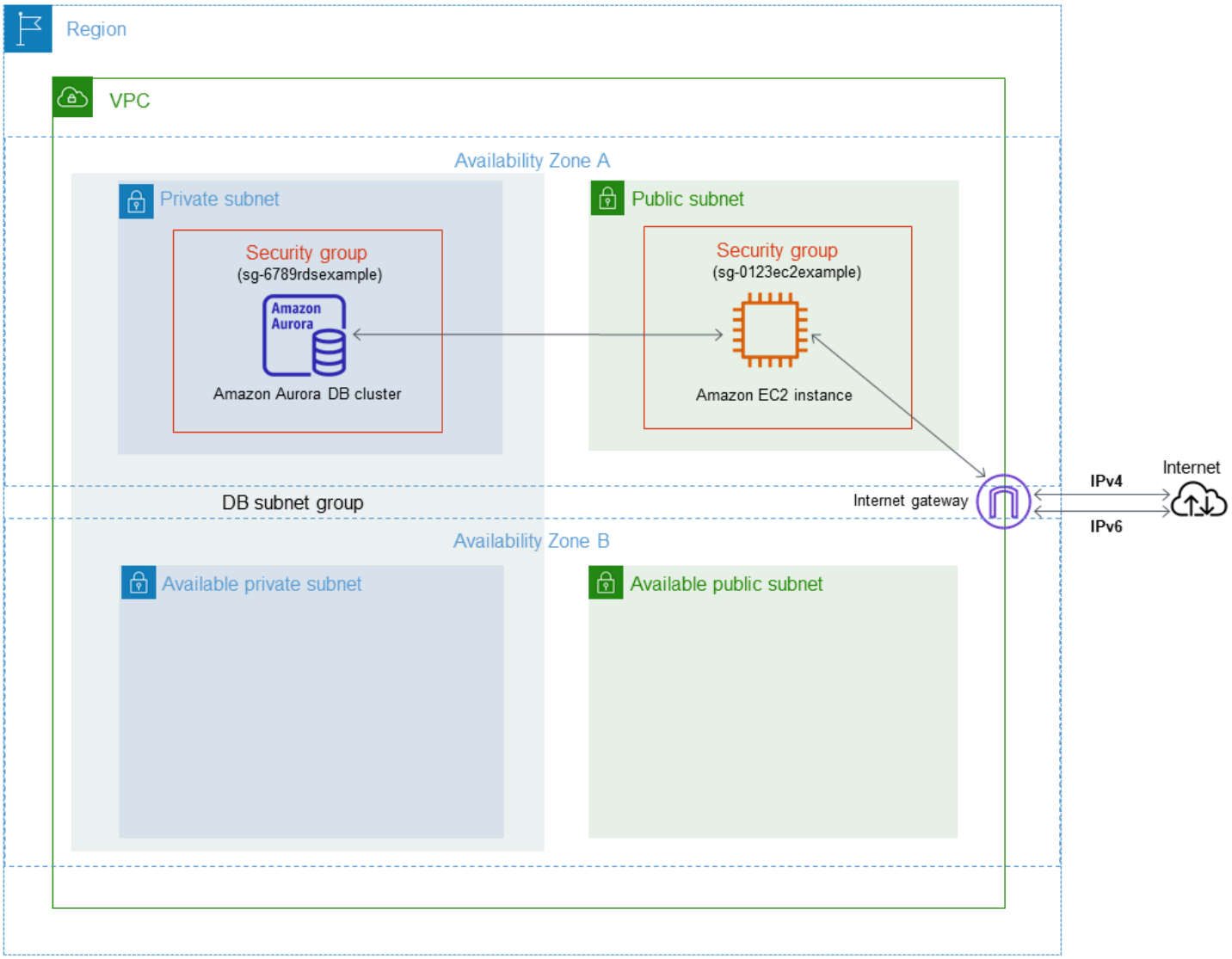
Tutorial: Membuat VPC untuk digunakan dengan klaster DB (mode tumpukan ganda)

Skenario umum mencakup klaster DB cloud privat virtual (VPC) berdasarkan layanan Amazon VPC. VPC ini membagikan data dengan instans Amazon EC2 publik yang dijalankan di VPC yang sama.

Dalam tutorial ini, Anda akan membuat VPC untuk skenario ini yang berfungsi dengan basis data yang berjalan dalam mode tumpukan ganda. Mode tumpukan ganda untuk mengaktifkan koneksi melalui protokol pengalamatan IPv6. Untuk informasi selengkapnya tentang alamat IP, lihat Penentuan alamat IP Amazon Aurora.

Klaster jaringan tumpukan ganda didukung di sebagian besar wilayah. Untuk informasi selengkapnya, lihat Ketersediaan instans DB jaringan dual-stack. Untuk melihat batasan mode tumpukan ganda, lihat Batasan untuk instans DB jaringan dual-stack.

Diagram berikut menunjukkan skenario ini.



Untuk informasi tentang skenario lain, lihat Skenario untuk mengakses klaster DB di VPC.

Klaster DB Anda harus tersedia hanya untuk server web Anda, dan bukan untuk internet publik. Dengan demikian, Anda akan membuat VPC dengan subnet publik dan privat. Instans Amazon EC2 di-hosting di subnet publik agar dapat menjangkau internet publik. Klaster DB di-hosting di subnet privat. Instans Amazon EC2 dapat terhubung ke klaster DB karena di-hosting dalam VPC yang sama. Namun, klaster DB tidak tersedia untuk internet publik, sehingga memberikan keamanan yang lebih besar.

Tutorial ini mengonfigurasi subnet publik dan privat tambahan di Zona Ketersediaan yang terpisah. Subnet ini tidak digunakan oleh tutorial. Grup subnet DB RDS membutuhkan subnet, setidaknya di dua Zona Ketersediaan. Subnet tambahan memudahkan untuk mengonfigurasi lebih dari satu instans DB Aurora.

Untuk membuat klaster DB yang menggunakan mode tumpukan ganda, tetapkan Mode tumpukan ganda untuk pengaturan Jenis jaringan. Anda juga dapat mengubah klaster DB dengan pengaturan yang sama. Untuk informasi selengkapnya tentang cara membuat klaster DB, lihat Membuat klaster DB Amazon Aurora. Untuk informasi selengkapnya tentang cara memodifikasi klaster DB, lihat Memodifikasi klaster DB Amazon Aurora.

Tutorial ini menjelaskan konfigurasi VPC untuk klaster DB Amazon Aurora. Untuk informasi selengkapnya tentang Amazon VPC, lihat Panduan Pengguna Amazon VPC.

Membuat VPC dengan subnet publik dan privat

Gunakan prosedur berikut untuk membuat VPC dengan subnet publik dan privat.

Membuat VPC dan subnet

1. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
2. Di pojok kanan atas AWS Management Console, pilih Wilayah untuk membuat VPC Anda. Contoh ini menggunakan Wilayah AS Timur (Ohio).
3. Di pojok kiri atas, pilih Dasbor VPC. Untuk mulai membuat VPC, pilih Buat VPC.
4. Untuk Sumber daya yang akan dibuat di bagian Pengaturan VPC, pilih VPC dan lainnya.
5. Untuk Pengaturan VPC lainnya, atur nilai-nilai ini:

   - Pembuatan otomatis tag nama – **`tutorial-dual-stack`**
   - Blok CIDR IPv4: – **`10.0.0.0/16`**
   - Blok CIDR IPv6 – Blok CIDR IPv6 yang disediakan Amazon
   - Penghunian – Default
   - Jumlah Zona Ketersediaan (AZ) – 2
   - Sesuaikan AZ – Pertahankan nilai default.
   - Jumlah subnet publik – 2
   - Jumlah subnet privat – 2
   - Sesuaikan subnet blok CIDR – Pertahankan nilai default.
   - Gateway NAT ($) – Tidak ada
   - Gateway internet khusus egress – Tidak
   - Titik akhir VPC – Tidak ada
   - Opsi DNS – Pertahankan nilai default.

> **Note**
>
> Amazon RDS membutuhkan setidaknya dua subnet di dua Zona Ketersediaan yang berbeda untuk mendukung deployment instans DB Multi-AZ. Tutorial ini membuat deployment Single-AZ, tetapi persyaratannya memudahkan konversi ke deployment instans DB Multi-AZ di masa mendatang.

6. Pilih Buat VPC.

Membuat grup keamanan VPC untuk instans Amazon EC2 publik

Selanjutnya, Anda akan membuat grup keamanan untuk akses publik. Untuk terhubung ke instans EC2 publik di VPC Anda, tambahkan aturan masuk ke grup keamanan VPC Anda yang mengizinkan lalu lintas untuk terhubung dari internet.

Membuat grup keamanan VPC

1. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
2. Pilih Dasbor VPC, pilih Grup Keamanan, lalu pilih Buat grup keamanan.
3. Di halaman Membuat grup keamanan, atur nilai-nilai ini:
   - Nama grup keamanan: **`tutorial-dual-stack-securitygroup`**
   - Deskripsi: **`Tutorial Dual-Stack Security Group`**
   - VPC: Pilih VPC yang Anda buat sebelumnya, misalnya: vpc-***`identifier`*** (tutorial–dual-stack-vpc)
4. Tambahkan aturan masuk ke grup keamanan.
   a. Tentukan alamat IP yang akan digunakan untuk terhubung ke instans EC2 di VPC Anda menggunakan Secure Shell (SSH).

      Contoh rentang alamat Protokol Internet versi 4 (IPv4) adalah `203.0.113.25/32`. Contoh rentang alamat Protokol Internet versi 6 (IPv6) adalah `2001:db8:1234:1a00::/64`.

      Dalam banyak kasus, Anda dapat menghubungkan melalui penyedia layanan Internet (ISP) atau dari belakang firewall Anda tanpa alamat IP statis. Jika demikian, temukan rentang alamat IP yang digunakan oleh komputer klien.

> **Warning**
>
> Jika Anda menggunakan `0.0.0.0/0` untuk IPv4 atau `::0` untuk IPv4, Anda memungkinkan semua alamat IP untuk mengakses instans publik Anda menggunakan SSH. Pendekatan ini dapat diterima untuk waktu yang singkat di lingkungan pengujian, tetapi tidak aman untuk lingkungan produksi. Dalam produksi, Anda hanya dapat memberikan otorisasi pada alamat IP atau rentang alamat tertentu saja untuk mengakses instans-instans Anda.

b. Di bagian Aturan masuk, pilih Tambahkan aturan.

c. Atur nilai berikut untuk aturan masuk baru Anda yang akan mengizinkan akses Secure Shell (SSH) ke instans Amazon EC2 Anda. Jika Anda melakukan ini, Anda dapat terhubung ke instans EC2 Anda untuk menginstal klien SQL dan aplikasi lainnya. Tentukan alamat IP agar Anda dapat mengakses instans EC2 Anda:

- Jenis: **`SSH`**
- Sumber: Rentang atau alamat IP dari langkah a. Contoh alamat IP IPv4 adalah **`203.0.113.25/32`**. Contoh alamat IP IPv6 adalah **`2001:DB8::/32`**.

5. Pilih Buat grup keamanan untuk membuat grup keamanan.

   Catat ID grup keamanan karena Anda membutuhkannya nanti dalam tutorial ini.

Membuat grup keamanan VPC untuk klaster DB privat

Agar klaster DB tetap privat, buat grup keamanan kedua untuk akses privat. Untuk terhubung ke klaster DB di VPC Anda, tambahkan aturan masuk ke grup keamanan VPC Anda. Hal ini mengizinkan lalu lintas dari instans Amazon EC2 Anda saja.

Membuat grup keamanan VPC

1. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
2. Pilih Dasbor VPC, pilih Grup Keamanan, lalu pilih Buat grup keamanan.
3. Di halaman Membuat grup keamanan, atur nilai-nilai ini:

- Nama grup keamanan: **`tutorial-dual-stack-db-securitygroup`**
- Deskripsi: **`Tutorial Dual-Stack DB Instance Security Group`**

- VPC: Pilih VPC yang Anda buat sebelumnya, misalnya: vpc-***identifier*** (tutorial–dual-stack-vpc)

4. Tambahkan aturan masuk ke grup keamanan:

   a. Di bagian Aturan masuk, pilih Tambahkan aturan.

   b. Atur nilai berikut untuk aturan masuk baru Anda yang akan mengizinkan lalu lintas MySQL di port 3306 dari instans Amazon EC2 Anda. Dengan melakukannya, Anda dapat terhubung dari instans EC2 Anda ke klaster DB Anda. Melakukan hal ini berarti Anda dapat mengirim data dari instans EC2 Anda ke basis data Anda.

      - Jenis: MySQL/Aurora
      - Sumber: Pengidentifikasi grup keamanan tutorial-dual-stack-securitygroup yang Anda buat sebelumnya dalam tutorial ini, misalnya sg-9edd5cfb.

5. Untuk membuat grup keamanan, pilih Buat grup keamanan.

Membuat grup subnet DB

Grup subnet DB adalah kumpulan subnet yang Anda buat dalam VPC dan yang Anda tetapkan untuk klaster DB. Dengan menggunakan grup subnet DB, Anda dapat menentukan VPC tertentu saat membuat klaster DB. Untuk membuat grup subnet DB yang kompatibel dengan `DUAL`, semua subnet harus kompatibel dengan `DUAL`. Agar kompatibel dengan `DUAL`, subnet harus memiliki CIDR IPv6 yang dikaitkan dengan subnet tersebut.

Membuat grup subnet DB

1. Identifikasi subnet privat untuk basis data Anda di VPC.

   a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.

   b. Pilih Dasbor VPC, lalu pilih Subnet.

   c. Perhatikan ID subnet dari subnet bernama tutorial-dual-stack-subnet-private1-us-west-2a dan tutorial-dual-stack-subnet-private2-us-west-2b.

      Anda memerlukan ID subnet saat membuat grup subnet DB.

2. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.

   Pastikan Anda terhubung ke konsol Amazon RDS, bukan konsol Amazon VPC.

3. Di panel navigasi, pilih Grup subnet.

4. Pilih Buat grup subnet DB.
5. Di halaman Membuat grup subnet DB, atur nilai-nilai ini di Detail grup subnet:
   - Nama: **`tutorial-dual-stack-db-subnet-group`**
   - Deskripsi: **`Tutorial Dual-Stack DB Subnet Group`**
   - VPC: tutorial-dual-stack-vpc (vpc-***`identifier`***)
6. Di bagian Tambahkan subnet, pilih nilai untuk opsi Zona Ketersediaan dan Subnet.

   Untuk tutorial ini, pilih us-east-2a dan us-east-2b untuk Zona Ketersediaan. Untuk Subnet, pilih subnet privat yang Anda identifikasi pada langkah sebelumnya.
7. Pilih Buat.

Grup subnet DB baru Anda akan muncul dalam daftar grup subnet DB di konsol RDS. Anda dapat memilih grup subnet DB untuk melihat detailnya. Detail ini termasuk protokol pengalamatan yang didukung, semua subnet yang terkait dengan grup tersebut, dan jenis jaringan yang didukung oleh grup subnet DB.

Membuat instans Amazon EC2 dalam mode tumpukan ganda

Untuk membuat instans Amazon EC2, ikuti petunjuk dalam Meluncurkan instans menggunakan wizard peluncuran instans baru di Panduan Pengguna Amazon EC2 untuk Instans Linux.

Di halaman Mengonfigurasi Detail Instans, atur nilai-nilai ini dan biarkan nilai lainnya sebagai default:

- Jaringan – Pilih VPC yang ada dengan subnet publik dan privat seperti tutorial-dual-stack-vpc (vpc-*`identifier`*) yang dibuat di Membuat VPC dengan subnet publik dan privat.
- Subnet – Pilih subnet publik yang sudah ada, seperti subnet-***`identifier`*** | tutorial-dual-stack-subnet-public1-us-east-2a | us-east-2a yang dibuat di Membuat grup keamanan VPC untuk instans Amazon EC2 publik.
- IP Publik yang Otomatis Ditetapkan – Pilih Aktifkan.
- IP IPv6 yang Otomatis Ditetapkan – Pilih Aktifkan.
- Firewall (grup keamanan) – Pilih Pilih grup keamanan yang ada.
- Grup keamanan umum – Pilih grup keamanan yang ada, seperti `tutorial-securitygroup` yang dibuat di Membuat grup keamanan VPC untuk instans Amazon EC2 publik. Pastikan grup keamanan yang Anda pilih menyertakan aturan masuk untuk akses Secure Shell (SSH) dan HTTP.

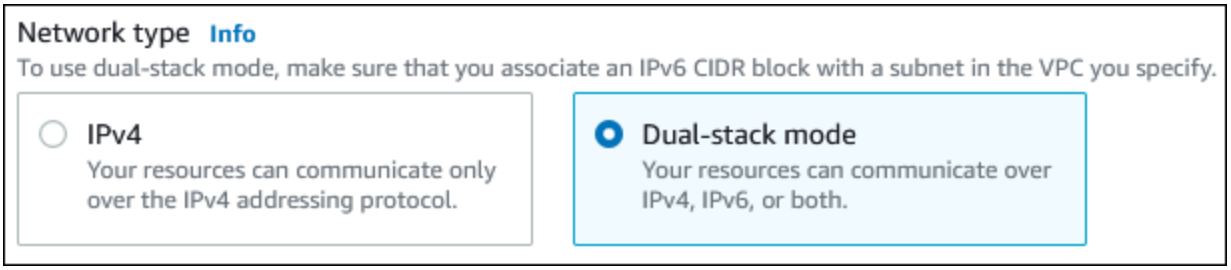
Membuat klaster DB dalam mode tumpukan ganda

Pada langkah ini, Anda akan membuat klaster DB yang berjalan dalam mode tumpukan ganda.

Membuat instans DB

1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
2. Di pojok kanan atas konsol, pilih Wilayah AWS tempat Anda akan membuat klaster DB. Contoh ini menggunakan Wilayah AS Timur (Ohio).
3. Di panel navigasi, pilih Basis data.
4. Pilih Buat basis data.
5. Di halaman Membuat basis data, pastikan bahwa opsi Pembuatan standar dipilih, lalu pilih jenis mesin Aurora MySQL DB.
6. Di bagian Konektivitas, atur nilai-nilai ini:

   - Jenis jaringan – Pilih Mode tumpukan ganda.

   - Cloud privat virtual (VPC) – Pilih VPC yang ada dengan subnet publik dan privat seperti tutorial-dual-stack-vpc (vpc-*identifier*) yang dibuat di Membuat VPC dengan subnet publik dan privat.

     VPC tersebut harus memiliki subnet di Zona Ketersediaan yang berbeda.

   - Grup subnet DB – Pilih grup subnet DB untuk VPC, seperti tutorial-dual-stack-db-subnet-group yang dibuat di Membuat grup subnet DB.
   - Akses publik – Pilih Tidak.
   - Grup keamanan VPC (firewall) – Pilih Pilih grup keamanan yang ada.
   - Grup keamanan VPC yang ada – Pilih grup keamanan VPC yang sudah ada dan dikonfigurasi untuk akses privat, seperti tutorial-dual-stack-db-securitygroup yang dibuat di Membuat grup keamanan VPC untuk klaster DB privat.

Hapus grup keamanan lainnya, seperti grup keamanan default, dengan memilih X yang dikaitkan dengan masing-masing grup keamanan.

- Zona Ketersediaan – Pilih us-west-2a.

  Untuk menghindari lalu lintas AZ, pastikan instans DB dan instans EC2 berada di Zona Ketersediaan yang sama.

7. Untuk bagian yang tersisa, tentukan pengaturan klaster DB Anda. Untuk informasi tentang setiap pengaturan, lihat Pengaturan untuk klaster Aurora DB.

Menghubungkan ke instans Amazon EC2 dan klaster DB

Setelah membuat instans Amazon EC2 dan klaster DB dalam mode tumpukan ganda, Anda dapat terhubung ke masing-masing instans menggunakan protokol IPv6. Untuk menghubungkan ke instans Amazon EC2 menggunakan protokol IPv6, ikuti petunjuk di Menghubungkan ke instans Linux di Panduan Pengguna Amazon EC2 untuk Instans Linux.

Untuk menghubungkan ke klaster DB Aurora MySQL dari instans Amazon EC2, ikuti petunjuk di Menghubungkan ke klaster DB Aurora MySQL.

Menghapus VPC

Setelah membuat VPC dan sumber daya lainnya untuk tutorial ini, Anda dapat menghapusnya jika tidak membutuhkannya lagi.

Jika Anda menambahkan sumber daya di VPC yang Anda buat untuk tutorial ini, Anda mungkin perlu menghapus sumber daya tersebut sebelum menghapus VPC. Contoh sumber dayanya adalah instans Amazon EC2 atau klaster DB. Untuk informasi selengkapnya, lihat Menghapus VPC di Panduan Pengguna Amazon VPC.

