aws

Panduan Pengguna

Amazon Aurora DSQL

Amazon Aurora DSQL: Panduan Pengguna

Table of Contents

Apa itu Amazon Aurora DSQL?

Amazon Aurora DSQL adalah layanan database relasional terdistribusi tanpa server yang dioptimalkan untuk beban kerja transaksional. Aurora DSQL menawarkan skala yang hampir tidak terbatas dan tidak mengharuskan Anda mengelola infrastruktur. Arsitektur aktif yang sangat tersedia menyediakan 99,99% Single-region dan 99,999% ketersediaan Multi-region.

Kapan menggunakan Aurora DSQL

Aurora DSQL dioptimalkan untuk beban kerja transaksional yang mendapat manfaat dari transaksi ACID dan model data relasional. Karena tanpa server, Aurora DSQL sangat ideal untuk pola aplikasi arsitektur microservice, serverless, dan event-driven. Aurora DSQL kompatibel dengan PostgreSQL, sehingga Anda dapat menggunakan driver yang sudah dikenal, pemetaan relasional objek (), kerangka kerja, dan fitur SQL. ORMs

Aurora DSQL secara otomatis mengelola infrastruktur sistem dan skala komputasi, I/O, dan penyimpanan berdasarkan beban kerja Anda. Karena Anda tidak memiliki server untuk menyediakan atau mengelola, Anda tidak perlu khawatir tentang downtime pemeliharaan yang terkait dengan penyediaan, penambalan, atau peningkatan infrastruktur.

Aurora DSQL membantu Anda membangun dan memelihara aplikasi perusahaan yang selalu tersedia dalam skala apa pun. Desain tanpa server aktif mengotomatiskan pemulihan kegagalan, jadi Anda tidak perlu khawatir tentang failover database tradisional. Aplikasi Anda mendapat manfaat dari ketersediaan Multi-AZ dan Multi-wilayah, dan Anda tidak perlu khawatir tentang konsistensi akhirnya atau data yang hilang terkait dengan failover.

Fitur utama di Aurora DSQL

Fitur utama berikut membantu Anda membuat database terdistribusi tanpa server untuk mendukung aplikasi ketersediaan tinggi Anda:

Arsitektur terdistribusi

Aurora DSQL terdiri dari komponen-komponen multi-tenant berikut:

- Relay dan konektivitas
- Komputasi dan database

- Log transaksi, kontrol konkurensi, dan isolasi
- Penyimpanan

Bidang kontrol mengoordinasikan komponen sebelumnya. Setiap komponen menyediakan redundansi di tiga Availability Zones (AZs), dengan penskalaan cluster otomatis dan penyembuhan sendiri jika terjadi kegagalan komponen. Untuk mempelajari lebih lanjut tentang bagaimana arsitektur ini mendukung ketersediaan tinggi, lihatKetahanan di Amazon Aurora DSQL.

Cluster Single-Region dan Multi-region

Cluster Aurora DSQL memberikan manfaat sebagai berikut:

- Replikasi data sinkron
- Operasi baca yang konsisten
- Pemulihan kegagalan otomatis
- Konsistensi data di beberapa AZs atau Wilayah

Jika komponen infrastruktur gagal, Aurora DSQL secara otomatis merutekan permintaan ke infrastruktur yang sehat tanpa intervensi manual. Aurora DSQL menyediakan transaksi atomisitas, konsistensi, isolasi, dan daya tahan (ACID) dengan konsistensi yang kuat, isolasi snapshot, atomisitas, dan daya tahan lintas AZ dan lintas wilayah.

Cluster peered Multi-Region memberikan ketahanan dan konektivitas yang sama dengan cluster Single-region. Tetapi mereka meningkatkan ketersediaan dengan menawarkan dua titik akhir Regional, satu di setiap wilayah cluster peered. Kedua titik akhir dari cluster peered menyajikan database logis tunggal. Mereka tersedia untuk operasi baca dan tulis bersamaan, dan memberikan konsistensi data yang kuat. Anda dapat membangun aplikasi yang berjalan di beberapa Wilayah secara bersamaan untuk kinerja dan ketahanan—dan tahu bahwa pembaca selalu melihat data yang sama.