Menghapus VPC dan sumber daya terkait

1. Hapus grup subnet DB:

   a. Buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
   b. Di panel navigasi, pilih Grup subnet.
   c. Pilih grup subnet DB yang akan dihapus, seperti tutorial-db-subnet-group.
   d. Pilih Hapus, lalu pilih Hapus di jendela konfirmasi.

2. Catat ID VPC:

a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
b. Pilih Dasbor VPC, lalu pilih VPC.
c. Dalam daftar, identifikasi VPC yang Anda buat, seperti tutorial-dual-stack-vpc.
d. Catat nilai ID VPC dari VPC yang Anda buat. Anda memerlukan ID VPC ini di langkah selanjutnya.

3. Hapus grup keamanan:

a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
b. Pilih Dasbor VPC, lalu pilih Grup Keamanan.
c. Pilih grup keamanan untuk instans DB Amazon RDS, seperti tutorial-dual-stack-db-securitygroup.
d. Untuk Tindakan, pilih Hapus grup keamanan, lalu pilih Hapus di halaman konfirmasi.
e. Di halaman Grup Keamanan, pilih grup keamanan untuk instans Amazon EC2, seperti tutorial-dual-stack-securitygroup.
f. Untuk Tindakan, pilih Hapus grup keamanan, lalu pilih Hapus di halaman konfirmasi.

4. Hapus gateway NAT:

a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
b. Pilih Dasbor VPC, lalu pilih Gateway NAT.
c. Pilih gateway NAT dari VPC yang Anda buat. Gunakan ID VPC untuk mengidentifikasi gateway NAT yang benar.
d. Untuk Tindakan, pilih Hapus gateway NAT.
e. Di halaman konfirmasi, masukkan **`delete`** lalu pilih Hapus.

5. Hapus VPC:

a. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
b. Pilih Dasbor VPC, lalu pilih VPC.
c. Pilih VPC yang ingin Anda hapus, seperti tutorial-dual-stack-vpc.
d. Untuk Tindakan, pilih Hapus VPC.

   Halaman konfirmasi menunjukkan sumber daya lain yang dikaitkan dengan VPC yang juga akan dihapus, termasuk subnet terkait.

e. Di halaman konfirmasi, masukkan **`delete`** lalu pilih Hapus.

6. Rilis alamat IP Elastis:

    a. Buka konsol Amazon EC2 di https://console.aws.amazon.com/ec2/.
    b. Pilih Dasbor EC2, lalu pilih IP Elastis.
    c. Pilih alamat IP Elastis yang ingin Anda rilis.
    d. Untuk Tindakan, pilih Rilis alamat IP Elastis.
    e. Di halaman konfirmasi, pilih Rilis.

Kuota dan batasan untuk Amazon Aurora

Setelah itu, Anda dapat menemukan deskripsi kuota sumber daya dan batasan penamaan untuk Amazon Aurora.

Topik

- Kuota dalam Amazon Aurora
- Batasan penamaan dalam Amazon Aurora
- Batas ukuran Amazon Aurora

Kuota dalam Amazon Aurora

Setiap AWS akun memiliki kuota, untuk setiap AWS Wilayah, pada jumlah sumber daya Amazon Aurora yang dapat dibuat. Setelah kuota sumber daya tercapai, panggilan tambahan untuk membuat sumber daya tersebut akan gagal dengan pengecualian.

Tabel berikut mencantumkan sumber daya dan kuota mereka per AWS Wilayah.

Nama	Default	Dapat disesuaikan	Deskripsi
Otorisasi per grup keamanan DB	Setiap Wilayah yang didukung: 20	Tidak	Jumlah otorisasi grup keamanan per grup keamanan DB
Versi mesin kustom	Setiap Wilayah yang didukung: 40	Ya	Jumlah maksimum versi mesin kustom yang diizinkan di akun di Wilayah saat ini
Grup parameter klaster DB	Setiap Wilayah yang didukung: 50	Tidak	Jumlah maksimum grup parameter klaster DB
Klaster DB	Setiap Wilayah yang didukung: 40	Ya	Jumlah maksimum klaster Aurora yang diizinkan di

Nama	Default	Dapat disesu an	Deskripsi
			akun ini dalam Wilayah saat ini
Instans DB	Setiap Wilayah yang didukung: 40	Ya	Jumlah maksimum instans DB yang diizinkan di akun ini dalam Wilayah saat ini
Grup subnet DB	Setiap Wilayah yang didukung: 50	Ya	Jumlah maksimum grup subnet DB
Ukuran konten permintaan HTTP API Data	Setiap Wilayah yang didukung: 4 Megabyte	Tidak	Ukuran maksimum yang diizinkan untuk konten permintaan HTTP.
Pasangan rahasia klaster serentak maksimum API Data	Setiap Wilayah yang didukung: 30	Tidak	Jumlah maksimum pasangan unik cluster dan rahasia DB Tanpa Server Aurora dalam permintaan API Data bersamaan untuk akun dan Wilayah saat ini. AWS

Nama	Default	Dapat disesu an	Deskripsi
Permintaan serentak maksimum API Data	Setiap Wilayah yang didukung: 500	Tidak	Jumlah maksimum permintaan API Data ke klaster DB Aurora Nirserver yang menggunakan rahasia yang sama dan dapat diproses pada saat yang sama. Permintaan tambahan dimasukkan ke dalam antrean dan diproses setelah permintaan dalam proses selesai.
Ukuran set hasil maksimum API Data	Setiap Wilayah yang didukung: 1 Megabyte	Tidak	Ukuran maksimum set hasil basis data yang dapat dikembalikan oleh API Data.
Ukuran maksimum API Data string respons JSON	Setiap Wilayah yang didukung: 10 Megabyte	Tidak	Ukuran maksimum string respons JSON yang disederhanakan dikembalikan oleh API Data RDS.
Permintaan API Data per detik	Setiap Wilayah yang didukung: 1.000 per detik	Tidak	Jumlah maksimum permintaan ke Data API per detik yang diizinkan di akun ini di AWS Wilayah saat ini.
Langganan peristiwa	Setiap Wilayah yang didukung: 20	Ya	Jumlah maksimum langganan peristiwa

Nama	Default	Dapat disesuaikan	Deskripsi
Peran IAM per klaster DB	Setiap Wilayah yang didukung: 5	Ya	Jumlah maksimum peran IAM yang terkait dengan klaster DB
Peran IAM per instans DB	Setiap Wilayah yang didukung: 5	Ya	Jumlah maksimum peran IAM yang terkait dengan instans DB
Snapshot klaster DB manual	Setiap Wilayah yang didukung: 100	Ya	Jumlah maksimum snapshot klaster DB manual
Snapshot instans DB manual	Setiap Wilayah yang didukung: 100	Ya	Jumlah maksimum snapshot instans DB manual
Grup opsi	Setiap Wilayah yang didukung: 20	Ya	Jumlah maksimum grup opsi
Grup parameter	Setiap Wilayah yang didukung: 50	Ya	Jumlah maksimum grup parameter
Proksi	Setiap Wilayah yang didukung: 20	Ya	Jumlah maksimum proxy yang diizinkan di akun ini di Wilayah saat ini AWS
Replika baca per primer	Setiap Wilayah yang didukung: 15	Ya	Jumlah maksimum replika baca per instans DB primer. Kuota ini tidak dapat disesuaikan untuk Amazon Aurora.

Nama	Default	Dapat disesu an	Deskripsi
Instans DB cadangan	Setiap Wilayah yang didukung: 40	Ya	Jumlah maksimum instans DB cadangan yang diizinkan di akun ini di Wilayah saat ini AWS
Aturan per grup keamanan	Setiap Wilayah yang didukung: 20	Tidak	Jumlah maksimum aturan per grup keamanan DB
Grup keamanan	Setiap Wilayah yang didukung: 25	Ya	Jumlah maksimum aturan grup keamanan DB
Grup keamanan (VPC)	Setiap Wilayah yang didukung: 5	Tidak	Jumlah maksimum grup keamanan DB per Amazon VPC
Subnet per grup subnet DB	Setiap Wilayah yang didukung: 20	Tidak	Jumlah maksimum subnet per grup subnet DB
Tanda per sumber daya	Setiap Wilayah yang didukung: 50	Tidak	Jumlah maksimum tanda per sumber daya Amazon RDS

Nama	Default	Dapat disesuaikan	Deskripsi
Total penyimpanan untuk semua instans DB	Setiap Wilayah yang didukung: 100.000 Gigabyte	Ya	Total penyimpanan maksimum (dalam GB) pada volume EBS untuk semua instans DB Amazon RDS yang ditambahkan bersama. Kuota ini tidak berlaku untuk Amazon Aurora, yang memiliki volume cluster maksimum 128 TiB untuk setiap cluster DB.

> **Note**
>
> Secara default, Anda dapat memiliki hingga total 40 instans DB. Instans DB RDS, instans DB Aurora, instans Amazon Neptune, dan instans Amazon DocumentDB berlaku untuk kuota ini. Jika aplikasi Anda memerlukan lebih banyak instans DB, Anda dapat meminta instans DB tambahan dengan membuka Konsol Service Quotas. Di panel navigasi, pilih Layanan AWS . Pilih Amazon Relational Database Service (Amazon RDS), pilih kuota, dan ikuti arahan untuk meminta peningkatan kuota. Untuk informasi selengkapnya, lihat Meminta peningkatan kuota di Panduan Pengguna Service Quotas.
>
> Pencadangan yang dikelola oleh AWS Backup dianggap sebagai snapshot cluster DB manual, tetapi tidak dihitung dalam kuota snapshot cluster manual. Untuk selengkapnya AWS Backup, lihat Panduan AWS Backup Pengembang.

Jika Anda menggunakan operasi API RDS dan melebihi kuota default untuk jumlah panggilan per detik, API Amazon RDS akan mengeluarkan kesalahan seperti berikut.

ClientError: Terjadi kesalahan (ThrottlingException) saat memanggil operasi *API_name*: Nilai terlampaui.

Di sini, kurangi jumlah panggilan per detik. Kuota dimaksudkan untuk mencakup sebagian besar kasus penggunaan. Jika batas yang lebih tinggi diperlukan, mintalah peningkatan kuota dengan menghubungi AWS Support. Buka halaman Pusat AWS Support, masuk jika perlu, dan pilih Buat kasus. Pilih Peningkatan batas layanan. Lengkapi dan kirimkan formulir ini.

> **Note**
>
> Kuota ini tidak dapat diubah di konsol Service Quotas Amazon RDS.

Batasan penamaan dalam Amazon Aurora

Tabel berikut menjelaskan batasan penamaan dalam Amazon Aurora.

Sumber daya atau item	Batasan
Pengidentifikasi klaster DB	Pengidentifikasi memiliki batasan penamaan ini: • Harus berisi 1–63 karakter alfanumerik atau tanda hubung. • Karakter pertamanya harus berupa huruf. • Tidak boleh diakhiri dengan satu tanda hubung atau berisi dua tanda hubung berturut-turut. • Harus unik untuk semua instans DB per AWS akun, per AWS Wilayah.
Nama Basis data awal	Batasan nama basis data berbeda antara Aurora MySQL dan PostgreSQL. Untuk informasi selengkapnya, lihat pengaturan yang tersedia saat membuat masing-masing klaster DB.
Nama pengguna utama	Batasan nama pengguna utama berbeda untuk setiap mesin basis data. Untuk informasi selengkapnya, lihat pengaturan yang tersedia saat membuat masing-masing klaster DB.
Kata sandi master	Kata sandi untuk pengguna utama basis data dapat mencakup karakter ASCII yang dapat dicetak kecuali

Sumber daya atau item	Batasan
	/, ', '', @, atau spasi. Untuk Oracle, & adalah batasan karakter tambahan. Kata sandi memiliki jumlah karakter ASCII yang dapat dicetak berikut ini, bergantung pada mesin DB: • Aurora MySQL: 8–41 • Aurora PostgreSQL: 8–99
Nama grup parameter DB	Nama-nama ini memiliki batasan berikut: • Harus berisi 1–255 karakter alfanumerik. • Karakter pertamanya harus berupa huruf. • Tanda hubung diperbolehkan, tetapi nama tidak boleh diakhiri dengan tanda hubung atau berisi dua tanda hubung berturut-turut.
Nama kelompok subnet DB	Nama-nama ini memiliki batasan berikut: • Harus berisi 1–255 karakter. • Karakter alfanumerik, spasi, tanda hubung, garis bawah, dan titik diperbolehkan.

Batas ukuran Amazon Aurora

Batas ukuran penyimpanan

Volume klaster Aurora dapat berkembang hingga ukuran maksimum 128 tebibyte (TiB) untuk versi mesin berikut:

- Semua versi Aurora MySQL versi 3 yang tersedia; Aurora MySQL versi 2, versi 2.09 dan yang lebih tinggi
- Semua versi Aurora PostgreSQL yang tersedia

Untuk versi mesin yang lebih rendah, ukuran maksimum volume klaster Aurora adalah 64 TiB. Untuk informasi selengkapnya, lihat Cara penyimpanan Aurora berubah ukuran secara otomatis.

Untuk memantau ruang penyimpanan yang tersisa, Anda dapat menggunakan metrik `AuroraVolumeBytesLeftTotal`. Untuk informasi selengkapnya, lihat Metrik tingkat klaster untuk Amazon Aurora.

Batas ukuran tabel SQL

Untuk klaster DB Aurora MySQL, ukuran tabel maksimumnya adalah 64 tebibyte (TiB). Untuk klaster DB Aurora PostgreSQL, ukuran tabel maksimumnya adalah 32 tebibyte (TiB). Sebaiknya Anda mengikuti praktik terbaik desain tabel, seperti mempartisi tabel besar.

Batas ID ruang tabel

ID ruang tabel maksimum untuk Aurora MySQL adalah 2147483647. Jika Anda sering membuat dan menghapus tabel, pastikan untuk mengetahui ID ruang tabel Anda dan rencanakan untuk menggunakan pembuangan logis. Untuk informasi selengkapnya, lihat Migrasi logis dari MySQL ke Amazon Aurora MySQL dengan menggunakan mysqldump.

Pemecahan Masalah untuk Amazon Aurora

Gunakan bagian berikut untuk membantu memecahkan masalah yang Anda miliki dengan instans DB di Amazon RDS dan Amazon Aurora.

Topik

- Tidak dapat terhubung ke instans DB Amazon RDS
- Masalah keamanan Amazon RDS
- Mengatur ulang kata sandi pemilik instans DB
- Penghentian atau boot ulang instans DB Amazon RDS
- Perubahan parameter DB Amazon RDS tidak diberlakukan
- Masalah memori yang dapat dikosongkan di Amazon Aurora
- Amazon Aurora Masalah MySQL out-of-memory
- Masalah replikasi Amazon Aurora MySQL

Untuk informasi tentang masalah debug menggunakan API Amazon RDS, lihat Pemecahan masalah aplikasi di Aurora.

Tidak dapat terhubung ke instans DB Amazon RDS

Jika Anda tidak dapat terhubung ke instans DB, berikut adalah penyebab-penyebab umumnya:

- Aturan masuk – Aturan akses yang diberlakukan oleh firewall lokal dan alamat IP yang diizinkan untuk mengakses instans DB mungkin tidak cocok. Kemungkinan besar masalahnya adalah aturan masuk dalam grup keamanan Anda.

  Secara default, instans DB tidak mengizinkan akses. Akses diberikan melalui grup keamanan yang terkait dengan VPC yang mengizinkan lalu lintas masuk dan keluar dari instans DB. Jika perlu, tambahkan aturan masuk dan keluar untuk situasi khusus Anda ke grup keamanan. Anda dapat menentukan alamat IP, rentang alamat IP, atau grup keamanan VPC lainnya.

  > **Note**
  >
  > Saat menambahkan aturan masuk baru, Anda dapat memilih IP Saya untuk Sumber agar dapat mengizinkan akses ke instans DB dari alamat IP yang terdeteksi di browser.

Untuk informasi selengkapnya tentang menyiapkan grup keamanan, lihat Berikan akses ke klaster DB dalam VPC dengan membuat grup keamanan.

> **Note**
>
> Koneksi klien dari alamat IP di dalam rentang 169.254.0.0/16 tidak diizinkan. Rentang ini adalah Automatic Private IP Addressing Range (APIPA) yang digunakan untuk alamat tautan lokal.

- Aksesibilitas publik – Untuk terhubung ke instans DB dari luar VPC, seperti menggunakan aplikasi klien, instans harus memiliki alamat IP publik yang ditetapkan untuk instans tersebut.

  Agar instans dapat diakses oleh publik, ubah instans dan pilih Ya di bagian Aksesibilitas publik. Untuk informasi selengkapnya, lihat Menyembunyikan instans DB dalam VPC dari internet.
- Port – Port yang Anda tentukan saat membuat instans DB tidak dapat digunakan untuk mengirim atau menerima komunikasi karena batasan firewall lokal Anda. Untuk menentukan apakah jaringan Anda memungkinkan port tertentu digunakan untuk komunikasi masuk dan keluar, hubungi administrator jaringan Anda.
- Ketersediaan – Untuk instans DB yang baru dibuat, instans DB memiliki status `creating` hingga instans DB siap digunakan. Ketika statusnya berubah menjadi `available`, Anda dapat terhubung ke instans DB. Tergantung pada ukuran instans DB Anda, perlu waktu hingga 20 menit sebelum instans tersedia.
- Gateway internet – Agar instans DB dapat diakses publik, subnet dalam grup subnet DB tersebut harus memiliki gateway internet.

  Mengonfigurasi gateway internet untuk subnet

  1. Masuk ke AWS Management Console dan buka konsol Amazon RDS di https://console.aws.amazon.com/rds/.
  2. Di panel navigasi, pilih Basis Data lalu pilih instans DB.
  3. Di tab Konektivitas & keamanan, tuliskan nilai ID VPC di VPC dan subnet ID di Subnet.
  4. Buka konsol Amazon VPC di https://console.aws.amazon.com/vpc/.
  5. Di panel navigasi, pilih Gateway Internet. Pastikan ada gateway internet yang dilampirkan ke VPC Anda. Atau, pilih Buat Gateway Internet untuk membuat gateway internet. Pilih gateway internet, lalu pilih Lampirkan ke VPC dan ikuti arahan untuk melampirkannya ke VPC Anda.

6. Di panel navigasi, pilih Subnet, lalu pilih subnet Anda.
7. Di tab Tabel Rute, pastikan ada rute dengan `0.0.0.0/0` sebagai tujuan dan gateway internet untuk VPC sebagai target.

   Jika Anda terhubung ke instans Anda menggunakan alamat IPv6, pastikan ada rute untuk semua lalu lintas IPv6 (`::/0`) yang mengarah ke gateway internet. Jika tidak, lakukan tindakan berikut:

   a. Pilih ID tabel rute (rtb-xxxxxxxx) untuk menavigasi ke tabel rute.
   b. Di tab Rute, pilih Edit rute. Pilih Tambahkan rute, gunakan `0.0.0.0/0` sebagai tujuan, dan gateway internet sebagai target.

      Untuk IPv6, pilih Tambahkan rute, gunakan `::/0` sebagai tujuan, dan gateway internet sebagai target.
   c. Pilih Simpan rute.

   Selain itu, jika Anda mencoba untuk terhubung ke titik akhir IPv6, pastikan rentang alamat IPv6 klien diizinkan untuk terhubung ke instans DB.

Untuk informasi selengkapnya, lihat Bekerja dengan instans DB dalam VPC.

Menguji koneksi ke instans DB

Anda dapat menguji koneksi ke instans DB menggunakan alat Linux atau Microsoft Windows umum.

Dari terminal Linux atau Unix, Anda dapat menguji koneksi dengan memasukkan hal berikut. Ganti *`DB-instance-endpoint`* dengan titik akhir dan *`port`* dengan port instans DB Anda.

```
nc -zv DB-instance-endpoint port
```

Misalnya, hal berikut menunjukkan contoh perintah dan nilai kembali.

```
nc -zv postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299

  Connection to postgresql1.c6c8mn7fake0.us-west-2.rds.amazonaws.com 8299 port [tcp/
vvr-data] succeeded!
```

Pengguna Windows dapat menggunakan Telnet untuk menguji koneksi ke instans DB. Tindakan Telnet tidak didukung selain untuk menguji koneksi. Jika koneksi berhasil, tindakan tidak akan mengembalikan pesan. Jika koneksi gagal, Anda akan menerima pesan kesalahan seperti berikut.

```
C:\>telnet sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com 819

  Connecting To sg-postgresql1.c6c8mntfake0.us-west-2.rds.amazonaws.com...Could not
 open
  connection to the host, on port 819: Connect failed
```

Jika tindakan Telnet berhasil, artinya grup keamanan Anda dikonfigurasi dengan benar.

> **Note**
>
> Amazon RDS tidak menerima lalu lintas Internet Control Message Protocol (ICMP), termasuk ping.

Memecahkan masalah autentikasi koneksi

Dalam beberapa kasus, Anda dapat terhubung ke instans DB tetapi mendapatkan kesalahan autentikasi. Dalam kasus ini, Anda mungkin ingin mengatur ulang kata sandi pengguna utama untuk instans DB. Anda dapat melakukan tindakan ini dengan mengubah instans RDS.

Masalah keamanan Amazon RDS

Untuk menghindari masalah keamanan, jangan pernah menggunakan nama AWS pengguna dan kata sandi utama Anda untuk akun pengguna. Praktik terbaik adalah menggunakan master Anda Akun AWS untuk membuat pengguna dan menetapkannya ke akun pengguna DB. Anda juga dapat menggunakan akun utama untuk membuat akun pengguna lain, jika perlu.

Untuk informasi tentang membuat pengguna, lihat Membuat pengguna IAM di Akun AWS. Untuk informasi tentang membuat pengguna AWS IAM Identity Center, lihat Mengelola identitas di Pusat Identitas IAM.

Pesan kesalahan “gagal mengambil atribut akun, fungsi konsol tertentu mungkin terganggu.”

Kesalahan ini dapat muncul karena beberapa alasan. Penyebabnya mungkin karena akun Anda kehilangan izin, atau akun Anda belum disiapkan dengan benar. Untuk akun baru, Anda mungkin belum menunggu akun hingga siap digunakan. Untuk akun lama, Anda mungkin tidak memiliki izin dalam kebijakan akses untuk melakukan tindakan tertentu, seperti membuat instans DB. Untuk memecahkan masalah ini, administrator perlu memberikan peran yang diperlukan ke akun Anda. Untuk informasi selengkapnya, lihat dokumentasi IAM.

Mengatur ulang kata sandi pemilik instans DB

Jika Anda terkunci dari klaster DB, Anda dapat masuk sebagai pengguna utama. Kemudian, Anda dapat mengatur ulang kredensial untuk pengguna administratif atau peran lainnya. Jika Anda tidak dapat masuk sebagai pengguna utama, pemilik AWS akun dapat mengatur ulang kata sandi pengguna utama. Untuk detail tentang akun administratif atau peran yang mungkin perlu Anda atur ulang, lihat Hak akses akun pengguna master.

Anda dapat mengubah kata sandi instans DB dengan menggunakan konsol Amazon RDS, AWS CLI perintah modify-db-instance, atau dengan menggunakan operasi ModifydBInstance API. Untuk informasi selengkapnya tentang cara mengubah instans DB dalam klaster DB, lihat Memodifikasi instans DB dalam klaster DB.

Penghentian atau boot ulang instans DB Amazon RDS

Penghentian instans DB dapat terjadi ketika instans DB di-boot ulang. Penghentian ini juga dapat terjadi saat instans DB diubah menjadi status yang mencegah akses ke instans tersebut, dan saat basis data di-boot ulang. Boot ulang dapat terjadi saat Anda melakukan boot ulang manual pada instans DB Anda. Boot ulang juga dapat terjadi saat Anda mengubah pengaturan instans DB yang memerlukan boot ulang sebelum dapat diberlakukan.

Boot ulang instans DB terjadi saat Anda mengubah pengaturan yang memerlukan boot ulang, atau ketika Anda secara manual menyebabkan boot ulang. Boot ulang dapat segera terjadi jika Anda mengubah pengaturan dan meminta perubahan segera diberlakukan. Atau, boot ulang dapat terjadi selama jendela pemeliharaan instans DB.

Boot ulang instans DB segera terjadi ketika salah satu hal berikut terjadi:

- Anda mengubah periode retensi pencadangan untuk instans DB dari 0 ke nilai selain nol atau dari nilai selain nol ke 0. Anda mengatur Terapkan Segera ke `true`.
- Anda mengubah kelas instans DB, dan Terapkan Segera diatur menjadi `true`.

Boot ulang instans DB terjadi selama jendela pemeliharaan saat salah satu dari hal berikut terjadi:

- Anda mengubah periode retensi pencadangan untuk instans DB dari 0 ke nilai selain nol atau dari nilai selain nol ke 0, dan Terapkan Segera diatur ke `false`.
- Anda mengubah kelas instans DB, dan Terapkan Segera diatur menjadi `false`.

Ketika Anda mengubah parameter statis dalam grup parameter DB, perubahan tersebut tidak diberlakukan hingga instans DB yang dikaitkan dengan grup parameter di-boot ulang. Perubahan ini memerlukan boot ulang manual. Instans DB tidak di-boot ulang secara otomatis selama jendela pemeliharaan.

Perubahan parameter DB Amazon RDS tidak diberlakukan

Dalam beberapa kasus, Anda mungkin mengubah parameter dalam grup parameter DB, tetapi tidak melihat perubahan akan diberlakukan. Jika demikian, Anda mungkin perlu melakukan boot ulang instans DB yang dikaitkan dengan grup parameter DB. Saat Anda mengubah parameter dinamis, perubahan akan langsung diberlakukan. Ketika Anda mengubah parameter statis, perubahan tersebut tidak diberlakukan hingga Anda melakukan boot ulang instans DB yang dikaitkan dengan grup parameter.

Anda dapat melakukan boot ulang pada instans DB menggunakan konsol RDS. Atau, Anda dapat secara eksplisit memanggil operasi API `RebootDBInstance`. Anda dapat mem-boot ulang tanpa failover jika instans DB ada dalam deployment Multi-AZ. Persyaratan untuk melakukan boot ulang pada instans DB terkait setelah perubahan parameter statis membantu memitigasi risiko kesalahan konfigurasi parameter yang memengaruhi panggilan API. Contohnya adalah memanggil `ModifyDBInstance` untuk mengubah kelas instans DB. Untuk informasi selengkapnya, lihat Memodifikasi parameter dalam grup parameter DB.

Masalah memori yang dapat dikosongkan di Amazon Aurora

Memori yang dikosongkan adalah total Random Access Memory (RAM) pada instans DB yang dapat dibuat tersedia untuk mesin basis data. Ini adalah jumlah dari memori sistem operasi (OS) yang

kosong serta memori cache halaman dan buffer yang tersedia. Mesin basis data menggunakan sebagian besar memori pada host, tetapi proses OS juga menggunakan sebagian RAM. Memori yang saat ini dialokasikan ke mesin basis data atau digunakan oleh proses OS tidak termasuk dalam memori yang dapat dikosongkan. Ketika mesin basis data kehabisan memori, instans DB dapat menggunakan ruang sementara yang biasanya digunakan untuk buffering dan caching. Seperti disebutkan sebelumnya, ruang sementara ini termasuk dalam memori yang dapat dikosongkan.

Anda menggunakan `FreeableMemory` metrik di Amazon CloudWatch untuk memantau memori yang dapat dibebaskan. Untuk informasi selengkapnya, lihat Ikhtisar metrik pemantauan di Amazon Aurora.

Jika instans DB Anda secara konsisten kehabisan memori yang dapat dikosongkan atau menggunakan ruang swap, pertimbangkan untuk meningkatkan ke kelas instans DB yang lebih besar. Untuk informasi selengkapnya, lihat Kelas instans DB Aurora.

Anda juga dapat mengubah pengaturan memori. Misalnya, pada Aurora MySQL , Anda dapat menyesuaikan ukuran parameter `innodb_buffer_pool_size`. Parameter ini diatur secara default ke 75 persen memori fisik. Untuk tips pemecahan masalah MySQL lainnya, lihat Bagaimana cara memecahkan masalah memori yang dapat dikosongkan rendah di basis data Amazon RDS for MySQL?

Untuk Aurora Serverless v2, `FreeableMemory` mewakili jumlah memori yang tidak terpakai yang tersedia ketika instans DB Aurora Serverless v2 diskalakan ke kapasitas maksimumnya. Anda mungkin memiliki instans yang diturunkan skalanya ke kapasitas yang relatif rendah, tetapi masih melaporkan nilai tinggi untuk `FreeableMemory`, karena instans dapat dinaikkan skalanya. Memori tersebut tidak tersedia saat ini, tetapi Anda bisa mendapatkannya jika Anda membutuhkannya.

Untuk setiap unit kapasitas Aurora (ACU) yang kapasitasnya saat ini di bawah kapasitas maksimum, `FreeableMemory` meningkat sekitar 2 GiB. Dengan demikian, metrik ini tidak mendekati nol sampai instans DB dinaikkan skalanya setinggi mungkin.

Jika metrik ini mendekati nilai `0`, instans DB telah dinaikkan skalanya setinggi mungkin. Skala ini mendekati batas memori yang tersedia. Pertimbangkan untuk meningkatkan pengaturan ACU maksimum untuk klaster. Jika metrik ini mendekati nilai `0` pada instans DB pembaca, pertimbangkan untuk menambahkan instans DB pembaca lain ke klaster. Dengan begitu, bagian hanya baca dari beban kerja dapat tersebar di lebih banyak instans DB, sehingga mengurangi penggunaan memori pada setiap instans DB pembaca. Untuk informasi selengkapnya, lihat CloudWatch Metrik Amazon penting untuk Aurora Serverless v2.

Untuk Aurora Serverless v1, Anda dapat mengubah rentang kapasitas untuk menggunakan lebih banyak ACU. Untuk informasi selengkapnya, lihat Memodifikasi klaster DB Aurora Serverless v1.

Amazon Aurora Masalah MySQL out-of-memory

Parameter tingkat instans `aurora_oom_response` Aurora MySQL dapat memungkinkan instans DB untuk memantau memori sistem dan memperkirakan memori yang digunakan oleh berbagai pernyataan dan koneksi. Jika sistem kehabisan memori, ia dapat melakukan daftar tindakan untuk mencoba melepaskan memori itu. Ia melakukannya dalam upaya untuk menghindari restart database karena masalah out-of-memory (OOM). Parameter tingkat instance mengambil serangkaian tindakan yang dipisahkan koma yang dilakukan instans DB saat memorinya rendah. `aurora_oom_response`Parameter ini didukung untuk Aurora MySQL versi 2 dan 3.

Nilai-nilai berikut, dan kombinasinya, dapat digunakan untuk `aurora_oom_response` parameter. String kosong berarti tidak ada tindakan yang diambil, dan secara efektif mematikan fitur, membuat database rentan terhadap restart OOM.

- `decline`— Menolak kueri baru ketika instans DB rendah pada memori.
- `kill_query`— Mengakhiri kueri dalam urutan konsumsi memori yang menurun hingga memori instance muncul di atas ambang batas rendah. Pernyataan DDL tidak berakhir.

  Untuk informasi selengkapnya, lihat pernyataan KILL dalam dokumentasi MySQL.
- `print`— Hanya mencetak kueri yang menghabiskan sejumlah besar memori.
- `tune` – Menyesuaikan cache tabel internal untuk melepas sebagian memori kembali ke sistem. Aurora MySQL mengurangi memori yang digunakan untuk cache seperti dan dalam kondisi memori rendah. `table_open_cache` `table_definition_cache` Akhirnya, Aurora MySQL mengatur penggunaan memorinya kembali normal ketika memori sistem tidak lagi rendah.

  Untuk informasi selengkapnya, lihat table_open_cache dan table_definition_cache dalam dokumentasi MySQL.

Dalam versi MySQL Aurora lebih rendah dari 3,06, untuk kelas instans DB dengan memori kurang dari atau sama dengan 4 GiB, ketika instance berada di bawah tekanan memori, tindakan default mencakup,,, dan. `print` `tune` `decline` `kill_query` Untuk kelas instance DB dengan memori lebih besar dari 4 GiB, nilai parameter kosong secara default (dinonaktifkan).

Di Aurora MySQL versi 3.06 dan lebih tinggi, untuk kelas instans DB dengan memori kurang dari atau sama dengan 4 GiB, Aurora MySQL juga menutup koneksi memakan memori teratas. Kolam

penyangga InnoDB disetel secara otomatis hingga 70% dari ukuran aslinya. Setelah instance kehabisan tekanan memori, kumpulan buffer InnoDB dikembalikan ke ukuran aslinya. Untuk kelas instance DB dengan memori lebih besar dari 4 GiB, nilai parameter secara defaultprint.

Jika Anda sering mengalami out-of-memory masalah, penggunaan memori dapat dipantau menggunakan tabel ringkasan memori saat performance_schema diaktifkan.

Masalah replikasi Amazon Aurora MySQL

Beberapa masalah replikasi MySQL juga berlaku untuk Aurora MySQL. Anda dapat mendiagnosis dan memperbaiki masalah tersebut.

Topik

- Mendiagnosis dan mengatasi jeda di antara replika baca
- Mendiagnosis dan menyelesaikan kegagalan replikasi baca MySQL
- Kesalahan replikasi terhenti

Mendiagnosis dan mengatasi jeda di antara replika baca

Setelah Anda membuat replika baca MySQL dan replikanya tersedia, Amazon RDS pertama-tama mereplikasi perubahan yang dibuat ke instans DB sumber sejak operasi replika baca dimulai. Selama fase ini, waktu jeda replikasi untuk replika baca lebih besar dari 0. Anda dapat memantau jeda waktu ini di Amazon CloudWatch dengan melihat metrik Amazon RDS.

Metrik AuroraBinlogReplicaLag melaporkan nilai kolom Seconds_Behind_Master dari perintah SHOW REPLICA STATUS MySQL. Untuk informasi selengkapnya, lihat SHOW REPLICA STATUS Statement di dokumentasi MySQL.

Saat metrik AuroraBinlogReplicaLag mencapai 0, replika telah menyamai instans DB sumber. Jika metrik AuroraBinlogReplicaLag menunjukkan -1, replikasi mungkin tidak aktif. Untuk memecahkan masalah kesalahan replikasi, lihat Mendiagnosis dan menyelesaikan kegagalan replikasi baca MySQL. Nilai AuroraBinlogReplicaLag sebesar -1 juga dapat berarti bahwa nilai Seconds_Behind_Master tidak dapat ditentukan atau NULL.

> **Note**
>
> Versi sebelumnya dari Aurora MySQL menggunakan SHOW SLAVE STATUS, bukan SHOW REPLICA STATUS. Jika Anda menggunakan Aurora MySQL versi 1 atau 2, gunakan SHOW

`SLAVE STATUS`. Gunakan `SHOW REPLICA STATUS` untuk Aurora MySQL versi 3 dan yang lebih tinggi.

Metrik `AuroraBinlogReplicaLag` menunjukkan -1 saat penghentian jaringan atau saat patch diterapkan selama jendela pemeliharaan. Dalam kasus ini, tunggu konektivitas jaringan hingga dipulihkan atau tunggu jendela pemeliharaan berakhir sebelum Anda memeriksa metrik `AuroraBinlogReplicaLag` lagi.

Teknologi replikasi baca MySQL bersifat asinkron. Oleh karena itu, Anda dapat sesekali mengharapkan peningkatan bagi metrik `BinLogDiskUsage` pada instans DB sumber dan bagi metrik `AuroraBinlogReplicaLag` pada replika baca. Misalnya, pertimbangkan situasi saat volume operasi tulis tinggi ke instans DB sumber terjadi secara paralel. Pada saat yang sama, operasi tulis ke replika baca akan diserialkan menggunakan thread I/O tunggal. Situasi tersebut dapat menyebabkan jeda antara instans sumber dan replika baca.

Untuk informasi selengkapnya tentang replika baca dan MySQL, lihat Replication implementation details dalam dokumentasi MySQL.

Anda dapat mengurangi lag antara pembaruan ke instans DB sumber dan pembaruan berikutnya ke replika baca dengan melakukan hal berikut:

- Atur kelas instans DB dari replika baca agar memiliki ukuran penyimpanan yang sebanding dengan ukuran dari instans DB sumber.
- Pastikan kompatibilitas pengaturan parameter di grup parameter DB yang digunakan oleh instans DB sumber dan replika baca. Untuk informasi selengkapnya dan contoh, lihat diskusi tentang parameter `max_allowed_packet` di bagian berikutnya.
- Nonaktifkan cache kueri. Untuk tabel yang sering diubah, menggunakan cache kueri dapat meningkatkan lag replika karena cache terkunci dan sering disegarkan. Dalam kasus ini, Anda mungkin akan melihat lebih sedikit lag replika jika menonaktifkan cache kueri. Anda dapat menonaktifkan cache kueri dengan mengatur `query_cache_type parameter` ke 0 dalam grup parameter DB untuk instans DB. Untuk informasi selengkapnya tentang cache kueri, lihat Konfigurasi cache kueri.
- Hangatkan kumpulan buffer pada replika baca untuk InnoDB for MySQL. Misalnya, anggaplah Anda memiliki sejumlah kecil tabel yang sering diperbarui dan Anda menggunakan skema tabel InnoDB atau XtraDB. Dalam kasus ini, dump tabel tersebut pada replika baca. Dengan melakukan hal ini, Anda akan menyebabkan mesin basis data memindai barisan tabel tersebut dari disk, lalu

menyimpannya di dalam kumpulan buffer. Pendekatan ini dapat mengurangi jeda replika. Bagian berikut menunjukkan satu contoh.

Untuk Linux, macOS, atau Unix:

```
PROMPT> mysqldump \
    -h <endpoint> \
    --port=<port> \
    -u=<username> \
    -p <password> \
    database_name table1 table2 > /dev/null
```

Untuk Windows:

```
PROMPT> mysqldump ^
    -h <endpoint> ^
    --port=<port> ^
    -u=<username> ^
    -p <password> ^
    database_name table1 table2 > /dev/null
```

Mendiagnosis dan menyelesaikan kegagalan replikasi baca MySQL

Amazon RDS memantau status replikasi replika baca Anda. RDS memperbarui bidang Status Replikasi instans replika baca menjadi `Error` jika replikasi berhenti karena alasan apa pun. Anda dapat meninjau detail kesalahan terkait yang dilontarkan oleh mesin MySQL dengan melihat kolom Kesalahan Replikasi. Peristiwa yang menunjukkan status replika baca juga dihasilkan, termasuk RDS-EVENT-0045, RDS-EVENT-0046, dan RDS-EVENT-0057. Untuk informasi selengkapnya tentang peristiwa dan berlangganan peristiwa, lihat Bekerja dengan pemberitahuan peristiwa Amazon RDS. Jika pesan kesalahan MySQL muncul, periksa kesalahannya di MySQL error message documentation.

Situasi umum yang dapat menyebabkan kesalahan replikasi mencakup hal-hal berikut:

- Nilai parameter `max_allowed_packet` untuk replika baca lebih kecil dari parameter `max_allowed_packet` untuk instans DB sumber.

  Parameter `max_allowed_packet` adalah parameter kustom yang dapat Anda atur di grup parameter DB. Parameter `max_allowed_packet` digunakan untuk menentukan ukuran

maksimum bahasa manipulasi data (DML) yang dapat dijalankan di basis data. Dalam beberapa kasus, nilai `max_allowed_packet` untuk instans DB sumber mungkin lebih besar dari nilai `max_allowed_packet` untuk replika baca. Jika demikian, proses replikasi dapat menimbulkan kesalahan dan menghentikan replikasi. Kesalahan yang paling umum adalah `packet bigger than 'max_allowed_packet' bytes`. Anda dapat memperbaiki kesalahan ini dengan mengatur agar replika sumber dan baca menggunakan grup parameter DB yang sama dengan nilai parameter `max_allowed_packet`.

- Menulis ke tabel di replika baca. Jika Anda membuat indeks pada replika baca, parameter `read_only` harus diatur ke 0 untuk membuat indeks. Jika Anda menulis ke tabel di replika baca, tindakan ini dapat memecah replikasi.
- Gunakan mesin penyimpanan nontransaksional seperti MyISAM. Replika baca membutuhkan mesin penyimpanan transaksional. Replikasi hanya didukung untuk mesin penyimpanan berikut: InnoDB for MySQL atau MariaDB.

  Anda dapat mengonversi tabel MyISAM ke InnoDB dengan perintah berikut:

  ```
  alter table <schema>.<table_name> engine=innodb;
  ```
- Gunakan kueri nondeterministik yang tidak aman seperti `SYSDATE()`. Untuk informasi selengkapnya, lihat Determination of safe and unsafe statements in binary logging di dokumentasi MySQL.