Kompatibilitas dengan database PostgreSQL

Lapisan database terdistribusi (komputasi) di Aurora DSQL didasarkan pada versi utama PostgreSQL saat ini. Anda dapat terhubung ke Aurora DSQL dengan driver dan alat PostgreSQL yang sudah dikenal, seperti. `psql` Aurora DSQL saat ini kompatibel dengan PostgreSQL versi 16 dan mendukung subset fitur PostgreSQL, ekspresi, dan tipe data. Untuk informasi selengkapnya tentang fitur SQL yang didukung, lihatKompatibilitas fitur SQL di Aurora DSQL.

Ketersediaan wilayah untuk Aurora DSQL

Dengan Amazon Aurora DSQL, Anda dapat menerapkan instans database di beberapa Wilayah AWS untuk mendukung aplikasi global dan memenuhi persyaratan residensi data. Ketersediaan wilayah menentukan di mana Anda dapat membuat dan mengelola cluster database Aurora DSQL. Administrator database dan arsitek aplikasi yang perlu merancang sistem basis data yang sangat tersedia dan terdistribusi secara global sering kali perlu memahami dukungan Wilayah untuk beban kerja mereka. Kasus penggunaan umum termasuk menyiapkan pemulihan bencana lintas wilayah, melayani pengguna dari instans basis data yang lebih dekat secara geografis untuk mengurangi latensi, dan memelihara salinan data di lokasi tertentu untuk kepatuhan.

Tabel berikut menunjukkan di Wilayah AWS mana Aurora DSQL saat ini tersedia dan titik akhir untuk masing-masing. Wilayah AWS

Didukung Wilayah AWS dan titik akhir

Nama wilayah	Wilayah	Titik Akhir	Protokol
US East (N. Virginia)	us-east-1	dsql.us-east-1.api.aws	HTTPS
US East (Ohio)	us-east-2	dsql.us-east-2.api.aws	HTTPS
US West (Oregon)	us-west-2	dsql.us-west-2.api.aws	HTTPS
Europe (Ireland)	eu-west-1	dsql.eu-west-1.api.aws	HTTPS
Europe (London)	eu-west-2	dsql.eu-west-2.api.aws	HTTPS
Europe (Paris)	eu-west-3	dsql.eu-west-3.api.aws	HTTPS
Asia Pacific (Tokyo)	ap-northeast-1	dsql.ap-northeast-1.api.aws	HTTPS
Asia Pacific (Seoul)	ap-northeast-2	dsql.ap-northeast-2.api.aws	HTTPS
Asia Pacific (Osaka)	ap-northeast-3	dsql.ap-northeast-3.api.aws	HTTPS

Ketersediaan klaster Multi-Region untuk Aurora DSQL

Anda dapat membuat klaster Multi-wilayah Aurora DSQL dalam set Wilayah tertentu. AWS Setiap Wilayah menetapkan kelompok Wilayah yang terkait secara geografis yang dapat bekerja sama dalam klaster Multi-wilayah.

Wilayah AS

- Timur AS (N. Virginia)
- AS Timur (Ohio)
- AS Barat (Oregon)

Wilayah Asia Pasifik

- Asia Pasifik (Osaka)
- Asia Pasifik (Seoul)
- Asia Pasifik (Tokyo)

Wilayah Eropa

- Eropa (Irlandia)
- Eropa (London)
- Eropa (Paris)

Batasan Penting

Cluster Multi-Region harus dibuat dalam satu set Region. Misalnya, Anda tidak dapat membuat klaster yang mencakup Wilayah AS Timur (Virginia Utara) dan Eropa (Irlandia).

> **Important**
>
> Aurora DSQL saat ini tidak mendukung klaster Multi-wilayah lintas benua.

Harga untuk Aurora DSQL

Untuk informasi biaya, lihat harga Aurora DSQL.

Apa selanjutnya?

Untuk informasi tentang komponen inti di Aurora DSQL dan untuk memulai layanan, lihat berikut ini:

- Memulai dengan Aurora DSQL
- Kompatibilitas fitur SQL di Aurora DSQL
- Mengakses Aurora DSQL
- Aurora DSQL dan PostgreSQL

Memulai dengan Aurora DSQL

Amazon Aurora DSQL adalah database relasional terdistribusi tanpa server yang dioptimalkan untuk beban kerja transaksional. Di bagian berikut, Anda akan belajar cara membuat kluster DSQL Aurora wilayah tunggal dan Multi-wilayah, menghubungkannya, dan membuat serta memuat skema sampel. Anda akan mengakses cluster dengan AWS Management Console dan berinteraksi dengan database Anda menggunakan `psql` utilitas. Pada akhirnya, Anda akan memiliki cluster Aurora DSQL yang berfungsi dan siap digunakan untuk beban kerja pengujian atau produksi.