Langkah-langkah berikut dapat membantu mengatasi kesalahan replikasi Anda:

- Jika Anda mengalami kesalahan logis dan dapat melewatkan kesalahan tersebut dengan aman, ikuti langkah-langkah yang dijelaskan dalam Melewati kesalahan replika saat ini. Instans DB Aurora MySQL Anda harus menjalankan versi yang mencakup prosedur `mysql_rds_skip_repl_error`. Untuk informasi selengkapnya, lihat mysql_rds_skip_repl_error.
- Jika Anda mengalami masalah posisi log biner (binlog), Anda dapat mengubah posisi tayangan ulang replika. Anda melakukannya dengan perintah `mysql.rds_next_master_log` untuk Aurora MySQL versi 1 dan 2. Anda melakukannya dengan perintah `mysql.rds_next_source_log` untuk Aurora MySQL versi 3 dan yang lebih tinggi. Instans DB Aurora MySQL Anda harus menjalankan versi yang mendukung perintah ini untuk mengubah posisi tayangan putar ulang replika. Untuk informasi versi, lihat mysql_rds_next_master_log.
- Jika Anda mengalami masalah performa sementara karena beban DML yang tinggi, Anda dapat mengatur parameter `innodb_flush_log_at_trx_commit` ke 2 dalam grup parameter DB pada replika baca. Dengan melakukan hal ini, Anda dapat membantu replika baca mengejar,

meskipun tindakan ini akan mengurangi atomisitas, konsistensi, isolasi, dan daya tahan (ACID) untuk sementara.

- Anda dapat menghapus replika baca dan membuat instans menggunakan pengidentifikasi instans DB yang sama. Dengan cara ini, titik akhir tetap sama dengan replika baca lama Anda.

Jika kesalahan replikasi diperbaiki, Status Replikasi berubah menjadi mereplikasi. Untuk informasi selengkapnya, lihat Memecahkan masalah replika baca MySQL.

Kesalahan replikasi terhenti

Ketika memanggil perintah `mysql.rds_skip_repl_error`, Anda mungkin menerima pesan kesalahan yang menyatakan bahwa replikasi mati atau dinonaktifkan.

Pesan kesalahan ini muncul karena replikasi dihentikan dan tidak dapat dimulai ulang.

Jika Anda perlu melewati sejumlah besar kesalahan, lag replikasi dapat meningkat hingga melampaui periode retensi default untuk file log biner. Dalam kasus ini, Anda mungkin mengalami kesalahan fatal karena file log biner dihapus sebelum diputar ulang di replika. Penghapusan ini menyebabkan replikasi berhenti, dan Anda tidak dapat lagi memanggil perintah `mysql.rds_skip_repl_error` untuk melewati kesalahan replikasi.

Anda dapat memitigasi masalah ini dengan meningkatkan jumlah jam penyimpanan file log biner pada sumber replikasi Anda. Setelah meningkatkan waktu retensi binlog, Anda dapat memulai ulang replikasi dan memanggil perintah `mysql.rds_skip_repl_error` sesuai kebutuhan.

Untuk mengatur waktu retensi binlog, gunakan prosedur mysql_rds_set_configuration. Tentukan parameter konfigurasi 'jam retensi binlog' sekaligus jumlah jam untuk menyimpan file binlog di klaster DB, hingga 2160 (90 hari). Default untuk Aurora MySQL adalah 24 (1 hari). Contoh berikut menetapkan periode penyimpanan file binlog menjadi 48 jam.

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

Referensi Amazon RDS API

Selain AWS Management Console dan AWS Command Line Interface (AWS CLI), Amazon RDS juga menyediakan API. Anda dapat menggunakan API untuk mengotomatiskan tugas untuk mengelola instans DB dan objek lain di Amazon RDS.

- Untuk daftar abjad operasi API, lihat Tindakan.
- Untuk daftar abjad jenis data, lihat Jenis data.
- Untuk daftar parameter kueri umum, lihat Parameter umum.
- Untuk deskripsi kode kesalahan, lihat Kesalahan umum.

Untuk informasi selengkapnya tentang AWS CLI, lihat Referensi AWS Command Line Interface untuk Amazon RDS.

Topik

- Menggunakan API Kueri
- Pemecahan masalah aplikasi di Aurora

Menggunakan API Kueri

Bagian berikut membahas secara singkat parameter dan autentikasi permintaan yang digunakan untuk API Kueri.

Untuk informasi umum tentang cara kerja API Kueri, lihat Permintaan kueri dalam Referensi API Amazon EC2.

Parameter Kueri

Permintaan berbasis Kueri HTTP adalah permintaan yang menggunakan parameter HTTP verb GET atau POST dan parameter Kueri yang bernama `Action`.

Setiap permintaan Kueri harus menyertakan beberapa parameter umum untuk menangani autentikasi dan pemilihan tindakan.

Beberapa operasi mengambil daftar parameter. Daftar ini ditentukan menggunakan notasi `param.`*n*. Nilai *n* adalah bilangan bulat yang dimulai dari 1.

Untuk informasi tentang Wilayah dan titik akhir Amazon RDS, buka ke Amazon Relational Database Service (RDS) di bagian Wilayah dan Titik Akhir dalam Referensi Umum Amazon Web.

Autentikasi permintaan Kueri

Anda hanya dapat mengirim permintaan Kueri melalui HTTPS, dan Anda harus menyertakan signature di setiap permintaan Kueri. Anda harus menggunakan AWS signature versi 4 atau signature versi 2. Untuk informasi selengkapnya, lihat Proses penandatanganan Signature Versi 4 dan Proses penandatanganan Signature versi 2.

Pemecahan masalah aplikasi di Aurora

Amazon RDS memberikan penjelasan tentang kesalahan spesifik dan deskriptif untuk membantu Anda memecahkan masalah saat berinteraksi dengan API Amazon RDS.

Topik

- Mengambil kesalahan
- Tips pemecahan masalah

Untuk informasi tentang pemecahan masalah untuk instans DB Amazon RDS, lihat Pemecahan Masalah untuk Amazon Aurora.

Mengambil kesalahan

Biasanya, Anda ingin aplikasi memeriksa apakah permintaan membuat kesalahan sebelum Anda menghabiskan waktu untuk memproses hasil. Cara termudah untuk mengetahui jika terjadi kesalahan adalah dengan mencari simpul `Error` di dalam respons dari API Amazon RDS.

Sintaksis XPath memberikan cara sederhana untuk mencari keberadaan simpul `Error`. Sintaksis ini juga memberikan cara yang relatif mudah untuk mengambil kode kesalahan dan pesan. Cuplikan kode berikut menggunakan modul Perl dan XML::XPath untuk menentukan jika kesalahan terjadi selama permintaan. Jika terjadi kesalahan, kode akan mencetak pesan dan kode kesalahan pertama dalam responsnya.

```
use XML::XPath;
    my $xp = XML::XPath->new(xml =>$response);
    if ( $xp->find("//Error") )
    {print "There was an error processing your request:\n", " Error code: ",
```

```
    $xp->findvalue("//Error[1]/Code"), "\n", " ",
    $xp->findvalue("//Error[1]/Message"), "\n\n"; }
```

Tips pemecahan masalah

Sebaiknya lakukan proses berikut untuk mendiagnosis dan menyelesaikan masalah dengan API Amazon RDS:

- Verifikasi bahwa Amazon RDS beroperasi secara normal di Wilayah AWS yang Anda targetkan dengan memeriksa http://status.aws.amazon.com.
- Periksa struktur permintaan Anda.

  Setiap operasi Amazon RDS memiliki halaman referensi di Referensi API Amazon RDS. Periksa ulang bahwa Anda menggunakan parameter dengan benar. Untuk mengetahui kemungkinan kesalahan, lihat contoh permintaan atau skenario pengguna untuk melihat apakah contoh tersebut melakukan operasi serupa.
- Periksa AWS re:Post.

  Amazon RDS memiliki komunitas pengembangan tempat Anda dapat mencari solusi untuk masalah yang dialami orang lain selama ini. Untuk melihat topik, buka AWS re:Post.

Riwayat dokumen

Versi API saat ini: 2014-10-31

Tabel berikut menjelaskan perubahan penting pada Panduan Pengguna Amazon Aurora. Untuk notifikasi tentang pembaruan dokumentasi ini, Anda dapat berlangganan feed RSS. Untuk informasi tentang Amazon Relational Database Service (Amazon RDS), lihat Panduan Pengguna Amazon Relational Database Service.

> **Note**
>
> Sebelum 31 Agustus 2018, Amazon Aurora didokumentasikan dalam Panduan Pengguna Amazon Relational Database Service. Untuk riwayat dokumen Aurora sebelumnya, lihat Riwayat dokumen dalam Panduan Pengguna Amazon Relational Database Service.

Anda dapat memfilter fitur Amazon Aurora baru di halaman Hal-Hal Baru dalam Basis Data. Untuk Produk, pilih Amazon Aurora. Kemudian, cari menggunakan kata kunci seperti **`global database`** atau **`Serverless`**.

Perubahan	Deskripsi	Tanggal
Dukungan Amazon RDS yang Diperluas	Membuat atau memulihkan Aurora MySQL versi 2 atau 3, atau database Aurora PostgreSQL versi 11 sekarang secara otomatis mendaftar kan database tersebut ke Amazon RDS Extended Support sehingga aplikasi Anda yang ada terus berfungsi sebagaimana adanya. Anda dapat memilih keluar dari RDS Extended Support untuk menghindari biaya setelah Aurora berakhir tanggal dukungan standar untuk	Maret 21, 2024

	mesin database Anda. Untuk informasi selengkapnya, lihat Menggunakan Dukungan Amazon RDS yang Diperluas.	
Pemfilteran data untuk integrasi nol-ETL	Amazon RDS mendukung pemfilteran data di database dan tingkat tabel untuk integrasi nol-ETL dengan Amazon Redshift. Untuk informasi selengkapnya, lihat Pemfilteran data untuk integrasi Aurora Zero-ETL dengan Amazon Redshift.	Maret 20, 2024
Integrasi Aurora MySQL dengan Amazon Bedrock	Anda sekarang dapat mengintegrasikan database Amazon Aurora MySQL dengan Amazon Bedrock untuk memberi daya pada aplikasi AI generatif. Untuk informasi selengkapnya, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora MySQL.	8 Maret 2024

Kebijakan AWS terkelola baru	Amazon RDS menambahkan kebijakan terkelola baru yang diberi nama `AmazonRDSCustomInstanceProfileRolePolicy` untuk memungkinkan RDS Custom melakukan tindakan otomatisasi dan tugas manajemen database melalui profil instans EC2. Untuk informasi selengkapnya, lihat Pembaruan Amazon RDS terhadap kebijakan terkelola AWS.	Februari 27, 2024
Dukungan Amazon RDS untuk AWS Secrets Manager di Wilayah Israel (Tel Aviv)	Amazon RDS mendukung Secrets Manager di Wilayah Israel (Tel Aviv). Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan Amazon RDS dan AWS Secrets Manager.	Februari 21, 2024
Dukungan Amazon RDS yang Diperluas	Amazon RDS sekarang secara otomatis mengaktifkan Amazon RDS Extended Support ketika Aurora MySQL dan Aurora PostgreSQL versi mesin utama di cluster DB Anda dan cluster global mencapai akhir Aurora dari tanggal dukungan standar. Untuk informasi selengkapnya, lihat Menggunakan Dukungan Amazon RDS yang Diperluas.	Februari 15, 2024

Aurora PostgreSQL 16.1 mendukung Babelfish untuk Aurora PostgreSQL 4.0.0	Aurora PostgreSQL 16.1 mendukung Babelfish 4.0.0. Untuk daftar fitur baru, lihat 16.1. Untuk daftar fitur yang didukung oleh setiap rilis Babelfish, lihat Fungsionalitas yang didukung di Babelfish menurut versi. Untuk informasi penggunaan, lihat Bekerja dengan Babelfish for Aurora PostgreSQL.	Januari 31, 2024
Perbarui ke Sertifikat CA default	Sertifikat CA default diatur `rds-ca-rsa2048-g1` ke . Untuk informasi selengkapnya, lihat Menggunakan SSL/TLS untuk mengenkripsi koneksi ke cluster DB.	Januari 26, 2024
RDS Proxy tersedia di Wilayah Eropa (Spanyol)	RDS Proxy sekarang tersedia di wilayah Eropa (Spanyol) . Untuk informasi selengkap nya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS.	8 Januari 2024

RDS Data API dengan Aurora PostgreSQL Serverless v2 dan disediakan	Anda sekarang dapat menggunakan RDS Data API dengan Aurora PostgreSQL Serverless v2 dan cluster DB yang disediakan. Dengan RDS Data API, Anda dapat mengakses cluster Aurora Anda melalui titik akhir HTTP yang aman dan menjalankan pernyataan SQL tanpa menggunakan driver database atau mengelola koneksi. Untuk informasi selengkapnya, lihat Menggunakan API Data RDS.	21 Desember 2023
Integrasi Aurora PostgreSQL dengan Amazon Bedrock	Anda sekarang dapat mengintegrasikan database Amazon Aurora PostgreSQL dengan Amazon Bedrock untuk memberi daya pada aplikasi AI generatif. Untuk informasi selengkapnya, lihat Menggunakan pembelajaran mesin Amazon Aurora dengan Aurora PostgreSQL.	21 Desember 2023
Amazon Aurora tersedia di Wilayah Kanada Barat (Calgary)	Amazon Aurora sekarang tersedia di Wilayah Kanada Barat (Calgary). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan.	Desember 20, 2023

Amazon RDS mendukung melihat dan menanggapi rekomendasi	Rekomendasi Amazon Aurora sekarang mencakup rekomendasi reaktif berbasis proaktif dan pembelajaran mesin berbasis ambang batas. Untuk informasi selengkapnya, lihat Melihat dan menanggapi rekomendasi Amazon Aurora.	Desember 19, 2023
Integrasi nol-ETL Aurora PostgreSQL dengan Amazon Redshift (pratinjau)	Anda kini dapat membuat integrasi nol-ETL dengan Amazon Redshift menggunakan klaster DB sumber Aurora PostgreSQL. Untuk rilis pratinjau, Anda harus membuat semua integrasi di Lingkungan Pratinjau Database Amazon RDS, di Timur AS (Ohio) (us-timur-2). Wilayah AWS Untuk informasi selengkapnya, lihat Bekerja dengan integrasi nol-ETL Aurora dengan Amazon Redshift.	28 November 2023
Amazon Aurora PostgreSQL mendukung penerusan tulis basis data global	Anda kini dapat mengaktifkan fungsi penerusan tulis pada klaster sekunder dalam basis data global yang berbasis Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menggunakan fungsi penerusan tulis dalam basis data global Aurora PostgreSQL.	9 November 2023

Dukungan Aurora PostgreSQL untuk Optimized Reads	Anda dapat menyelesaikan pemrosesan kueri yang lebih cepat untuk Aurora PostgreSQL dengan Aurora Optimized Reads. Untuk informasi selengkapnya, lihat Meningkatkan performa kueri untuk Aurora PostgreSQL dengan Aurora Optimized Reads.	8 November 2023
Amazon RDS mengekspor metrik Performance Insights ke Amazon CloudWatch	Performance Insights memungkinkan Anda mengekspor dasbor metrik yang telah dikonfigurasi sebelumnya atau kustom ke Amazon. CloudWatch Dasbor metrik yang diekspor tersedia untuk dilihat di konsol. CloudWatch Anda juga dapat mengekspor widget metrik Performance Insights yang dipilih dan melihat data metrik di konsol. CloudWatch Untuk informasi selengkapnya, lihat Mengekspor metrik Performance Insights ke. CloudWatch	8 November 2023
Integrasi nol-ETL Aurora MySQL dengan ketersediaan umum Amazon Redshift	Integrasi nol-ETL dengan Amazon Redshift kini tersedia secara umum untuk Aurora MySQL. Untuk informasi selengkapnya, lihat Bekerja dengan integrasi nol-ETL Aurora dengan Amazon Redshift.	7 November 2023

Aurora PostgreSQL mendukung Deployment Blue/Green RDS	Anda kini dapat membuat deployment blue/green dari klaster DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menggunakan Deployment Blue/Green Amazon RDS untuk pembaruan basis data.	26 Oktober 2023
Aurora MySQL mendukung enkripsi sisi server dengan (SSE-KMS) AWS KMS keys	Di Aurora MySQL versi 3.05 dan yang lebih tinggi, Anda dapat menggunakan SSE-KMS, termasuk Kunci yang dikelola AWS dan kunci yang dikelola pelanggan, untuk enkripsi data sisi server yang Anda muat atau simpan ke Amazon S3. Untuk informasi selengkapnya, lihat Memuat data ke klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3 dan Menyimpan data dari klaster DB Amazon Aurora MySQL dari file teks di bucket Amazon S3.	25 Oktober 2023

Pengoptimalan Aurora MySQL mengurangi waktu mulai ulang basis data	Di Aurora MySQL versi 3.05 dan yang lebih tinggi, kami telah memperken alkan pengoptimalan yang mengurangi waktu mulai ulang basis data. Pengoptim alan ini menghasilkan periode nonaktif hingga 65% lebih sedikit daripada tanpa pengoptimalan, dan lebih sedikit gangguan pada beban kerja basis data Anda, setelah memulai ulang. Untuk informasi selengkapnya, lihat Pengoptimalan untuk mengurangi waktu mulai ulang basis data.	25 Oktober 2023
Memperbarui ke kebijakan AWS terkelola	Kebijakan terkelola `AmazonRDSPerforman ceInsightsReadOnly` dan `AmazonRDSPerforman ceInsightsFullAcce ss` sekarang menyertak an `Sid` (ID pernyataan) sebagai pengidentifikasi dalam pernyataan kebijakan. Untuk informasi selengkapnya, lihat Pembaruan Amazon RDS terhadap kebijakan terkelola AWS.	23 Oktober 2023

Amazon RDS menerbitkan metrik penghitung Performance Insights ke Amazon CloudWatch	Fungsi matematika metrik DB_PERF_INSIGHTS di konsol CloudWatch memungkinkan Anda melakukan kueri Amazon RDS untuk metrik penghitung Performance Insights. Untuk informasi selengkapnya, lihat Membuat CloudWatch alarm untuk memantau Amazon Aurora.	20 September 2023
Amazon Aurora mendukung point-in-time pemulihan (PITR) dengan AWS Backup	Anda sekarang dapat mengelola pencadangan otomatis (berkelanjutan) Aurora AWS Backup dan mengembalikan ke waktu yang ditentukan dari mereka. Untuk informasi selengkapnya, lihat Memulihkan klaster DB ke waktu tertentu menggunakan AWS Backup.	7 September 2023
Dukungan Amazon RDS yang Diperluas	Amazon Aurora mengumumkan kemampuan mendatang untuk terus menjalankan versi mesin utama Aurora MySQL dan Aurora PostgreSQL di instans DB Anda setelah akhir tanggal dukungan standar Aurora. Untuk informasi selengkapnya, lihat Menggunakan Dukungan Amazon RDS yang Diperluas.	1 September 2023

Amazon Aurora MySQL memperluas dukungan untuk Percona XtraBackup	Anda sekarang dapat melakukan migrasi fisik basis data MySQL 8.0 ke klaster DB Aurora MySQL versi 3. Untuk informasi selengkapnya, lihat Migrasi fisik dari MySQL menggunakan XtraBackup Percona dan Amazon S3.	24 Agustus 2023
Basis data global Aurora mendukung failover basis data global	Basis data global Aurora kini mendukung failover global terkelola, yang memungkinkan Anda untuk lebih mudah pulih dari bencana alam regional atau penonaktifan total tingkat layanan. Untuk mempelajari lebih lanjut fitur ini, lihat Melakukan failover terkelola untuk basis data global Aurora. Fitur yang sebelumnya disebut "failover terencana yang dikelola" sekarang disebut "switchover". Untuk informasi tentang switchover, lihat Melakukan switchover untuk basis data global Amazon Aurora.	21 Agustus 2023

Memperbarui ke izin kebijakan AWS terkelola	Kebijakan terkelola `AmazonRDSFullAccess` memiliki izin baru yang memungkinkan Anda membuat, melihat, dan menghapus laporan analisis performa untuk jangka waktu tertentu. Untuk informasi selengkapnya, lihat pembaruan Amazon RDS ke kebijakan AWS terkelola.	17 Agustus 2023
Memperbarui ke izin kebijakan AWS terkelola	Penambahan izin baru ke kebijakan terkelola `AmazonRDSPerformanceInsightsReadOnly` dan penambahan kebijakan terkelola baru `AmazonRDSPerformanceInsightsFullAccess` memungkinkan Anda membuat laporan analisis beban DB untuk jangka waktu tertentu. Untuk informasi selengkapnya, lihat pembaruan Amazon RDS ke kebijakan AWS terkelola.	16 Agustus 2023

Amazon RDS mendukung analisis periode waktu muat DB dengan Wawasan Performa	Wawasan Performa memungkinkan Anda membuat laporan analisis performa untuk periode waktu tertentu. Laporan ini memberikan wawasan yang diidentifikasi dan rekomendasi untuk mengatasi masalah performa. Untuk informasi selengkapnya, lihat Menganalisis beban DB untuk jangka waktu tertentu.	16 Agustus 2023
Amazon Aurora mendukung fitur retensi cadangan otomatis untuk klaster DB	Anda kini dapat mempertahankan cadangan otomatis untuk klaster Aurora yang dihapus dan memulihkannya ke titik waktu tertentu. Untuk informasi selengkapnya, lihat Mempertahankan cadangan otomatis.	4 Agustus 2023
Amazon Aurora tersedia di Wilayah Israel (Tel Aviv)	Amazon Aurora kini tersedia di Wilayah Israel (Tel Aviv). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan.	1 Agustus 2023
Amazon Aurora MySQL mendukung penerusan tulis lokal (dalam klaster)	Anda kini dapat meneruskan operasi tulis dari instans DB pembaca ke instans DB penulis dalam klaster DB Aurora MySQL. Untuk informasi selengkapnya, lihat Menggunakan fungsi penerusan tulis lokal di klaster DB Amazon Aurora MySQL.	31 Juli 2023

Amazon Aurora mendukung Aurora Serverless v2 tambahan Wilayah AWS	Anda sekarang dapat membuat cluster Aurora Serverless v2 DB di Asia Pasifik (Melbourne) Wilayah AWS. Untuk informasi selengkapnya tentang Aurora Serverless v2, lihat Menggunakan Aurora Serverless v2.	28 Juni 2023
Amazon Aurora memperkenalkan integrasi nol-ETL dengan Amazon Redshift (pratinjau)	Integrasi nol-ETL memberikan solusi yang terkelola sepenuhnya untuk menyediakan data transaksional di Amazon Redshift dalam hitungan detik setelah ditulis ke klaster DB Aurora MySQL. Untuk informasi selengkapnya, lihat Bekerja dengan integrasi nol-ETL Aurora dengan Amazon Redshift.	28 Juni 2023
Amazon RDS menyediakan gabungan Performance Insights CloudWatch dan tampilan metrik di dasbor Performance Insights	Amazon RDS kini menyediakan tampilan konsolidasi Performance Insights CloudWatch dan metrik di dasbor Performance Insights. Untuk informasi selengkapnya, lihat Melihat metrik gabungan di konsol Amazon RDS.	24 Mei 2023
Amazon Aurora mendukung kelas instans db.r7g	Anda kini dapat menggunakan kelas instans db.r7g untuk membuat klaster DB Aurora. Untuk informasi selengkapnya, lihat Kelas instans DB Aurora.	11 Mei 2023

Amazon Aurora mendukung konfigurasi penyimpanan klaster DB baru	Dengan Aurora I/O-Optimized, Anda hanya membayar penggunaan dan penyimpanan klaster DB Anda, tanpa biaya tambahan untuk operasi I/O baca dan tulis. Untuk informasi selengkapnya, lihat Konfigurasi penyimpanan untuk klaster DB Amazon Aurora.	11 Mei 2023
Amazon Aurora mendukung tambahan Aurora Serverless v2Wilayah AWS	Anda sekarang dapat membuat cluster Aurora Serverless v2 DB sebagai berikut Wilayah AWS: Asia Pasifik (Hyderabad), Eropa (Spanyol) Eropa (Zurich), dan Timur Tengah (UEA). Untuk informasi selengkapnya tentang Aurora Serverless v2, lihat Menggunakan Aurora Serverless v2.	4 Mei 2023
Aurora Serverless v1 mendukung konversi ke yang disediakan	Anda dapat mengonversi klaster DB Aurora Serverless v1 langsung ke klaster DB yang disediakan. Untuk informasi selengkapnya, lihat Mengonversi klaster DB Aurora Serverless v1 ke yang disediakan.	27 April 2023

Aurora Serverless v1 mendukung Amazon Aurora PostgreSQL versi 13	Anda kini dapat membuat klaster DB Aurora Serverless v1 yang menjalankan Aurora PostgreSQL versi 13. Untuk informasi selengkapnya, lihat Aurora Serverless v1.	27 April 2023
Dukungan Amazon Aurora untuk AWS Secrets Manager di Wilayah China	Amazon Aurora mendukung Secrets Manager di Wilayah Tiongkok (Beijing) dan Tiongkok (Ningxia). Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan Amazon Aurora dan AWS Secrets Manager.	20 April 2023
Amazon Aurora mendukung peristiwa penerbitan dengan tag ke pelanggan topik	Pemberitahuan acara Amazon Aurora yang dikirim ke Amazon Simple Notificat ion Service (Amazon SNS) atau EventBridge Amazon sekarang berisi tag peristiwa di badan pesan. Tag ini menyediakan data sumber daya yang dipengaruhi oleh peristiwa layanan. Untuk informasi selengkapnya, lihat Tag dan atribut notifikasi peristiwa Amazon RDS.	17 April 2023

Memperbarui ke izin peran yang terkait dengan layanan IAM	`AmazonRDSReadOnlyAccess` Kebijakan `AmazonRDSFullAccess` dan sekarang memberikan izin tambahan untuk memungkinkan tampilan temuan Amazon DevOps Guru di konsol RDS. Untuk informasi selengkapnya, lihat pembaruan Amazon RDS ke kebijakan AWS terkelola.	30 Maret 2023
Amazon Aurora mendukung basis data global di Wilayah Asia Pasifik (Melbourne)	Anda kini dapat membuat basis data global Aurora di Wilayah Asia Pasifik (Melbourne). Untuk informasi tentang basis data global Aurora, lihat Menggunakan basis data global Amazon Aurora.	22 Maret 2023
Memperbarui ke izin kebijakan AWS terkelola	`AmazonRDSReadOnlyAccess` Kebijakan `AmazonRDSFullAccess` dan sekarang memberikan izin tambahan ke Amazon. CloudWatch Untuk informasi selengkapnya, lihat pembaruan Amazon RDS ke kebijakan AWS terkelola.	16 Maret 2023
Proksi RDS tersedia di Wilayah Tiongkok	Proksi RDS sekarang tersedia di Wilayah Tiongkok (Beijing) dan Tiongkok (Ningxia). Untuk informasi selengkap nya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS.	15 Maret 2023

Amazon Aurora mendukung Aurora Serverless v2 di Wilayah Tiongkok	Aurora Serverless v2 kini tersedia di Wilayah Tiongkok (Beijing) dan Tiongkok (Ningxia). Untuk informasi selengkapnya, lihat Aurora Serverless v2.	15 Maret 2023
Proksi RDS tersedia di Wilayah Asia Pasifik (Jakarta)	Sekarang Proksi RDS tersedia di Wilayah Asia Pasifik (Jakarta). Untuk informasi selengkapnya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS.	8 Maret 2023
Amazon Aurora MySQL mendukung autentikasi Kerberos	Anda kini dapat menggunakan autentikasi Kerberos untuk mengautentikasi pengguna saat mereka terhubung ke klaster DB Aurora MySQL Anda. Untuk informasi selengkapnya, lihat Menggunakan autentikasi Kerberos untuk Aurora MySQL.	8 Maret 2023

Amazon Aurora mendukung database global sebagai tambahan Wilayah AWS	Anda kini dapat membuat basis data global Aurora di Wilayah berikut: Afrika (Cape Town), Asia Pasifik (Hong Kong), Asia Pasifik (Hyderabad), Asia Pasifik (Jakarta), Eropa (Milan), Eropa (Spanyol), Eropa (Zürich), Timur Tengah (Bahrain), dan Timur Tengah (UAE). Untuk informasi tentang basis data global Aurora, lihat Menggunakan basis data global Amazon Aurora.	6 Maret 2023
Amazon Aurora mendukung penyalinan snapshot cluster DB sebagai tambahan Wilayah AWS	Anda kini dapat menyalin snapshot klaster DB di Wilayah berikut: Afrika (Cape Town), Asia Pasifik (Hong Kong), Asia Pasifik (Hyderabad), Asia Pasifik (Jakarta), Asia Pasifik (Melbourne), Eropa (Milan), Eropa (Spanyol), Eropa (Zürich), Timur Tengah (Bahrain), dan Timur Tengah (UAE). Untuk informasi tentang penyalinan snapshot klaster DB, lihat Menyalin snapshot klaster DB.	6 Maret 2023

Amazon DevOps Guru untuk RDS mendukung wawasan proaktif	Amazon DevOps Guru for RDS menerbitkan wawasan proaktif dengan rekomendasi untuk membantu Anda mengatasi masalah di database Aurora Anda sebelum diprediksi terjadi. Untuk informasi selengkapnya, lihat Cara Kerja DevOps Guru for RDS.	28 Februari 2023
Amazon Aurora MySQL versi 1 tidak digunakan lagi	Aurora MySQL versi 1 (kompatibel dengan MySQL 5.6) tidak digunakan lagi. Untuk informasi selengkapnya, lihat Sisa waktu ketersediaan Amazon Aurora versi utama.	28 Februari 2023
Aurora Serverless v1 mendukung pengaturan periode pemeliharaan klaster DB	Anda kini dapat mengatur periode pemeliharaan untuk klaster DB Aurora Serverless v1. Untuk informasi selengkapnya, lihat Menyesuaikan periode pemeliharaan klaster DB yang diinginkan.	27 Februari 2023
Amazon Aurora mendukung Aliran Aktivitas Basis Data di Wilayah Asia Pasifik (Hyderabad), Eropa (Spanyol), dan Timur Tengah (UEA).	Untuk informasi selengkapnya, lihat Aliran Aktivitas Basis Data.	27 Januari 2023
Amazon Aurora tersedia di Wilayah Asia Pasifik (Melbourne)	Amazon Aurora kini tersedia di Wilayah Asia Pasifik (Melbourne). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan.	23 Januari 2023

Tentukan otoritas sertifikat (CA) selama pembuatan klaster DB	Anda kini dapat menentukan CA mana yang akan digunakan untuk sertifikat server klaster DB selama pembuatan klaster DB. Untuk informasi selengkapnya, lihat Otoritas sertifikat.	5 Januari 2023
Dukungan Aurora MySQL 3.* untuk pelacakan mundur	Aurora MySQL versi 3.* kini menawarkan sebuah cara cepat untuk pulih dari kesalahan pengguna, seperti meletakkan tabel yang salah atau menghapus baris yang salah. Pelacakan mundur memungkinkan Anda memindahkan basis data ke titik waktu sebelumnya tanpa harus memulihkan dari cadangan, dan dapat diselesaikan dalam hitungan detik, bahkan untuk basis data yang besar. Untuk perincian, lihat Melacak mundur klaster DB Aurora.	4 Januari 2023
Gunakan Amazon RDS Blue/Green Deployment yang tersedia dalam tambahan Wilayah AWS	Fitur Deployment Blue/Green kini tersedia di Wilayah Tiongkok (Beijing) dan Tiongkok (Ningxia). Untuk informasi selengkapnya, lihat Menggunakan Deployment Blue/Green Amazon RDS untuk pembaruan basis data.	22 Desember 2022

Pembaruan terhadap izin peran yang ditautkan layanan IAM	ServiceRolePolicy Kebijakan AmazonRDS sekarang memberikan izin tambahan untuk. AWS Secrets Manager Untuk informasi selengkapnya, lihat pembaruan Amazon RDS ke kebijakan AWS terkelola.	22 Desember 2022
Amazon Aurora terintegrasi dengan AWS Secrets Manager manajemen kata sandi	Aurora dapat mengelola kata sandi pengguna master untuk klaster DB di Secrets Manager. Untuk informasi selengkapnya, lihat Manajemen kata sandi dengan Amazon Aurora dan AWS Secrets Manager.	22 Desember 2022
Amazon Aurora mendukung tambahan Aurora Serverless v2Wilayah AWS	Aurora Serverless v2 kini tersedia di Wilayah Afrika (Cape Town) dan Eropa (Milan). Untuk informasi selengkapnya, lihat Aurora Serverless v2.	21 Desember 2022
Aurora PostgreSQL mendukung Proksi RDS dengan PostgreSQL 14	Anda kini dapat membuat Proksi RDS dengan klaster DB Aurora PostgreSQL 14. Untuk informasi selengkapnya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS.	13 Desember 2022

Amazon Aurora memberi tahu Anda tentang anomali terbaru yang terdeteksi oleh Amazon Guru DevOps	Halaman detail basis data konsol memberi tahu Anda tentang anomali saat ini dan anomali yang terjadi dalam 24 jam terakhir. Untuk informasi selengkapnya, lihat Cara Kerja DevOps Guru for RDS.	13 Desember 2022
Proksi Amazon RDS mendukung basis data global	Anda kini dapat menggunakan Proksi RDS dengan basis data global Aurora. Untuk informasi selengkapnya, lihat Menggunakan Proksi RDS dengan basis data global Aurora.	7 Desember 2022
Klaster DB Aurora PostgreSQL mendukung Ekstensi Bahasa Tepercaya untuk PostgreSQL	Ekstensi Bahasa Tepercaya untuk PostgreSQL adalah kit pengembangan sumber terbuka yang memungkinkan Anda membuat ekstensi PostgreSQL beperforma tinggi dan menjalankannya dengan aman di klaster DB Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Bekerja dengan Ekstensi Bahasa Tepercaya untuk PostgreSQL.	30 November 2022

Monitor Amazon GuardDuty RDS Protection untuk ancaman akses	Saat Anda mengaktifkan Perlindungan GuardDuty RDS, GuardDuty mengkonsumsi peristiwa login RDS dari database Aurora Anda, memantau peristiwa ini, dan memprofilkannya untuk potensi ancaman orang dalam atau aktor eksternal. Ketika GuardDuty RDS Protection mendeteksi potensi ancaman, GuardDuty menghasilkan temuan baru dengan rincian tentang database yang berpotensi dikompromikan. Untuk informasi selengkapnya, lihat Memantau ancaman dengan Perlindungan GuardDuty RDS.	30 November 2022
Gunakan Deployment Blue/Green Amazon RDS untuk pembaruan basis data	Anda dapat membuat perubahan pada klaster DB di lingkungan staging dan menguji perubahan tersebut tanpa memengaruhi klaster DB produksi Anda. Jika sudah siap, Anda dapat mempromosikan lingkungan staging menjadi lingkungan produksi baru, dengan periode nonaktif yang minimal. Untuk informasi selengkapnya, lihat Menggunakan Deployment Blue/Green Amazon RDS untuk pembaruan basis data.	27 November 2022

Amazon Aurora tersedia di Wilayah Asia Pasifik (Hyderabad)	Amazon Aurora kini tersedia di Wilayah Asia Pasifik (Hyderabad). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan.	22 November 2022
Amazon Aurora tersedia di Wilayah Eropa (Spanyol)	Amazon Aurora kini tersedia di Wilayah Eropa (Spanyol) . Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan.	16 November 2022
Amazon Aurora tersedia di Wilayah Eropa (Zürich)	Amazon Aurora kini tersedia di Wilayah Eropa (Zürich). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan.	9 November 2022
Amazon Aurora mendukung pengeksporan data ke Amazon S3 dari klaster DB	Anda kini dapat mengekspor data klaster Aurora langsung ke S3, tanpa harus membuat snapshot terlebih dahulu. Untuk informasi selengkapnya, lihat Mengekspor data klaster DB ke Amazon S3.	27 Oktober 2022
Amazon Aurora MySQL mendukung ekspor yang lebih cepat ke Amazon S3	Anda kini dapat menyaksikan performa yang maksimal 10x lebih cepat untuk mengekspor data snapshot klaster DB ke S3 untuk klaster Aurora MySQL yang kompatibel dengan MySQL 5.7 dan 8.0. Untuk informasi selengkapnya, lihat Mengekspor data snapshot klaster DB ke Amazon S3.	20 Oktober 2022

Amazon Aurora mendukung penyiapan otomatis konektivitas antara klaster DB Aurora dan instans EC2	Anda dapat menggunakan AWS Management Console untuk mengatur konektivitas antara cluster Aurora DB yang ada dan instans EC2. Untuk informasi selengkapnya, lihat Menghubungkan instans EC2 dan klaster DB Aurora secara otomatis.	14 Oktober 2022
AWS Driver JDBC untuk PostgreSQL umumnya tersedia	Driver AWS JDBC untuk PostgreSQL adalah driver klien yang dirancang untuk Aurora PostgreSQL. Driver AWS JDBC untuk PostgreSQL sekarang tersedia secara umum. Untuk informasi selengkapnya, lihat Menghubungkan dengan Driver AWS JDBC untuk PostgreSQL.	6 Oktober 2022
Amazon Aurora mendukung peningkatan di tempat untuk Aurora MySQL yang kompatibel dengan MySQL 5.7	Anda dapat melakukan peningkatan di tempat untuk mengubah klaster Aurora MySQL yang kompatibel dengan MySQL 5.7 menjadi klaster Aurora MySQL yang kompatibel dengan MySQL 8.0. Untuk informasi selengkapnya, lihat Meningkatkan dari Aurora MySQL 2.x ke 3.x.	26 September 2022

Wawasan Performa menampilkan 25 kueri SQL teratas	Di dasbor Wawasan Performa, tab SQL Teratas menampilkan 25 kueri SQL yang paling berkontribusi pada beban DB. Untuk informasi selengkapnya, lihat Gambaran umum tab SQL Teratas.	13 September 2022
Aurora MySQL mendukung kelas instans DB yang baru	Anda kini dapat menggunakan kelas instans db.r6i untuk klaster DB Aurora MySQL. Untuk informasi selengkapnya, lihat Kelas instans DB.	13 September 2022
Amazon Aurora tersedia di Wilayah Timur Tengah (UAE)	Amazon Aurora kini tersedia di Wilayah Timur Tengah (UAE). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan.	30 Agustus 2022
Amazon Aurora mendukung penyiapan otomatis konektivitas dengan instans EC2	Saat membuat klaster Aurora DB, Anda dapat menggunakannya AWS Management Console untuk mengatur konektivitas antara instans Amazon Elastic Compute Cloud dan kluster DB baru. Untuk informasi selengkapnya, lihat Mengonfigurasi konektivitas jaringan otomatis dengan instans EC2.	22 Agustus 2022

Amazon Aurora mendukung mode tumpukan ganda	Klaster DB kini dapat berjalan dalam mode tumpukan ganda. Dalam mode tumpukan ganda, sumber daya dapat berkomunikasi dengan klaster DB melalui IPv4, IPv6, atau keduanya. Untuk informasi selengkapnya, lihat Pengalamatan IP Amazon Aurora.	17 Agustus 2022
Amazon Aurora mendukung peningkatan di tempat untuk Aurora Serverless v1 yang kompatibel dengan PostgreSQL	Anda dapat melakukan peningkatan di tempat untuk klaster Aurora Serverless v1 yang kompatibel dengan PostgreSQL 10 untuk mengubah klaster yang ada menjadi klaster Aurora Serverless v1 yang kompatibel dengan PostgreSQL 11. Untuk prosedur peningkatan di tempat, lihat Memodifikasi klaster DB Aurora Serverless v1.	8 Agustus 2022
Wawasan Performa mendukung Wilayah Asia Pasifik (Jakarta)	Sebelumnya, Anda tidak dapat menggunakan Wawasan Performa di Wilayah Asia Pasifik (Jakarta). Pembatasan ini telah dihapus. Untuk informasi selengkapnya, lihat Dukungan Wilayah AWS untuk Wawasan Performa.	21 Juli 2022

Amazon Aurora mendukung kelas instans DB yang baru	Anda kini dapat menggunakan kelas instans DB db.r6i untuk klaster DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Kelas instans DB.	14 Juli 2022
Wawasan Performa RDS mendukung periode retensi tambahan	Sebelumnya, Wawasan Performa hanya menawarkan dua periode retensi: 7 hari (default) atau 2 tahun (731 hari). Sekarang, jika Anda perlu mempertahankan data performa Anda selama lebih dari 7 hari, Anda dapat menentukan 1–24 bulan. Untuk informasi selengkapnya, lihat Harga dan retensi data untuk Wawasan Performa.	1 Juli 2022
Amazon Aurora mendukung peningkatan di tempat untuk Aurora Serverless v1 yang kompatibel dengan MySQL	Anda dapat melakukan peningkatan di tempat untuk klaster Aurora Serverless v1 yang kompatibel dengan MySQL 5.6 untuk mengubah klaster yang ada menjadi klaster Aurora Serverless v1 yang kompatibel dengan MySQL 5.7. Untuk prosedur peningkatan di tempat, lihat Memodifikasi klaster DB Aurora Serverless v1.	16 Juni 2022