Topik

Prasyarat

Sebelum Anda dapat mulai menggunakan Aurora DSQL, pastikan Anda memenuhi prasyarat berikut:

- Identitas IAM Anda harus memiliki izin untuk masuk ke. AWS Management Console
- Identitas IAM Anda harus memenuhi salah satu kriteria berikut:
  - Akses untuk melakukan tindakan apa pun pada sumber daya apa pun di Akun AWS
  - Izin IAM `iam:CreateServiceLinkedRole` dan kemampuan untuk mendapatkan akses ke tindakan kebijakan IAM `dsql:*`
- Jika Anda menggunakan lingkungan mirip Unix, pastikan bahwa Python versi 3.8+ dan versi 14+ diinstal. AWS CLI `psql` Untuk memeriksa versi aplikasi Anda, jalankan perintah berikut.

```
python3 --version
psql --version
```

  Jika Anda menggunakan AWS CLI di lingkungan yang berbeda, pastikan bahwa Anda secara manual mengatur Python versi 3.8+ dan versi 14+. `psql`

- Jika Anda berniat mengakses Aurora DSQL menggunakan, AWS CloudShell Python versi 3.8+ dan `psql` versi 14+ disediakan tanpa pengaturan tambahan. Untuk informasi lebih lanjut tentang AWS CloudShell, lihat Apa itu AWS CloudShell? .
- Jika Anda berniat untuk mengakses Aurora DSQL menggunakan GUI, gunakan atau. DBeaver JetBrains DataGrip Untuk informasi selengkapnya, lihat Mengakses Aurora DSQL dengan DBeaver dan Mengakses Aurora DSQL dengan JetBrains DataGrip.

Mengakses Aurora DSQL

Anda dapat mengakses Aurora DSQL melalui teknik berikut. Untuk mempelajari cara menggunakan CLI,, dan APIs SDKs, lihat. Mengakses Aurora DSQL

Topik

- Mengakses Aurora DSQL melalui AWS Management Console
- Mengakses Aurora DSQL menggunakan klien SQL
- Menggunakan protokol PostgreSQL dengan Aurora DSQL

Mengakses Aurora DSQL melalui AWS Management Console

Anda dapat mengakses AWS Management Console untuk Aurora DSQL di. https://console.aws.amazon.com/dsql

Mengakses Aurora DSQL menggunakan klien SQL

Aurora DSQL menggunakan protokol PostgreSQL. Gunakan klien interaktif pilihan Anda dengan menyediakan token autentikasi IAM yang ditandatangani sebagai kata sandi saat menghubungkan ke klaster Anda. Token otentikasi adalah string karakter unik yang dihasilkan Aurora DSQL secara dinamis AWS menggunakan Signature Version 4.

Aurora DSQL menggunakan token hanya untuk otentikasi. Token tidak memengaruhi koneksi setelah dibuat. Jika Anda mencoba menyambung kembali menggunakan token kedaluwarsa, permintaan koneksi ditolak. Untuk informasi selengkapnya, lihat Menghasilkan token otentikasi di Amazon Aurora DSQL.

Topik

- Mengakses Aurora DSQL dengan psql (terminal interaktif PostgreSQL)

- Mengakses Aurora DSQL dengan DBeaver
- Mengakses Aurora DSQL dengan JetBrains DataGrip

Mengakses Aurora DSQL dengan psql (terminal interaktif PostgreSQL)

`psql`Utilitas adalah front-end berbasis terminal untuk PostgreSQL. Ini memungkinkan Anda untuk mengetik kueri secara interaktif, menerbitkannya ke PostgreSQL, dan melihat hasil kueri. Untuk meningkatkan waktu respons kueri, gunakan klien PostgreSQL versi 17.

Unduh installer sistem operasi Anda dari halaman Unduhan PostgreSQL. Untuk informasi lebih lanjut tentang`psql`, lihat https://www.postgresql. org/docs/current/app-psql.htm.