Aurora mendukung menyalakan Amazon DevOps Guru di konsol RDS	Anda dapat mengaktifkan cakupan DevOps Guru untuk database Aurora Anda dari dalam konsol RDS. Untuk informasi selengkapnya, lihat Menyiapkan DevOps Guru untuk RDS.	9 Juni 2022
Amazon Aurora mendukung publikasi peristiwa ke topik Amazon SNS yang terenkripsi	Amazon Aurora kini dapat memublikasikan peristiwa ke topik Amazon Simple Notification Service (Amazon SNS) yang mengaktifkan enkripsi di sisi server (SSE), untuk perlindungan tambahan terhadap peristiwa yang berisi data sensitif. Untuk informasi selengkapnya, lihat Berlangganan notifikasi peristiwa Amazon RDS.	1 Juni 2022
Amazon RDS menerbitkan metrik penggunaan ke Amazon CloudWatch	`AWS/Usage` Namespace di Amazon CloudWatch menyertakan metrik penggunaan tingkat akun untuk kuota layanan Amazon RDS Anda. Untuk informasi selengkapnya, lihat metrik CloudWatch penggunaan Amazon untuk Amazon Aurora.	28 April 2022

Kumpulan hasil API Data dalam format JSON	Parameter opsional untuk fungsi `ExecuteStatement` menyebabkan hasil kueri ditetapkan untuk ditampilkan sebagai string dalam format JSON. Kumpulan hasil JSON mudah dan praktis untuk diubah menjadi struktur data dalam bahasa aplikasi Anda. Untuk informasi selengkapnya, lihat Memproses hasil kueri dalam format JSON.	27 April 2022
Amazon Aurora Serverless v2 kini tersedia secara umum	Amazon Aurora Serverless v2 tersedia secara umum untuk semua pengguna. Untuk informasi selengkapnya, lihat Menggunakan Aurora Serverless v2.	21 April 2022
Aurora MySQL mendukung cipher suite yang dapat dikonfigurasi	Dengan Aurora MySQL, Anda kini dapat menggunakan cipher suite yang dapat dikonfigurasi untuk mengontrol enkripsi koneksi yang diterima server basis data Anda. Untuk informasi selengkapnya, lihat Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora MySQL.	15 April 2022

Aurora PostgreSQL mendukung Proksi RDS dengan PostgreSQL 13	Anda kini dapat membuat Proksi RDS dengan klaster DB Aurora PostgreSQL 13. Untuk informasi selengkapnya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS.	4 April 2022
Catatan Rilis untuk Aurora PostgreSQL	Sekarang ada panduan terpisah untuk catatan rilis Amazon Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Catatan rilis Aurora PostgreSQL.	22 Maret 2022
Catatan Rilis untuk Aurora MySQL	Sekarang ada panduan terpisah untuk catatan rilis Amazon Aurora MySQL. Untuk informasi selengkapnya, lihat Catatan rilis Aurora MySQL.	22 Maret 2022
Aurora PostgreSQL mendukung peningkatan versi multi-utama	Anda kini dapat melakukan peningkatan versi klaster DB Aurora PostgreSQL di beberapa versi utama. Untuk informasi selengkapnya, lihat Cara melakukan peningkatan versi utama.	4 Maret 2022

Aurora PostgreSQL mendukung cipher suite yang dapat dikonfigurasi	Dengan Aurora PostgreSQL versi 11.8 dan yang lebih tinggi, Anda kini dapat menggunakan cipher suite yang dapat dikonfigurasi untuk mengontrol enkripsi koneksi yang diterima server basis data Anda. Untuk informasi tentang penggunaan cipher suite yang dapat dikonfigurasi dengan Aurora PostgreSQL, lihat Mengonfigurasi cipher suite untuk koneksi ke klaster DB Aurora PostgreSQL.	4 Maret 2022
AWS Driver JDBC untuk MySQL umumnya tersedia	Driver AWS JDBC untuk MySQL adalah driver klien yang dirancang untuk ketersediaan tinggi Aurora MySQL. Driver AWS JDBC untuk MySQL sekarang tersedia secara umum. Untuk informasi selengkapnya, lihat Menghubungkan dengan Amazon Web Services JDBC Driver for MySQL.	2 Maret 2022

Aurora PostgreSQL 13.5 mendukung Babelfish untuk Babelfish for Aurora PostgreSQL 1.1.0	Aurora PostgreSQL 13.5 mendukung Babelfish 1.1.0. Untuk daftar fitur baru, lihat 13.5. Untuk daftar fitur yang didukung oleh setiap rilis Babelfish, lihat Fungsionalitas yang didukung di Babelfish menurut versi. Untuk informasi penggunaan, lihat Bekerja dengan Babelfish for Aurora PostgreSQL.	28 Februari 2022
Amazon Aurora mendukung Aliran Aktivitas Basis Data di Wilayah Asia Pasifik (Jakarta)	Untuk informasi selengkapnya, lihat Support Wilayah AWS untuk aliran aktivitas database.	16 Februari 2022
Wawasan Performa mendukung operasi API yang baru	Wawasan Performa kini mendukung operasi API berikut: `GetResourceMetadata` , `ListAvailableResourceDimensions` , dan `ListAvailableResourceMetrics` . Untuk informasi selengkapnya, lihat Mengambil metrik dengan API Wawasan Performa dalam panduan ini dan Referensi API Wawasan Performa Amazon RDS.	12 Januari 2022

Proksi Amazon RDS mendukung peristiwa	Proxy RDS sekarang menghasilkan acara yang dapat Anda berlangganan dan lihat di CloudWatch Acara atau konfigurasikan untuk dikirim ke Amazon EventBridge. Untuk informasi selengkapnya, lihat Menggunakan peristiwa Proksi RDS.	11 Januari 2022
Proxy RDS tersedia dalam tambahan Wilayah AWS	Proksi RDS sekarang tersedia di Wilayah berikut: Afrika (Cape Town), Asia Pasifik (Hong Kong), Asia Pasifik (Osaka), Eropa (Milan), Eropa (Paris), Eropa (Stockholm), Timur Tengah (Bahrain), dan Amerika Selatan (Sao Paulo). Untuk informasi selengkap nya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS.	5 Januari 2022
Amazon Aurora tersedia di Wilayah Asia Pasifik (Jakarta)	Amazon Aurora kini tersedia di Wilayah Asia Pasifik (Jakarta) . Untuk informasi selengkap nya, lihat Wilayah dan Zona Ketersediaan.	13 Desember 2021

DevOpsGuru untuk Amazon RDS memberikan wawasan dan rekomendasi terperinci untuk Amazon Aurora	DevOpsGuru untuk RDS menambang Performance Insights untuk data terkait kinerja. Dengan menggunakan data ini, layanan menganalisis performa instans DB Amazon Aurora dan dapat membantu Anda mengatasi masalah performa. Untuk mempelaja ri lebih lanjut, lihat Menganali sis anomali kinerja dengan DevOps Guru untuk RDS dalam panduan ini dan lihat Ikhtisar DevOps Guru untuk RDS di Panduan Pengguna Amazon DevOps Guru.	1 Desember 2021
Aurora PostgreSQL mendukung Proksi RDS dengan PostgreSQL 12	Anda kini dapat membuat Proksi RDS dengan klaster basis data Aurora PostgreSQL 12. Untuk informasi selengkap nya tentang Proksi RDS, lihat Menggunakan Proksi Amazon RDS.	22 November 2021
Aurora mendukung kelas instance AWS Graviton2 untuk Aliran Aktivitas Database	Anda dapat menggunakan aliran aktivitas basis data dengan kelas instans db.r6g untuk Aurora MySQL dan Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Kelas instans DB yang didukung.	3 November 2021

Dukungan Amazon Aurora untuk lintas akun AWS KMS keys	Anda dapat menggunakan kunci KMS dari yang berbeda Akun AWS untuk enkripsi saat mengekspor snapshot DB ke Amazon S3. Untuk informasi selengkapnya, lihat Mengekspor data snapshot DB ke Amazon S3.	3 November 2021
Amazon Aurora mendukung Babelfish for Aurora PostgreSQL	Babelfish for Aurora PostgreSQL memperlua s basis data Edisi yang Kompatibel dengan Amazon Aurora PostgreSQL Anda dengan kemampuan menerima koneksi basis data dari klien Server SQL Microsoft. Untuk informasi selengkapnya, lihat Bekerja dengan Babelfish for Aurora PostgreSQL.	28 Oktober 2021
Aurora Serverless v1 mungkin memerlukan SSL untuk koneksi	Parameter klaster Aurora `force_ssl` untuk PostgreSQL dan `require_secure_transport` untuk MySQL kini didukung untuk Aurora Serverless v1. Untuk informasi selengkapnya, lihat Menggunakan TLS/SSL dengan Aurora Serverless v1.	26 Oktober 2021

Amazon Aurora mendukung Performance Insights sebagai tambahan Wilayah AWS	Wawasan Performa tersedia di Wilayah Timur Tengah (Bahrain), Afrika (Cape Town), Eropa (Milan), dan Asia Pasifik (Osaka). Untuk informasi selengkapnya, lihat Dukungan Wilayah AWS untuk Wawasan Performa.	5 Oktober 2021
Batas waktu penskalaan otomatis yang dapat dikonfigurasi untuk Aurora Serverless v1	Anda dapat memilih durasi tunggu Aurora Serverless v1 sebelum menemukan titik penskalaan otomatis. Jika tidak ditemukan titik penskalaan otomatis selama periode tersebut, Aurora Serverless v1 membatalkan peristiwa penskalaan atau memaksa perubahan kapasitas, bergantung pada tindakan batas waktu yang Anda pilih. Untuk informasi selengkapnya, lihat Penskalaan otomatis untuk v1Aurora Serverless.	10 September 2021

Aurora mendukung kelas instans X2g dan T4g	Aurora MySQL dan Aurora PostgreSQL sekarang dapat menggunakan kelas instans X2g dan T4g. Kelas instans yang dapat Anda gunakan bergantung pada versi Aurora MySQL atau Aurora PostgreSQL. Untuk informasi selengkapnya tentang jenis instans yang didukung, lihat Kelas instans DB.	10 September 2021
Amazon RDS mendukung Proksi RDS di VPC bersama	Anda kini dapat membuat Proksi RDS di cloud privat virtual (VPC) bersama. Untuk informasi selengkapnya tentang Proksi RDS, lihat "Mengelola Koneksi dengan Proksi Amazon RDS" dalam Panduan Pengguna Amazon RDS atau Panduan Pengguna Aurora.	6 Agustus 2021
Halaman kebijakan versi Aurora	Panduan Pengguna Amazon Aurora sekarang menyertakan bagian berisi informasi umum tentang versi Aurora dan kebijakan terkait. Untuk detailnya, lihat versi Amazon Aurora.	14 Juli 2021

Kecualikan peristiwa API Data dari AWS CloudTrail jejak	Anda dapat mengecualikan peristiwa Data API dari CloudTrail jejak. Untuk informasi selengkapnya, lihat Mengecualikan peristiwa API Data dari AWS CloudTrail jejak.	2 Juli 2021
Edisi yang Kompatibel dengan Amazon Aurora PostgreSQL mendukung ekstensi tambahan	Ekstensi yang baru didukung termasuk pg_bigm, pg_cron, pg_partman, dan pg_procta b. Untuk informasi selengkapnya, lihat Versi ekstensi untuk Edisi yang Kompatibel dengan Amazon Aurora PostgreSQL.	17 Juni 2021
Mengklonakan klaster Aurora Serverless	Anda kini dapat membuat klaster klona yang Aurora Serverless. Untuk informasi selengkapnya tentang pengklonaan, lihat Mengklonakan volume untuk klaster DB Aurora.	16 Juni 2021
Basis data global Aurora tersedia di Wilayah Tiongkok (Beijing) dan Tiongkok (Ningxia)	Anda kini dapat membuat basis data global Aurora di Wilayah Tiongkok (Beijing) dan Tiongkok (Ningxia). Untuk informasi tentang basis data global Aurora, lihat Bekerja dengan basis data global Amazon Aurora.	19 Mei 2021

Dukungan FIPS 140-2 untuk API Data	API Data mendukung Publikasi Federal Information Processing Standard 140-2 (FIPS 140-2) untuk koneksi SSL/TLS. Untuk informasi selengkapnya, lihat Ketersediaan API Data.	14 Mei 2021
AWS Driver JDBC untuk PostgreSQL (pratinjau)	Driver AWS JDBC untuk PostgreSQL, sekarang tersedia dalam pratinjau , adalah driver klien yang dirancang untuk ketersediaan tinggi Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menghubungkan Driver JDBC Layanan Web Amazon untuk PostgreSQL (pratinjau).	27 April 2021
Data API tersedia dalam tambahan Wilayah AWS	API Data kini tersedia di Wilayah Asia Pasifik (Seoul) dan Kanada (Pusat). Untuk informasi selengkapnya, lihat Ketersediaan API Data.	9 April 2021
Amazon Aurora mendukung kelas instans DB Graviton2	Anda kini dapat menggunakan kelas instans DB Graviton2 db.r6g.x untuk membuat klaster DB yang menjalankan MySQL atau PostgreSQL. Untuk informasi selengkapnya, lihat Jenis kelas instans DB.	12 Maret 2021

Peningkatan titik akhir Proksi RDS	Anda dapat membuat titik akhir tambahan yang terkait dengan setiap Proksi RDS. Membuat titik akhir dalam VPC yang berbeda memungkin kan akses lintas VPC untuk proksi tersebut. Proksi untuk klaster Aurora MySQL juga dapat memiliki titik akhir hanya baca. Titik akhir pembaca ini terhubung ke instans DB pembaca di dalam klaster dan dapat meningkatkan skalabili tas dan ketersediaan baca untuk aplikasi intensif kueri. Untuk informasi selengkap nya tentang Proksi RDS, lihat “Mengelola Koneksi dengan Proksi Amazon RDS” dalam Panduan Pengguna Amazon RDS atau Panduan Pengguna Aurora.	8 Maret 2021
Amazon Aurora tersedia di Wilayah Asia Pasifik (Osaka)	Amazon Aurora kini tersedia di Wilayah Asia Pasifik (Osaka). Untuk informasi selengkap nya, lihat Wilayah dan Zona Ketersediaan.	1 Maret 2021
Aurora PostgreSQL mendukung pengaktifan autentikasi IAM dan Kerberos pada klaster DB yang sama	Aurora PostgreSQL kini mendukung pengaktifan autentikasi IAM dan Kerberos pada klaster DB yang sama. Untuk informasi selengkapnya, lihat Autentikasi basis data dengan Amazon Aurora.	24 Februari 2021

Basis data global Aurora kini mendukung failover terencana yang terkelola	Basis data global Aurora kini mendukung failover terencana yang terkelola sehingga Anda dapat lebih mudah mengubah Wilayah AWS utama dari basis data global Aurora Anda. Anda dapat menggunakan failover terencana yang terkelola dengan basis data global Aurora yang berkondisi baik saja. Untuk mempelajari lebih lanjut, lihat Pemulihan bencana dan basis data global Amazon Aurora. Untuk informasi referensi, lihat `FailoverGlobalCluster` dalam Referensi API Amazon RDS.	11 Februari 2021
API Data untuk Aurora Serverless kini mendukung lebih banyak jenis data	Dengan API Data untuk Aurora Serverless, Anda kini dapat menggunakan jenis data `UUID` dan `JSON` sebagai input ke basis data Anda. Termasuk dengan API Data untuk Aurora Serverless, Anda kini dapat menampilkan nilai jenis `LONG` dari basis data Anda sebagai nilai `STRING`. Untuk mempelajari lebih lanjut, lihat Memanggil API Data. Untuk informasi referensi tentang jenis data yang didukung, lihat `SqlParameter` dalam Referensi API Layanan Data Amazon RDS.	2 Februari 2021

Aurora PostgreSQL mendukung peningkatan versi utama ke PostgreSQL 12	Dengan Aurora PostgreSQL, Anda kini dapat meningkatkan mesin DB ke versi utama 12. Untuk informasi selengkapnya, lihat Meningkatkan mesin DB PostgreSQL untuk Aurora PostgreSQL.	28 Januari 2021
Aurora MySQL mendukung peningkatan di tempat	Anda dapat meningkatkan klaster Aurora MySQL 1.x ke Aurora MySQL 2.x, yang mempertahankan instans DB, titik akhir, dan sebagainya dari klaster asli. Teknik peningkatan di tempat ini menghindari ketidaknyamanan penyiapan klaster baru dengan memulihkan snapshot. Hal ini juga menghindari overhead untuk penyalinan semua data tabel ke klaster baru. Untuk informasi selengkapnya, lihat Meningkatkan versi utama klaster DB Aurora MySQL dari 1.x ke 2.x.	11 Januari 2021
AWS Driver JDBC untuk MySQL (pratinjau)	Driver AWS JDBC untuk MySQL, sekarang tersedia dalam pratinjau, adalah driver klien yang dirancang untuk ketersediaan tinggi Aurora MySQL. Untuk informasi selengkapnya, lihat Menghubungkan Driver JDBC Layanan Web Amazon untuk MySQL (pratinjau).	7 Januari 2021

Aurora mendukung aliran aktivitas basis data pada klaster sekunder basis data global	Anda dapat memulai aliran aktivitas basis data pada klaster primer atau sekunder dari Aurora PostgreSQL atau Aurora MySQL. Untuk versi mesin yang didukung, lihat Batasan basis data global Aurora.	22 Desember 2020
Klaster multi-master dengan 4 instans DB	Jumlah maksimal instans DB pada klaster multi-master Aurora MySQL sekarang adalah empat. Sebelumnya, jumlah maksimalnya adalah dua instans DB. Untuk informasi selengkapnya, lihat Bekerja dengan Klaster Multi-Master Aurora.	17 Desember 2020
Aurora PostgreSQL mendukung fungsi AWS Lambda	Anda sekarang dapat memanggil AWS Lambda fungsi untuk cluster Aurora PostgreSQL DB Anda. Untuk informasi selengkapnya, lihat Menginvokasi fungsi Lambda dari klaster DB Aurora PostgreSQL.	11 Desember 2020
Amazon Aurora mendukung kelas instans DB Graviton2 dalam pratinjau	Anda kini dapat menggunakan kelas instans DB Graviton2 db.r6g.x dalam pratinjau untuk membuat klaster DB yang menjalankan MySQL atau PostgreSQL. Untuk informasi selengkapnya, lihat Jenis kelas instans DB.	11 Desember 2020

Amazon Aurora Serverless v2 kini tersedia dalam pratinjau.	Amazon Aurora Serverless v2 tersedia dalam pratinjau. Untuk menggunakan Amazon Aurora Serverless v2, ajukan permohonan akses. Untuk informasi selengkapnya, lihat halaman Aurora Serverless v2.	1 Desember 2020

Aurora PostgreSQL kini tersedia untuk Aurora Serverless di lebih banyak Wilayah AWS.	Aurora PostgreSQL kini tersedia untuk Aurora Serverless di lebih banyak Wilayah AWS. Anda sekarang dapat memilih untuk menjalankan Aurora PostgreSQL Serverless v1 penawaran Wilayah AWS yang samaAurora MySQL Serverless v1. Tambahan Wilayah AWS dengan Aurora Serverless dukungan termasuk AS Barat (Californ ia N.), Asia Pasifik (Singapur a) Asia Pasifik (Sydney) Asia Pasifik (Seoul) Asia Pasifik (Mumbai) Kanada (Tengah) Eropa (London) dan Eropa (Paris). Untuk daftar berisi semua Wilayah dan mesin DB Aurora yang didukung untuk Aurora Serverless, lihat Aurora Nirserver. API Data Amazon RDS untuk Aurora Serverless juga tersedia di Wilayah AWS yang sama ini. Untuk daftar berisi semua Wilayah dengan dukungan untuk API Data untuk Aurora Serverless, lihat API Data untuk Aurora Nirserver	24 November 2020

Wawasan Performa Amazon RDS memperkenalkan dimensi baru	Anda dapat mengelompokkan beban basis data berdasarkan grup dimensi untuk basis data, aplikasi (PostgreSQL), dan jenis sesi (PostgreSQL). Amazon RDS juga mendukung dimensi db.name, db.application.name (PostgreSQL), dan db.session_type.name (PostgreSQL). Untuk informasi selengkapnya, lihat Tabel beban teratas.	24 November 2020
Aurora Serverless mendukung Aurora PostgreSQL versi 10.12	Aurora PostgreSQL untuk Aurora Serverless telah ditingkatkan ke Aurora PostgreSQL versi 10.12 di seluruh Wilayah AWS tempat Aurora PostgreSQL untuk Aurora Serverless didukung. Untuk informasi selengkapnya, lihat Aurora Serverless.	4 November 2020
API Data kini mendukung otorisasi berbasis tag	API Data kini mendukung otorisasi berbasis tag. Jika Anda sudah melabeli sumber daya klaster RDS dengan tag, Anda dapat menggunakan tag ini dalam pernyataan kebijakan Anda untuk mengontrol akses melalui API Data. Untuk informasi selengkapnya, lihat Mengotorisasi akses ke API Data.	27 Oktober 2020

Amazon Aurora memperluas dukungan untuk mengekspor snapshot ke Amazon S3	Anda kini dapat mengekspor data snapshot DB ke Amazon S3 di semua Wilayah AWS komersial. Untuk informasi selengkapnya, lihat Mengekspor data snapshot DB ke Amazon S3.	22 Oktober 2020
Basis data global Aurora mendukung pengklonaan	Anda kini dapat menciptakan klona klaster DB primer dan sekunder dari basis data global Aurora Anda. Anda dapat melakukannya dengan menggunakan AWS Management Console dan memilih opsi menu Create clone. Anda juga dapat menggunakan AWS CLI dan menjalankan `restore-db-cluster-to-point-in-time` perintah dengan `--restore-type copy-on-write` opsi. Menggunakan AWS Management Console atau AWS CLI, Anda juga dapat mengkloning cluster DB dari database global Aurora Anda di seluruh akun. AWS Untuk informasi selengkapnya tentang pengklonaan, lihat Mengklonakan volume klaster DB Aurora.	19 Oktober 2020

Amazon Aurora mendukung pengubahan ukuran dinamis untuk volume klaster	Mulai dari Aurora MySQL 1.23 dan 2.09, dan Aurora PostgreSQL 3.3.0 dan Aurora PostgreSQL 2.6.0, Aurora mengurangi ukuran volume klaster setelah Anda menghapus data melalui pengoperasian seperti `DROP TABLE`. Untuk memanfaatkan peningkatan ini, tingkatkan ke salah satu versi yang sesuai, bergantung pada mesin basis data yang digunakan klaster Anda. Untuk informasi tentang fitur ini dan cara memeriksa ruang penyimpanan yang masih ada dan telah digunakan untuk klaster Aurora, lihat Mengelola Performa dan Menskalakan Klaster DB Aurora.	13 Oktober 2020
Amazon Aurora mendukung ukuran volume hingga 128 TiB	Volume klaster Aurora yang baru dan lama kini bertambah ke ukuran maksimum 128 tebibyte (TiB). Untuk informasi selengkapnya, lihat Cara penyimpanan Aurora bertambah.	22 September 2020

Aurora PostgreSQL mendukung kelas instans DB db.r5 dan db.t3 untuk Wilayah Tiongkok (Ningxia)	Anda kini dapat membuat klaster DB Aurora PostgreSQL di Wilayah Tiongkok (Ningxia) yang menggunakan kelas instans DB db.r5 dan db.t3. Untuk informasi selengkapnya, lihat Kelas instans DB.	3 September 2020

Peningkatan kueri paralel Aurora	Mulai dengan Aurora MySQL 2.09 dan 1.23, Anda dapat memanfaatkan peningkatan pada fitur kueri paralel. Pembuatan klaster kueri paralel tidak lagi membutuhkan mode mesin khusus. Anda kini dapat mengaktifkan dan menonaktifkan kueri paralel menggunakan opsi konfigurasi `aurora_parallel_query` untuk klaster apa pun yang telah disediakan yang menjalankan versi Aurora MySQL yang kompatibel. Anda dapat meningkatkan klaster yang sudah ada ke versi Aurora MySQL yang kompatibel dan menggunakan kueri paralel, bukan membuat klaster baru serta mengimpor data ke klaster. Anda dapat menggunakan Wawasan Performa untuk klaster kueri paralel. Anda dapat menghentikan dan memulai klaster kueri paralel. Anda dapat membuat klaster kueri paralel Aurora yang kompatibel dengan MySQL 5.7. Kueri paralel berfungsi untuk tabel yang menggunakan format baris `DYNAMIC`. Cluster query paralel dapat menggunakan otentikasi AWS Identity and Access Management (IAM).	2 September 2020

	Instans DB pembaca dalam klaster kueri paralel dapat memanfaatkan tingkat isolasi `READ COMMITTED` . Anda kini juga dapat membuat klaster kueri paralel di Wilayah AWS tambahan. Untuk informasi selengkapnya tentang fitur kueri paralel dan peningkatan ini, lihat Bekerja dengan kueri paralel untuk Aurora MySQL.	
Mengubah ke parameter Aurora MySQL binlog_rows_query_log_events	Anda kini dapat mengubah nilai parameter konfigurasi Aurora MySQL `binlog_rows_query_log_events` . Sebelumnya, parameter ini tidak dapat diubah.	26 Agustus 2020

Dukungan untuk peningkatan otomatis versi minor untuk Aurora MySQL	Dengan Aurora MySQL, pengaturan Aktifkan peningkatan otomatis versi minor kini akan diterapkan saat Anda menentukannya untuk klaster DB Aurora MySQL. Saat Anda mengaktifkan peningkatan otomatis versi minor, Aurora secara otomatis meningkatkan ke versi minor baru saat versi baru tersebut dirilis. Peningkatan otomatis terjadi selama periode pemeliharaan untuk basis data. Untuk Aurora MySQL, fitur ini hanya berlaku untuk Aurora MySQL versi 2, yang kompatibel dengan MySQL 5.7. Pertama-tama, prosedur peningkatan otomatis akan mengarahkan klaster DB Aurora MySQL ke versi 2.07.2. Untuk informasi selengkapnya tentang cara kerja fitur ini pada Aurora MySQL, lihat Peningkatan dan Patch Basis Data untuk Amazon Aurora MySQL.	3 Agustus 2020
Aurora PostgreSQL mendukung peningkatan versi utama ke PostgreSQL versi 11	Dengan Aurora PostgreSQL, Anda kini dapat meningkatkan mesin DB ke versi utama 11. Untuk informasi selengkapnya, lihat Meningkatkan mesin DB PostgreSQL untuk Aurora PostgreSQL.	28 Juli 2020

Amazon Aurora mendukung AWS PrivateLink	Amazon Aurora sekarang mendukung pembuatan titik akhir Amazon VPC untuk panggilan Amazon RDS API untuk menjaga lalu lintas antara aplikasi dan Aurora di jaringan. AWS Untuk informasi selengkapnya, lihat titik akhir VPC antarmuka dan Amazon Aurora (AWS PrivateLink).	9 Juli 2020
Proksi RDS tersedia secara umum	Proksi RDS kini tersedia secara umum. Anda dapat menggunakan Proksi RDS dengan RDS untuk MySQL, Aurora MySQL, RDS untuk PostgreSQL, dan Aurora PostgreSQL untuk beban kerja produksi. Untuk informasi selengkapnya tentang Proksi RDS, lihat "Mengelola Koneksi dengan Proksi Amazon RDS" dalam Panduan Pengguna Amazon RDS atau Panduan Pengguna Aurora.	30 Juni 2020

Penerusan tulis basis data global Aurora	Anda kini dapat mengaktif kan kemampuan tulis pada klaster sekunder di basis data global. Dengan penerusan tulis, Anda mengeluarkan pernyataan DML di klaster sekunder, Aurora akan meneruskan tulisan tersebut ke klaster primer, dan data yang diperbarui akan direplika si ke semua klaster sekunder. Untuk informasi selengkap nya, lihat Menulis penerusan untuk sekunder Wilayah AWS dengan database global Aurora.	18 Juni 2020
Aurora mendukung integrasi dengan AWS Backup	Anda dapat menggunakan AWS Backup untuk mengelola backup cluster Aurora DB. Untuk informasi selengkap nya, lihat Ikhtisar pencadang an dan pemulihan klaster DB Aurora.	10 Juni 2020
Aurora PostgreSQL mendukung kelas instans DB db.t3.large	Anda kini dapat membuat klaster DB Aurora PostgreSQ L yang menggunakan kelas instans DB db.t3.large. Untuk informasi selengkapnya, lihat Kelas instans DB.	5 Juni 2020

Basis data global Aurora mendukung PostgreSQL versi 11.7 dan sasaran titik pemulihan (RPO) yang terkelola	Anda kini dapat membuat basis data global Aurora untuk mesin basis data PostgreSQL versi 11.7. Anda juga dapat mengelola bagaimana basis data global PostgreSQL dipulihkan dari kegagalan menggunakan sasaran titik pemulihan (RPO). Untuk informasi selengkapnya, lihat Pemulihan Bencana Lintas Wilayah untuk basis data global Aurora.	4 Juni 2020
Aurora MySQL mendukung pemantauan basis data dengan aliran aktivitas basis data	Aurora MySQL sekarang menyertakan aliran aktivitas database, yang menyediakan aliran near-real-time data aktivitas database dalam database relasional Anda. Untuk informasi selengkapnya, lihat Menggunakan aliran aktivitas basis data.	2 Juni 2020
Editor kueri tersedia dalam tambahan Wilayah AWS	Editor kueri untuk Aurora Serverless sekarang tersedia dalam tambahan. Wilayah AWS Untuk informasi selengkapnya, lihat Ketersediaan editor kueri.	28 Mei 2020
Data API tersedia dalam tambahan Wilayah AWS	Data API sekarang tersedia dalam tambahan Wilayah AWS. Untuk informasi selengkapnya, lihat Ketersediaan API Data.	28 Mei 2020

Proksi RDS tersedia di Wilayah Kanada (Pusat)	Anda kini dapat menggunakan pratinjau Proksi RDS di Wilayah Kanada (Pusat). Untuk informasi selengkapnya tentang Proksi RDS, lihat Mengelola koneksi dengan proksi Amazon RDS (pratinjau).	28 Mei 2020
Basis data global Aurora dan replika baca lintas wilayah	Untuk basis data global Aurora, Anda kini dapat membuat replika baca lintas Wilayah Aurora MySQL dari klaster primer di wilayah yang sama seperti klaster sekunder. Untuk informasi selengkapnya tentang Basis Data Global Aurora dan replika baca lintas Wilayah, lihat Bekerja dengan basis data global Amazon Aurora dan Mereplikasi DB Amazon Aurora MySQL.	18 Mei 2020

Proxy RDS tersedia di lebih banyak Wilayah AWS	Anda kini dapat menggunakan pratinjau Proksi RDS di Wilayah AS Barat (California Utara), Wilayah Eropa (London), Wilayah Eropa (Frankfurt), Wilayah Asia Pasifik (Seoul), Wilayah Asia Pasifik (Mumbai), Wilayah Asia Pasifik (Singapura), dan Wilayah Asia Pasifik (Sydney). Untuk informasi selengkapnya tentang Proksi RDS, lihat Mengelola koneksi dengan proksi Amazon RDS (pratinjau).	13 Mei, 2020
Edisi yang Kompatibel dengan Aurora PostgreSQL mendukung direktori aktif Microsoft on-premise atau yang di-host sendiri	Anda kini dapat menggunakan Direktori Aktif on-premise atau yang di-host sendiri untuk autentikasi pengguna Kerberos saat mereka terhubung ke klaster DB Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Menggunakan autentikasi Kerberos dengan Aurora PostgreSQL.	7 Mei 2020

Cluster multi-master Aurora MySQL tersedia di lebih banyak Wilayah AWS	Anda kini dapat membuat klaster multi-master Aurora di Wilayah Asia Pasifik (Seoul), Wilayah Asia Pasifik (Tokyo), Wilayah Asia Pasifik (Mumbai), dan Wilayah Eropa (Frankfurt). Untuk informasi selengkapnya tentang klaster multi-master, lihat Bekerja dengan klaster multi-master Aurora.	7 Mei 2020
Wawasan Performa mendukung analisis statistik kueri Aurora MySQL yang sedang berjalan	Anda kini dapat menganalisis statistik kueri yang sedang berjalan dengan Wawasan Performa untuk instans DB Aurora MySQL. Untuk informasi selengkapnya, lihat Menganalisis statistik kueri yang sedang berjalan.	5 Mei 2020
Pustaka klien Java untuk API Data tersedia secara umum	Pustaka klien Java untuk API Data kini tersedia secara umum. Anda dapat mengunduh dan menggunakan pustaka klien Java untuk API Data. Pustaka ini memungkinkan Anda untuk memetakan kelas sisi klien terhadap permintaan dan respons dari API Data. Untuk informasi selengkapnya, lihat Menggunakan pustaka klien Java untuk API Data.	30 April 2020

Amazon Aurora tersedia di Wilayah Eropa (Milan)	Amazon Aurora kini tersedia di Wilayah Eropa (Milan). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan.	28 April 2020
Amazon Aurora tersedia di Wilayah Eropa (Milan)	Amazon Aurora kini tersedia di Wilayah Eropa (Milan). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan.	27 April 2020
Amazon Aurora tersedia di Wilayah Afrika (Cape Town)	Amazon Aurora kini tersedia di Wilayah Afrika (Cape Town). Untuk informasi selengkapnya, lihat Wilayah dan Zona Ketersediaan.	22 April 2020
Aurora PostgreSQL kini mendukung kelas instans DB db.r5.16xlarge dan db.r5.8xlarge	Anda kini dapat membuat klaster DB Aurora PostgreSQL yang menjalankan PostgreSQL dan menggunakan kelas instans DB db.r5.16xlarge dan db.r5.8xlarge. Untuk informasi selengkapnya, lihat Spesifikasi perangkat keras untuk kelas instans DB untuk Aurora.	8 April 2020

Proksi Amazon RDS untuk PostgreSQL	Proksi Amazon RDS kini tersedia untuk PostgreSQL. Anda dapat menggunakan Proksi RDS untuk mengurangi overhead manajemen koneksi pada klaster Anda dan juga kemungkinan kesalahan "terlalu banyak koneksi". Proksi RDS saat ini dalam pratinjau publik untuk PostgreSQL. Untuk informasi selengkapnya, lihat Mengelola koneksi dengan proksi Amazon RDS (pratinjau).	8 April 2020
Basis data global Aurora kini mendukung Aurora PostgreSQL	Anda kini dapat membuat basis data global Aurora untuk mesin basis data PostgreSQL. Basis data global Aurora mencakup beberapa Wilayah AWS, memungkinkan pembacaan global latensi rendah dan pemulihan bencana dari pemadaman di seluruh wilayah. Untuk informasi selengkapnya, lihat Bekerja dengan basis data global Amazon Aurora.	10 Maret 2020

Dukungan untuk peningkatan versi utama untuk Aurora PostgreSQL	Dengan Aurora PostgreSQL, Anda kini dapat meningkatkan mesin DB ke versi utama. Dengan demikian, Anda dapat langsung beralih ke versi utama terbaru saat meningkatkan versi mesin PostgreSQL tertentu. Untuk informasi selengkapnya, lihat Meningkatkan mesin DB PostgreSQL untuk Aurora PostgreSQL.	4 Maret 2020
Aurora PostgreSQL mendukung autentikasi Kerberos	Anda kini dapat menggunakan autentikasi Kerberos untuk mengautentikasi pengguna saat mereka terhubung ke klaster DB Aurora PostgreSQL Anda. Untuk informasi selengkapnya, lihat Menggunakan autentikasi Kerberos dengan Aurora PostgreSQL.	28 Februari 2020
Data API mendukung AWS PrivateLink	Data API sekarang mendukung pembuatan titik akhir Amazon VPC untuk panggilan API Data untuk menjaga lalu lintas antara aplikasi dan API Data di jaringan. AWS Untuk informasi selengkapnya, lihat Membuat titik akhir VPC Amazon (AWS PrivateLink) untuk API Data.	6 Februari 2020

Dukungan machine learning Aurora di Aurora PostgreSQL	Ekstensi `aws_ml` Aurora PostgreSQL menyediakan fungsi yang Anda gunakan dalam kueri database Anda untuk memanggil Amazon Comprehend untuk analisis sentimen dan menjalankan model pembelajaran mesin Anda sendiri. SageMaker Untuk informasi selengkap nya, lihat Menggunakan kemampuan machine learning (ML) dengan Aurora.	5 Februari 2020
Aurora PostgreSQL mendukung pengeksporan data ke Amazon S3	Anda dapat membuat kueri data dari klaster DB Aurora PostgreSQL dan mengekspornya langsung ke file yang tersimpan dalam bucket Amazon S3. Untuk informasi selengkapnya, lihat Mengekspor data dari klaster DB Aurora PostgreSQL ke Amazon S3.	5 Februari 2020
Dukungan untuk mengekspor data snapshot DB ke Amazon S3	Amazon Aurora mendukung pengeksporan data snapshot DB ke Amazon S3 for MySQL dan PostgreSQL. Untuk informasi selengkapnya, lihat Mengekspor data snapshot DB ke Amazon S3.	9 Januari 2020

Catatan rilis Aurora MySQL dalam riwayat dokumen	Bagian ini sekarang menyertakan entri riwayat untuk catatan rilis Edisi yang Kompatibel dengan Aurora MySQL untuk versi yang dirilis setelah 31 Agustus 2018. Untuk catatan rilis lengkap dari versi tertentu, pilih tautan di kolom pertama entri riwayat.	13 Desember 2019
Proksi Amazon RDS	Anda dapat mengurangi overhead manajemen koneksi pada klaster, dan mengurangi kemungkinan kesalahan "terlalu banyak koneksi", dengan menggunakan Proksi Amazon RDS. Anda mengaitkan setiap proksi dengan instans DB RDS atau klaster DB Aurora. Kemudian, Anda menggunakan titik akhir proksi dalam string koneksi untuk aplikasi Anda. Proksi Amazon RDS saat ini dalam status pratinjau publik. Proksi ini mendukung mesin basis data Aurora MySQL. Untuk informasi selengkapnya, lihat Mengelola koneksi dengan proksi Amazon RDS (pratinjau).	3 Desember 2019

API Data untuk Aurora Serverless v1 mendukung petunjuk pemetaan jenis data	Anda kini dapat menggunakan petunjuk untuk menginstruksikan API Data untuk Aurora Serverless v1 agar mengirim nilai `String` ke basis data sebagai jenis yang berbeda. Untuk informasi selengkapnya, Memanggil API Data.	26 November 2019
API Data untuk Aurora Serverless v1 mendukung pustaka klien Java (pratinjau)	Anda dapat mengunduh dan menggunakan pustaka klien Java untuk API Data. Pustaka ini memungkinkan Anda untuk memetakan kelas sisi klien terhadap permintaan dan respons dari API Data. Untuk informasi selengkapnya, lihat Menggunakan pustaka klien Java untuk API Data.	26 November 2019
Aurora PostgreSQL memenuhi syarat FedRAMP HIGH	Aurora PostgreSQL memenuhi syarat FedRAMP HIGH. Untuk detail tentang AWS dan upaya kepatuhan, lihat AWS layanan dalam cakupan berdasarkan program kepatuhan.	26 November 2019

Tingkat isolasi READ COMMITTED diaktifkan untuk replika Amazon Aurora MySQL	Anda kini dapat mengaktifkan tingkat isolasi `READ COMMITTED` pada Replika Aurora MySQL. Untuk melakukan hal ini, Anda harus mengaktifkan pengaturan konfigurasi `aurora_read_replica_read_committed_isolation_enabled` di tingkat sesi. Menggunakan tingkat isolasi `READ COMMITTED` untuk kueri yang membutuhkan waktu lama pada klaster OLTP dapat membantu mengatasi masalah terkait panjang daftar riwayat. Sebelum mengaktifkan pengaturan ini, pastikan untuk memahami perbedaan perilaku isolasi pada Replika Aurora dengan penerapan MySQL standar dari `READ COMMITTED`. Untuk informasi selengkapnya, lihat Tingkat isolasi Aurora MySQL.	25 November 2019
Wawasan Performa mendukung analisis statistik kueri Aurora PostgreSQL yang sedang berjalan	Anda kini dapat menganalisis statistik kueri yang sedang berjalan dengan Wawasan Performa untuk instans DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menganalisis statistik kueri yang sedang berjalan.	25 November 2019

Ada lebih banyak klaster dalam basis data global Aurora	Anda kini dapat menambahkan beberapa wilayah sekunder ke basis data global Aurora. Anda dapat memanfaatkan pembacaan global latensi rendah dan pemulihan bencana di area geografis yang lebih luas. Untuk informasi selengkapnya tentang basis data global Aurora, lihat Bekerja dengan basis data global Amazon Aurora.	25 November 2019
Dukungan machine learning Aurora di Aurora MySQL	Di Aurora MySQL 2.07 dan lebih tinggi, Anda dapat menghubungi Amazon Comprehend untuk analisis sentimen dan untuk berbagai algoritma pembelajaran mesin. SageMaker Anda dapat menggunakan hasilnya secara langsung dalam aplikasi basis data dengan menyematkan panggilan ke fungsi yang tersimpan dalam kueri Anda. Untuk informasi selengkapnya, lihat Menggunakan kemampuan machine learning (ML) dengan Aurora.	25 November 2019