Jika Anda sudah AWS CLI menginstal, gunakan contoh berikut untuk terhubung ke cluster Anda. Anda dapat menggunakan AWS CloudShell, yang dilengkapi dengan `psql` prainstal, atau Anda dapat menginstal `psql` langsung.

```
# Aurora DSQL requires a valid IAM token as the password when connecting.
# Aurora DSQL provides tools for this and here we're using Python.
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token \
  --region us-east-1 \
  --expires-in 3600 \
  --hostname your_cluster_endpoint)

# Aurora DSQL requires SSL and will reject your connection without it.
export PGSSLMODE=require

# Connect with psql, which automatically uses the values set in PGPASSWORD and
 PGSSLMODE.
# Quiet mode suppresses unnecessary warnings and chatty responses but still outputs
 errors.
psql --quiet \
  --username admin \
  --dbname postgres \
  --host your_cluster_endpoint
```

Mengakses Aurora DSQL dengan DBeaver

DBeaver adalah alat basis data berbasis GUI open-source. Anda dapat menggunakannya untuk terhubung dan mengelola database Anda. Untuk mengunduh DBeaver, lihat halaman unduhan di situs web DBeaver Komunitas. Langkah-langkah berikut menjelaskan cara menghubungkan ke cluster Anda menggunakan DBeaver.

Untuk mengatur koneksi Aurora DSQL baru di DBeaver

1. Pilih Koneksi Database Baru.
2. Di jendela New Database Connection, pilih PostgreSQL.
3. Di tab Pengaturan Koneksi/Utama, pilih Connect by: Host dan masukkan informasi berikut.
   - Host — Gunakan endpoint cluster Anda.

     Database — Masukkan `postgres`

     Otentikasi - Pilih `Database Native`

     Nama Pengguna — Masukkan `admin`

     Kata Sandi — Hasilkan token otentikasi. Salin token yang dihasilkan dan gunakan sebagai kata sandi Anda.
4. Abaikan peringatan apa pun dan tempel token otentikasi Anda ke bidang DBeaverKata Sandi.

   > **Note**
   >
   > Anda harus mengatur mode SSL di koneksi klien. Aurora DSQL mendukung. `SSLMODE=require` Aurora DSQL memberlakukan komunikasi SSL di sisi server dan menolak koneksi non-SSL.
5. Anda harus terhubung ke cluster Anda dan dapat mulai menjalankan pernyataan SQL.

> **Important**
>
> Fitur administratif yang disediakan oleh DBeaver database PostgreSQL (seperti Session Manager dan Lock Manager) tidak berlaku untuk database, karena arsitekturnya yang unik. Meskipun dapat diakses, layar ini tidak memberikan informasi yang dapat dipercaya tentang kesehatan atau status database.

Otentikasi kredensi kedaluwarsa untuk DBeaver

Sesi yang ditetapkan tetap diautentikasi selama maksimal 1 jam atau sampai DBeaver terputus atau waktu habis. Untuk membuat koneksi baru, berikan token otentikasi yang valid di bidang Kata Sandi pengaturan koneksi. Mencoba membuka sesi baru (misalnya, untuk membuat daftar tabel baru, atau

konsol SQL baru) memaksa upaya otentikasi baru. Jika token otentikasi yang dikonfigurasi dalam pengaturan Koneksi tidak lagi valid, sesi baru gagal, dan DBeaver membatalkan semua sesi yang dibuka sebelumnya. Ingatlah hal ini saat memilih durasi token autentikasi IAM Anda dengan opsi tersebut. `expires-in`

Mengakses Aurora DSQL dengan JetBrains DataGrip

JetBrains DataGrip adalah IDE lintas platform untuk bekerja dengan SQL dan database, termasuk PostgreSQL. DataGrip termasuk GUI yang kuat dengan editor SQL cerdas. Untuk mengunduh DataGrip, buka halaman unduhan di situs JetBrains web.

Untuk mengatur koneksi Aurora DSQL baru di JetBrains DataGrip

1. Pilih Sumber Data Baru dan pilih PostgreSQL.
2. Di Sources/General tab Data, masukkan informasi berikut:
   - Host — Gunakan endpoint cluster Anda.

     Port - Aurora DSQL menggunakan PostgreSQL default: `5432`

     Database - Aurora DSQL menggunakan PostgreSQL default `postgres`

     Otentikasi — Pilih`User & Password` .

     Nama Pengguna — Masukkan`admin`.

     Kata sandi — Hasilkan token dan tempelkan ke bidang ini.