Basis data global Aurora tidak lagi membutuhkan pengaturan mode mesin	Anda tidak perlu lagi menentukan `--engine-mode=global` saat membuat klaster yang ditujukan untuk menjadi bagian dari basis data global Aurora. Semua klaster Aurora yang memenuhi persyaratan kompatibilitas layak untuk menjadi bagian dari basis data global. Misalnya, klaster saat ini harus menggunakan Aurora MySQL versi 1 dengan kompatibilitas MySQL 5.6. Untuk informasi tentang basis data global Aurora, lihat Bekerja dengan basis data global Amazon Aurora.	25 November 2019
Basis data global Aurora tersedia untuk Aurora MySQL versi 2	Mulai dari Aurora MySQL 2.07, Anda dapat membuat basis data global Aurora dengan kompatibilitas MySQL 5.7. Anda tidak perlu menentukan mode mesin `global` untuk klaster primer atau sekunder. Anda dapat menambahkan klaster yang baru disediakan dengan Aurora MySQL 2.07 atau yang lebih tinggi ke Basis Data Global Aurora. Untuk informasi tentang Basis Data Global Aurora, lihat Bekerja dengan basis data global Amazon Aurora.	25 November 2019

Pengoptimalan pertentangan hot row Aurora MySQL tersedia tanpa mode lab	Pengoptimalan pertentangan hot row kini sudah tersedia secara umum untuk Aurora MySQL dan mode lab Aurora tidak perlu ditetapkan ke ON. Fitur ini meningkatkan secara signifikan throughput untuk beban kerja dengan banyak transaksi yang bersaing untuk baris di halaman yang sama. Peningkatan ini melibatkan perubahan algoritma pembukaan kunci yang digunakan oleh Aurora MySQL.	19 November 2019
Hash join Aurora MySQL tersedia tanpa mode lab	Fitur hash join sudah tersedia secara umum untuk Aurora MySQL dan mode lab Aurora tidak perlu ditetapkan ke ON. Fitur ini dapat meningkatkan performa kueri saat Anda harus menggabungkan data dalam jumlah besar menggunakan equijoin. Untuk informasi selengkapnya tentang cara menggunakan fitur ini, lihat Bekerja dengan hash join di Aurora MySQL.	19 November 2019

Dukungan Aurora MySQL 2.* untuk lebih banyak kelas instans db.r5	Klaster Aurora MySQL kini mendukung jenis instans db.r5.8xlarge, db.r5.16x large, dan db.r5.24xlarge. Untuk informasi selengkapnya tentang jenis instans untuk klaster Aurora MySQL, lihat Memilih kelas instans DB.	19 November 2019
Dukungan Aurora MySQL 2.* untuk pelacakan mundur	Aurora MySQL versi 2.* kini menawarkan sebuah cara cepat untuk pulih dari kesalahan pengguna, seperti meletakkan tabel yang salah atau menghapus baris yang salah. Pelacakan mundur memungkinkan Anda memindahkan basis data ke titik waktu sebelumnya tanpa harus memulihkan dari cadangan, dan dapat diselesai kan dalam hitungan detik, bahkan untuk basis data yang besar. Untuk perincian, lihat Melacak mundur klaster DB Aurora.	19 November 2019

Dukungan tag penagihan untuk Aurora	Anda kini dapat menggunakan tag untuk terus melacak alokasi biaya untuk sumber daya seperti klaster Aurora, instans DB dalam klaster Aurora, I/O, cadangan, snapshot, dan sebagainya. Anda dapat melihat biaya yang terkait dengan setiap tag menggunakan AWS Cost Explorer. Untuk informasi selengkapnya tentang cara menggunakan tag dengan Aurora, lihat Memberikan Tag pada sumber daya Amazon RDS. Untuk informasi umum tentang tag dan cara menggunakannya untuk analisis biaya, lihat Menggunakan tag alokasi biaya dan Tag alokasi dana yang ditentukan pengguna.	23 Oktober 2019
API Data untuk Aurora PostgreSQL	Aurora PostgreSQL kini mendukung penggunaan API Data dengan klaster DB Amazon Aurora Serverless v1. Untuk informasi selengkapnya, lihat Menggunakan API Data untuk Aurora Serverless v1.	23 September 2019

Aurora PostgreSQL mendukung pengunggahan log database ke log CloudWatch	Anda dapat mengonfigurasi klaster DB PostgreSQL Aurora untuk mempublikasikan data log ke grup log di Amazon Logs. CloudWatch Dengan CloudWatch Log, Anda dapat melakukan analisis real-time dari data log, dan menggunakannya CloudWatch untuk membuat alarm dan melihat metrik. Anda dapat menggunakan CloudWatch Log untuk menyimpan catatan log Anda dalam penyimpanan yang sangat tahan lama. Untuk informasi selengkapnya, lihat Menerbitkan log PostgreSQL Aurora ke Amazon Logs. CloudWatch	9 Agustus 2019
Klaster multi-master untuk Aurora MySQL	Anda dapat menyiapkan klaster multi-master Aurora MySQL. Dalam klaster ini, setiap instans DB memiliki kemampuan baca/tulis. Untuk informasi selengkapnya, lihat Bekerja dengan klaster multi-master Aurora.	8 Agustus 2019

Aurora PostgreSQL mendukung Aurora Serverless v1	Anda kini dapat menggunakan Amazon Aurora Serverless v1 dengan Aurora PostgreSQL. Klaster DB Aurora Nirserver akan secara otomatis aktif, nonaktif, dan menaikkan atau menurunkan skala kapasitas komputasinya berdasarkan kebutuhan aplikasi Anda. Untuk informasi selengkapnya, lihat Menggunakan Amazon Aurora Serverless v1.	9 Juli 2019
Pengklonaan lintas akun untuk Aurora MySQL	Anda sekarang dapat mengkloning volume cluster untuk cluster Aurora MySQL DB antar akun. AWS Anda mengizinkan berbagi melalui AWS Resource Access Manager (AWS RAM). Volume kloning kloning menggunakan copy-on-write mekanisme, yang hanya membutuhkan penyimpanan tambahan untuk data baru atau yang diubah. Untuk informasi selengkapnya tentang pengklonaan untuk Aurora, lihat Mengklona basis data di klaster DB Aurora.	2 Juli 2019
Aurora PostgreSQL mendukung kelas instans DB db.t3	Anda kini dapat membuat klaster DB Aurora PostgreSQL yang menggunakan kelas instans DB db.t3. Untuk informasi selengkapnya, lihat Kelas instans DB.	20 Juni 2019

Dukungan untuk pengimpor an data dari Amazon S3 untuk Aurora PostgreSQL	Anda kini dapat mengimpor data dari file Amazon S3 ke tabel di klaster DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Mengekspor data Amazon S3 ke klaster DB Aurora PostgreSQL.	19 Juni 2019
Aurora PostgreSQL kini menyediakan pemulihan failover cepat dengan manajemen cache klaster	Aurora PostgreSQL kini menyediakan manajemen cache klaster untuk memastika n pemulihan instans DB primer yang cepat jika terjadi failover. Untuk informasi selengkapnya, lihat Pemulihan cepat setelah failover dengan manajemen cache klaster.	11 Juni 2019
API Data untuk Aurora Serverless v1 tersedia secara umum	Anda dapat mengakses klaster Aurora Serverless v1 dengan aplikasi berbasis layanan web menggunakan Data API. Untuk informasi selengkapnya, lihat Menggunakan API Data untuk Aurora Serverless v1.	30 Mei 2019
Aurora PostgreSQL mendukung pemantauan basis data dengan aliran aktivitas basis data	Aurora PostgreSQL sekarang menyertakan aliran aktivitas database, yang menyediakan aliran data aktivitas database dalam near-real-time database relasional Anda. Untuk informasi selengkapnya, lihat Menggunakan aliran aktivitas basis data.	30 Mei 2019

Rekomendasi Amazon Aurora	Saat ini, Amazon Aurora menyediakan rekomendasi otomatis untuk sumber daya Aurora. Untuk informasi selengkapnya, lihat Menggunakan rekomendasi Amazon Aurora.	22 Mei 2019
Dukungan Wawasan Performa untuk basis data global Aurora	Anda kini dapat menggunakan Wawasan Performa dengan Basis Data Global Aurora. Untuk informasi tentang Wawasan Performa Aurora, lihat Menggunakan wawasan performa Amazon RDS. Untuk informasi tentang basis data global Aurora, lihat Bekerja dengan basis data global Aurora.	13 Mei 2019
Wawasan Performa tersedia untuk Aurora MySQL 5.7	Wawasan Performa Amazon RDS kini tersedia untuk Aurora MySQL versi 2.x, yang kompatibel dengan MySQL 5.7. Untuk informasi selengkapnya, lihat Menggunakan wawasan performa Amazon RDS.	3 Mei 2019

Database global Aurora tersedia di lebih banyak Wilayah AWS	Anda sekarang dapat membuat database global Aurora di sebagian besar tempat Wilayah AWS Aurora tersedia. Untuk informasi tentang basis data global Aurora, lihat Bekerja dengan basis data global Amazon Aurora.	30 April 2019
Kapasitas minimum adalah sebesar 1 untuk Aurora Serverless v1	Pengaturan kapasitas minimum yang dapat Anda gunakan untuk sebuah klaster Aurora Serverles s v1 adalah 1. Sebelumny a, jumlah minimumnya adalah 2. Untuk informasi tentang cara menentukan nilai kapasitas Aurora Nirserver, lihat Menetapkan kapasitas klaster DB Aurora Serverless v1.	29 April 2019
Tindakan waktu tunggu habis Aurora Serverless v1	Anda kini dapat menentukan tindakan yang harus diambil saat perubahan kapasitas Aurora Serverless v1 habis waktu tunggunya. Untuk informasi selengkapnya, lihat Tindakan waktu tunggu habis untuk perubahan kapasitas.	29 April 2019

Penagihan per detik	Amazon RDS sekarang ditagih dalam kenaikan 1 detik di semua Wilayah AWS kecuali AWS GovCloud (AS) untuk instans sesuai permintaan. Untuk informasi selengkapnya, lihat Penagihan instans DB untuk Aurora.	25 April 2019
Berbagi Aurora Serverless v1 snapshot di seluruh Wilayah AWS	Dengan Aurora Serverless v1, snapshot selalu dienkripsi. Jika Anda mengenkripsi snapshot dengan milik Anda sendiri AWS KMS key, Anda sekarang dapat menyalin atau membagikan snapshot tersebut. Wilayah AWS Untuk informasi tentang snapshot klaster DB Aurora Serverless v1, lihat Aurora Serverless v1 dan snapshot.	17 April 2019
Memulihkan cadangan MySQL 5.7 dari Amazon S3	Anda kini dapat membuat sebuah cadangan basis data MySQL versi 5.7, menyimpannya di Amazon S3, lalu memulihkan file cadangan ke sebuah klaster DB Aurora MySQL yang baru. Untuk informasi selengkapnya, lihat Memigrasikan data dari basis data MySQL eksternal ke klaster DB Aurora MySQL.	17 April 2019

Berbagi snapshot Aurora Serverless v1 antar-wilayah	Dengan Aurora Serverless v1, snapshot selalu dienkripsi. Jika Anda mengenkripsi snapshot dengan milik Anda sendiri AWS KMS key, Anda sekarang dapat menyalin atau membagikan snapshot di seluruh wilayah. Untuk informasi tentang snapshot klaster DB Aurora Serverless v1, lihat Aurora Nirserver dan snapshot.	16 April 2019
Tutorial Aurora proof-of-concept	Anda dapat mempelajari cara melakukan pembuktian konsep untuk mencoba aplikasi dan beban kerja Anda dengan Aurora. Untuk tutorial lengkap, lihat Melakukan pembuktian konsep Aurora.	16 April 2019
Aurora Serverless v1 mendukung pemulihan dari cadangan Amazon S3	Anda kini dapat mengimpor backup dari Amazon S3 ke klaster Aurora Serverless. Untuk detail tentang prosedur tersebut, lihat Memigrasikan data dari MySQL dengan menggunakan bucket Amazon S3.	16 April 2019

Parameter baru yang dapat dimodifikasi untuk Aurora Serverless v1	Anda kini dapat memodifikasi parameter DB berikut untuk klaster Aurora Serverless v1: `innodb_file_format` , `innodb_file_per_table` , `innodb_large_prefix` , `innodb_lock_wait_timeout` , `innodb_monitor_disable` , `innodb_monitor_enable` , `innodb_monitor_reset` , `innodb_monitor_reset_all` , `innodb_print_all_deadlocks` , `log_warnings` , `net_read_timeout` , `net_retry_count` , `net_write_timeout` , `sql_mode`, dan `tx_isolation` . Untuk informasi selengkapnya tentang parameter konfigurasi untuk klaster Aurora Serverless v1, lihat Aurora Serverless v1 dan grup parameter.	4 April 2019
Aurora PostgreSQL mendukung kelas instans DB db.r5	Anda kini dapat membuat klaster DB Aurora PostgreSQL yang menggunakan kelas instans DB db.r5. Untuk informasi selengkapnya, lihat Kelas instans DB.	4 April 2019

Replikasi logis Aurora PostgreSQL	Anda kini dapat menggunakan replikasi logis PostgreSQL untuk mereplikasi bagian dari basis data untuk klaster DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menggunakan replikasi logis PostgreSQL.	28 Maret 2019
Dukungan GTID untuk Aurora MySQL 2.04	Anda kini dapat menggunakan replikasi dengan fitur ID transaksi global (GTID) dari MySQL 5.7. Fitur ini menyederhanakan proses replikasi log biner (binlog) antara Aurora MySQL dan basis data eksternal yang kompatibel dengan MySQL 5.7. Replikasi tersebut dapat menggunakan klaster Aurora MySQL sebagai sumber atau tujuan. Fitur ini tersedia untuk Aurora MySQL 2.04 dan yang lebih tinggi. Untuk informasi selengkapnya tentang replikasi berbasis GTID dan Aurora MySQL, lihat Menggunakan replikasi berbasis GTID untuk Aurora MySQL.	25 Maret 2019

Mengunggah Aurora Serverless v1 log ke Amazon CloudWatch	Anda sekarang dapat meminta Aurora mengunggah log database CloudWatch untuk sebuah Aurora Serverless v1 cluster. Untuk informasi selengkapnya, lihat Melihat klaster DB Aurora Nirserver. Sebagai bagian dari peningkatan ini, Anda kini dapat menentukan nilai untuk parameter tingkat instans di grup parameter klaster DB, dan nilai tersebut berlaku untuk semua instans DB di klaster tersebut kecuali jika Anda menggantinya di grup parameter DB. Untuk informasi selengkapnya, lihat Bekerja dengan grup parameter DB dan grup parameter klaster DB.	25 Februari 2019
Aurora MySQL mendukung kelas instans DB db.t3	Anda kini dapat membuat klaster DB Aurora MySQL yang menggunakan kelas instans DB db.t3. Untuk informasi selengkapnya, lihat Kelas instans DB.	25 Februari 2019
Aurora MySQL mendukung kelas instans DB db.r5	Anda kini dapat membuat klaster DB Aurora MySQL yang menggunakan kelas instans DB db.r5. Untuk informasi selengkapnya, lihat Kelas instans DB.	25 Februari 2019

Penghitung Wawasan Performa untuk Aurora MySQL	Anda kini dapat menambahkan penghitung performa ke diagram Wawasan Performa untuk instans DB Aurora MySQL. Untuk informasi selengkapnya, lihat Komponen dasbor Wawasan Performa.	19 Februari 2019
Wawasan Performa Amazon RDS mendukung tampilan lebih banyak teks SQL untuk Aurora MySQL	Wawasan Performa Amazon RDS kini mendukung tampilan lebih banyak teks SQL di dasbor Wawasan Performa untuk instans DB Aurora MySQL. Untuk informasi selengkapnya, lihat Menampilkan lebih banyak teks SQL di dasbor Wawasan Performa.	6 Februari 2019
Wawasan Performa Amazon RDS mendukung tampilan lebih banyak teks SQL untuk Aurora PostgreSQL	Wawasan Performa Amazon RDS kini mendukung tampilan lebih banyak teks SQL di dasbor Wawasan Performa untuk instans DB Aurora PostgreSQL. Untuk informasi selengkapnya, lihat Menampilkan lebih banyak teks SQL di dasbor Wawasan Performa.	24 Januari 2019

Penagihan cadangan Aurora	Anda dapat menggunakan CloudWatch metrik `AmazonTotalBackupStorageBilled`, `SnapshotStorageUsed`, dan `BackupRetentionPeriodStorageUsed` untuk memantau penggunaan ruang cadangan Aurora Anda. Untuk informasi selengkapnya tentang cara menggunakan CloudWatch metrik, lihat Ikhtisar pemantauan. Untuk informasi selengkapnya tentang cara mengelola penyimpanan untuk data cadangan, lihat Memahami penggunaan penyimpanan cadangan Aurora.	3 Januari 2019
Penghitung Wawasan Performa	Anda kini dapat menambahkan penghitung performa ke diagram Wawasan Performa. Untuk informasi selengkapnya, lihat Komponen dasbor Wawasan Performa.	6 Desember 2018

Basis data global Aurora	Anda kini dapat membuat basis data global Aurora. Basis data global Aurora mencakup beberapa Wilayah AWS, memungkinkan pembacaan global latensi rendah dan pemulihan bencana dari pemadaman di seluruh wilayah. Untuk informasi selengkapnya, lihat Bekerja dengan basis data global Amazon Aurora.	28 November 2018
Manajemen rencana kueri di Aurora PostgreSQL	Aurora PostgreSQL kini menyediakan manajemen rencana kueri yang dapat Anda gunakan untuk mengelola rencana eksekusi kueri PostgreSQL. Untuk informasi selengkapnya, lihat Mengelola rencana eksekusi kueri untuk Aurora PostgreSQL.	20 November 2018
Editor kueri untuk Aurora Serverless v1 (beta)	Anda dapat menjalankan pernyataan SQL dalam konsol Amazon RDS pada klaster Aurora Serverless v1. Untuk informasi selengkapnya, lihat Menggunakan editor kueri untuk Aurora Serverless v1.	20 November 2018

API Data untuk Aurora Serverless v1 (beta)	Anda dapat mengakses klaster Aurora Serverless v1 dengan aplikasi berbasis layanan web menggunakan API Data. Untuk informasi selengkapnya, lihat Menggunakan API Data untuk Aurora Nirserver.	20 November 2018
Dukungan TLS untuk Aurora Serverless v1	Klaster Aurora Serverless v1 mendukung enkripsi TLS/SSL. Untuk informasi selengkapnya, lihat TLS/SSL untuk Aurora Nirserver.	19 November 2018
Titik akhir kustom	Anda kini dapat membuat titik akhir yang dikaitkan dengan kumpulan instans DB arbitrer. Fitur ini membantu penyeimbangan beban dan ketersediaan yang tinggi untuk klaster Aurora dengan beberapa instans DB memiliki kapasitas atau konfigurasi yang berbeda dibandingkan yang lain. Anda dapat menggunakan titik akhir kustom, bukannya terhubung ke instans DB tertentu melalui titik akhir instansnya. Untuk informasi selengkapnya, lihat Manajemen koneksi Amazon Aurora.	12 November 2018
Dukungan autentikasi IAM di Aurora PostgreSQL	Aurora PostgreSQL kini mendukung autentikasi IAM. Untuk informasi selengkapnya, lihat Autentikasi basis data IAM.	8 November 2018

Grup parameter kustom untuk restorasi dan pemulihan titik waktu	Anda kini dapat menentukan grup parameter kustom saat memulihkan snapshot atau melakukan operasi pemulihan titik waktu. Untuk informasi selengkapnya, lihat Memulihkan dari snapshot klaster DB dan Memulihkan klaster DB ke waktu yang ditentukan.	15 Oktober 2018
Perlindungan penghapusan untuk klaster DB Aurora	Saat Anda mengaktifkan perlindungan penghapusan untuk klaster DB, basis data tidak dapat dihapus oleh pengguna. Untuk informasi selengkapnya, lihat Menghapus klaster DB.	26 September 2018
Fitur Stop/Start Aurora	Anda kini dapat menonaktifkan atau mengaktifkan keseluruhan klaster Aurora dengan satu pengoperasian. Untuk informasi selengkapnya, lihat Menonaktifkan dan mengaktifkan klaster Aurora.	24 September 2018

Fitur kueri paralel untuk Aurora MySQL	Aurora MySQL kini menawarkan sebuah opsi untuk memparalelkan kerja I/O untuk kueri di seluruh infrastruktur penyimpanan Aurora. Fitur ini mempercepat kueri analitik intensif data, yang sering kali merupakan pengoperasian yang paling banyak memakan waktu dalam beban kerja. Untuk informasi selengkapnya, lihat Bekerja dengan kueri paralel untuk Aurora MySQL.	20 September 2018
Panduan baru	Ini merupakan rilisan pertama dari Panduan Pengguna Amazon Aurora.	31 Agustus 2018

AWS Glosarium

Untuk AWS terminologi terbaru, lihat AWS glosarium di Referensi.Glosarium AWS

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.