     URL - Jangan ubah bidang ini. Ini akan diisi secara otomatis berdasarkan bidang lain.
3. Kata sandi — Berikan ini dengan membuat token otentikasi. Salin output yang dihasilkan dari generator token dan tempel ke bidang kata sandi.

   > **Note**
   >
   > Anda harus mengatur mode SSL di koneksi klien. Aurora DSQL mendukung. `PGSSLMODE=require` Aurora DSQL memberlakukan komunikasi SSL di sisi server dan akan menolak koneksi non-SSL.
4. Anda harus terhubung ke cluster Anda dan dapat mulai menjalankan pernyataan SQL:

> **Important**
>
> Beberapa tampilan yang DataGrip disediakan oleh database PostgreSQL (seperti Sessions) tidak berlaku untuk database karena arsitekturnya yang unik. Meskipun dapat diakses, layar ini tidak memberikan informasi yang dapat dipercaya tentang sesi aktual yang terhubung ke database.

Kedaluwarsa kredensi otentikasi

Sesi yang ditetapkan tetap diautentikasi selama maksimal 1 jam atau sampai pemutusan eksplisit atau batas waktu sisi klien terjadi. Jika koneksi baru perlu dibuat, token Otentikasi baru harus dibuat dan disediakan di bidang Kata Sandi Properti Sumber Data. Mencoba membuka sesi baru (misalnya, untuk membuat daftar tabel baru, atau konsol SQL baru) memaksa upaya otentikasi baru. Jika token otentikasi yang dikonfigurasi dalam pengaturan Koneksi tidak lagi valid, sesi baru itu akan gagal dan semua sesi yang dibuka sebelumnya akan menjadi tidak valid.

Menggunakan protokol PostgreSQL dengan Aurora DSQL

PostgreSQL menggunakan protokol berbasis pesan untuk komunikasi antara klien dan server. Protokol ini didukung TCP/IP berulang-ulang melalui soket unix-domain. Tabel berikut menunjukkan bagaimana Aurora DSQL mendukung protokol PostgreSQL.

PostgreSQL	Aurora DSQL	Catatan
Peran (juga dikenal sebagai Pengguna atau Grup)	Peran Database	Aurora DSQL menciptakan peran untuk Anda bernama. `admin` Saat membuat peran basis data kustom, Anda harus menggunakan peran tersebut untuk mengaitkannya dengan `admin` peran IAM untuk mengautentikasi saat menghubungkan ke klaster Anda. Untuk informasi selengkapnya, lihat Mengonfigurasi peran basis data kustom.
Host (juga dikenal sebagai hostname atau hostspec)	Titik Akhir klaster	Aurora DSQL Kluster wilayah tunggal menyediakan satu titik akhir terkelola dan secara otomatis mengarahkan lalu lintas jika tidak tersedia di dalam Wilayah.

PostgreSQL	Aurora DSQL	Catatan
Port	N/A - gunakan default `5432`	Ini adalah PostgreSQL default.
Database (dbname)	menggunakan `postgres`	Aurora DSQL membuat database ini untuk Anda saat Anda membuat cluster.
Modus SSL	SSL selalu diaktifkan di sisi server	Di Aurora DSQL, Aurora DSQL mendukung Mode SSL. `require` Koneksi tanpa SSL ditolak oleh Aurora DSQL.
Kata sandi	Token Otentikasi	Aurora DSQL membutuhkan token otentikasi sementara alih-alih kata sandi yang berumur panjang. Untuk mempelajari selengkap nya, lihat Menghasilkan token otentikasi di Amazon Aurora DSQL.

Langkah 1: Buat cluster Aurora DSQL Single-region

Unit dasar Aurora DSQL adalah cluster, yang merupakan tempat Anda menyimpan data Anda. Dalam tugas ini, Anda membuat cluster dalam satu Wilayah AWS.

Untuk membuat cluster Single-region di Aurora DSQL

1. Masuk ke AWS Management Console dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql
2. Pilih Buat cluster dan kemudian Single-Region.
3. (Opsional) Dalam pengaturan Cluster, pilih salah satu opsi berikut:
   - Pilih Sesuaikan pengaturan enkripsi (lanjutan) untuk memilih atau membuat pengaturan AWS KMS key.
   - Pilih Aktifkan perlindungan penghapusan untuk mencegah operasi penghapusan klaster Anda. Secara default, perlindungan penghapusan dipilih.
4. (Opsional) Di Tag, pilih atau masukkan tag untuk cluster ini.
5. Pilih Buat klaster.

Langkah 2: Hubungkan ke cluster Aurora DSQL Anda

Titik akhir cluster secara otomatis dihasilkan saat Anda membuat cluster Aurora DSQL berdasarkan ID cluster dan Region. Format penamaan adalah*clusterid*.dsql.*region*.on.aws. Klien menggunakan endpoint untuk membuat koneksi jaringan ke cluster Anda.

Otentikasi dikelola menggunakan IAM sehingga Anda tidak perlu menyimpan kredensi dalam database. Token otentikasi adalah string karakter unik yang dihasilkan secara dinamis. Token hanya digunakan untuk otentikasi dan tidak memengaruhi koneksi setelah dibuat. Sebelum mencoba terhubung, pastikan identitas IAM Anda memiliki `dsql:DbConnectAdmin` izin, seperti yang dijelaskan dalam. Prasyarat

> **Note**
>
> Untuk mengoptimalkan kecepatan koneksi database, gunakan klien PostgreSQL versi 17 dan atur untuk mengarahkan:. `PGSSLNEGOTIATION PGSSLNEGOTIATION=direct`

Untuk terhubung ke klaster Anda dengan token otentikasi

1. Di konsol Aurora DSQL, pilih cluster yang ingin Anda sambungkan.
2. Pilih Hubungkan.
3. Salin titik akhir dari Endpoint (Host).
4. Pastikan Anda Connect as admin dipilih di bagian Authentication token (Password).
5. Salin token otentikasi yang dihasilkan. Token ini berlaku selama 15 menit.
6. Pada baris perintah sistem operasi, gunakan perintah berikut untuk memulai `psql` dan terhubung ke cluster Anda. Ganti *your_cluster_endpoint* dengan titik akhir cluster yang Anda salin sebelumnya.

   ```
   PGSSLMODE=require \
     psql --dbname postgres \
     --username admin \
     --host your_cluster_endpoint
   ```

   Saat diminta kata sandi, masukkan token otentikasi yang Anda salin sebelumnya. Jika Anda mencoba menyambung kembali menggunakan token kedaluwarsa, permintaan koneksi ditolak. Untuk informasi selengkapnya, lihat Menghasilkan token otentikasi di Amazon Aurora DSQL.

7. Tekan Enter. Anda akan melihat prompt PostgreSQL.

```
postgres=>
```

Jika Anda mendapatkan kesalahan akses ditolak, pastikan identitas IAM Anda memiliki `dsql:DbConnectAdmin` izin. Jika Anda memiliki izin dan terus mendapatkan kesalahan penolakan akses, lihat Memecahkan masalah IAM dan Bagaimana saya bisa memecahkan masalah kesalahan operasi yang ditolak atau tidak sah dengan kebijakan IAM? .

Langkah 3: Jalankan contoh perintah SQL di Aurora DSQL

Uji cluster Aurora DSQL Anda dengan menjalankan pernyataan SQL. Pernyataan contoh berikut memerlukan file data bernama `department-insert-multirow.sql` dan`invoice.csv`, yang dapat Anda unduh dari aurora-dsql-samplesaws-samples/ repositori. GitHub

Untuk menjalankan contoh perintah SQL di Aurora DSQL

1. Buat skema bernama`example`.

```
CREATE SCHEMA example;
```

2. Buat tabel faktur yang menggunakan UUID yang dibuat secara otomatis sebagai kunci utama.

```
CREATE TABLE example.invoice(
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    created timestamp,
    purchaser int,
    amount float);
```

3. Buat indeks sekunder yang menggunakan tabel kosong.

```
CREATE INDEX ASYNC invoice_created_idx on example.invoice(created);
```

4. Buat tabel departemen.

```
CREATE TABLE example.department(id INT PRIMARY KEY UNIQUE, name text, email text);
```

5. Gunakan perintah `psql \include` untuk memuat file bernama `department-insert-multirow.sql` yang Anda unduh dari aurora-dsql-samplesaws-samples/ repositori aktif. GitHub Ganti *my-path* dengan jalur ke salinan lokal Anda.

```
\include my-path/department-insert-multirow.sql
```

6. Gunakan perintah `psql \copy` untuk memuat file bernama `invoice.csv` yang Anda unduh dari aurora-dsql-samplesaws-samples/ repositori aktif. GitHub Ganti *my-path* dengan jalur ke salinan lokal Anda.

```
\copy example.invoice(created, purchaser, amount) from my-path/invoice.csv csv
```

7. Tanyakan departemen dan urutkan berdasarkan total penjualan mereka.

```
SELECT name, sum(amount) AS sum_amount
FROM example.department LEFT JOIN example.invoice ON
 department.id=invoice.purchaser
GROUP BY name
HAVING sum(amount) > 0
ORDER BY sum_amount DESC;
```

Output sampel berikut menunjukkan bahwa Departemen Satu memiliki penjualan terbanyak.

```
           name            |     sum_amount
---------------------------+--------------------
 Example Department One    |  32628.75608634601
 Example Department Three  |  32427.43955110429
 Example Department Eight  | 32256.810987098102
 Example Department Five   |  31391.14891163639
 Example Department Seven  | 31253.236846746757
 Example Department Six    |  29699.06014910414
 Example Department Two    |  29465.58360076501
 Example Department Four   |  28764.19185819191
(8 rows)
```

Langkah 4: Buat cluster Multi-region

Saat Anda membuat cluster Multi-region, Anda menentukan Wilayah berikut:

Wilayah Terpencil

Ini adalah Wilayah tempat Anda membuat cluster kedua. Anda membuat cluster kedua di Region ini dan mengintip ke cluster awal Anda. Aurora DSQL mereplikasi semua penulisan pada cluster awal ke cluster jarak jauh. Anda dapat membaca dan menulis di cluster mana pun.

Wilayah Saksi

Wilayah ini menerima semua data yang ditulis ke cluster Multi-region. Namun, Wilayah saksi tidak meng-host titik akhir klien dan tidak menyediakan akses data pengguna. Jendela terbatas dari log transaksi terenkripsi dipertahankan di Wilayah saksi. Log ini memfasilitasi pemulihan dan mendukung kuorum transaksional jika suatu Wilayah menjadi tidak tersedia.

Contoh berikut menunjukkan cara membuat cluster awal, membuat cluster kedua di Region yang berbeda, dan kemudian mengintip dua cluster untuk membuat cluster Multi-region. Ini juga menunjukkan replikasi penulisan lintas wilayah dan pembacaan yang konsisten dari kedua titik akhir Regional.

Untuk membuat cluster Multi-region

1. Masuk ke AWS Management Console dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql
2. Pada panel navigasi, silakan pilih Klaster.
3. Pilih Buat cluster dan kemudian Multi-Region.
4. (Opsional) Dalam pengaturan Cluster, pilih salah satu opsi berikut untuk klaster awal Anda:
   - Pilih Sesuaikan pengaturan enkripsi (lanjutan) untuk memilih atau membuat pengaturan AWS KMS key.
   - Pilih Aktifkan perlindungan penghapusan untuk mencegah operasi penghapusan klaster Anda. Secara default, perlindungan penghapusan dipilih.
5. Di pengaturan Multi-Region, pilih opsi berikut untuk klaster awal Anda:
   - Di Wilayah Saksi, pilih Wilayah. Saat ini, hanya Wilayah yang berbasis di AS yang didukung untuk menyaksikan Wilayah di klaster Multi-wilayah.
   - (Opsional) Di ARN cluster Remote Region, masukkan ARN untuk cluster yang ada di Wilayah lain. Jika tidak ada klaster yang berfungsi sebagai klaster kedua di klaster Multi-region Anda, selesaikan penyiapan setelah Anda membuat klaster awal.
6. (Opsional) Pilih tag untuk klaster awal Anda.

7. Pilih Buat klaster untuk membuat klaster awal Anda. Jika Anda tidak memasukkan ARN pada langkah sebelumnya, konsol akan menampilkan pemberitahuan tertunda pengaturan Cluster.
8. Dalam pemberitahuan tertunda penyiapan klaster, pilih Selesaikan pengaturan klaster Multi-wilayah. Tindakan ini memulai pembuatan cluster kedua di Wilayah lain.
9. Pilih salah satu opsi berikut untuk cluster kedua Anda:
   - Tambahkan ARN cluster Region jarak jauh — Pilih opsi ini jika ada cluster, dan Anda ingin itu menjadi cluster kedua di cluster Multi-region Anda.
   - Buat cluster di Wilayah lain - Pilih opsi ini untuk membuat cluster kedua. Di Remote Region, pilih Region untuk cluster kedua ini.
10. Pilih Buat cluster di ***your-second-region***, di *your-second-region* mana lokasi cluster kedua Anda. Konsol terbuka di Wilayah kedua Anda.
11. (Opsional) Pilih pengaturan cluster untuk cluster kedua Anda. Misalnya, Anda dapat memilih AWS KMS key.
12. Pilih Buat cluster untuk membuat cluster kedua Anda.
13. Pilih Peer in ***initial-cluster-region***, di mana *initial-cluster-region* adalah Region yang menghosting cluster pertama yang Anda buat.
14. Saat diminta, pilih Konfirmasi. Langkah ini melengkapi pembuatan cluster Multi-region Anda.

Untuk terhubung ke cluster kedua

1. Buka konsol Aurora DSQL dan pilih Region untuk cluster kedua Anda.
2. Pilih Klaster.
3. Pilih baris untuk cluster kedua di cluster Multi-region Anda.
4. Dalam Tindakan, pilih Buka di CloudShell.
5. Pilih Connect sebagai admin.
6. Pilih Luncurkan CloudShell.
7. Pilih Jalankan.
8. Buat skema sampel dengan mengikuti langkah-langkah diLangkah 3: Jalankan contoh perintah SQL di Aurora DSQL.

   Contoh transaksi

Example

```
CREATE SCHEMA example;
CREATE TABLE example.invoice(id UUID PRIMARY KEY DEFAULT gen_random_uuid(), created
 timestamp, purchaser int, amount float);
CREATE INDEX ASYNC invoice_created_idx on example.invoice(created);
CREATE TABLE example.department(id INT PRIMARY KEY UNIQUE, name text, email text);
```

9. Gunakan `psql` `copy` dan `include` perintah untuk memuat data sampel. Untuk informasi selengkapnya, lihat Langkah 3: Jalankan contoh perintah SQL di Aurora DSQL.

```
\copy example.invoice(created, purchaser, amount) from samples/invoice.csv csv
\include samples/department-insert-multirow.sql
```

Untuk melakukan kueri data di klaster kedua dari Region yang menghosting klaster awal Anda

1. Di konsol Aurora DSQL, pilih Region untuk cluster awal Anda.
2. Pilih Klaster.
3. Pilih baris untuk cluster kedua di cluster Multi-region Anda.
4. Dalam Tindakan, pilih Buka di CloudShell.
5. Pilih Connect sebagai admin.
6. Pilih Luncurkan CloudShell.
7. Pilih Jalankan.
8. Kueri data yang Anda masukkan ke dalam cluster kedua.

Example

```
SELECT name, sum(amount) AS sum_amount
FROM example.department
LEFT JOIN example.invoice ON department.id=invoice.purchaser
GROUP BY name
HAVING sum(amount) > 0
ORDER BY sum_amount DESC;
```

Otentikasi dan otorisasi untuk Aurora DSQL

Aurora DSQL menggunakan peran dan kebijakan IAM untuk otorisasi klaster. Anda mengaitkan peran IAM dengan peran database PostgreSQL untuk otorisasi database. Pendekatan ini menggabungkan manfaat dari IAM dengan hak istimewa PostgreSQL. Aurora DSQL menggunakan fitur-fitur ini untuk memberikan otorisasi komprehensif dan kebijakan akses untuk cluster, database, dan data Anda.

Mengelola klaster Anda menggunakan IAM

Untuk mengelola klaster Anda, gunakan IAM untuk otentikasi dan otorisasi:

Autentikasi IAM

Untuk mengautentikasi identitas IAM Anda ketika Anda mengelola cluster Aurora DSQL, Anda harus menggunakan IAM. Anda dapat memberikan otentikasi menggunakan AWS Management Console, AWS CLI, atau AWS SDK.

Otorisasi IAM

Untuk mengelola cluster Aurora DSQL, berikan otorisasi menggunakan tindakan IAM untuk Aurora DSQL. Misalnya, untuk mendeskripsikan klaster, pastikan identitas IAM Anda memiliki izin untuk tindakan IAM`dsql:GetCluster`, seperti pada contoh tindakan kebijakan berikut.

```
{
  "Effect": "Allow",
  "Action": "dsql:GetCluster",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/my-cluster"
}
```

Untuk informasi selengkapnya, lihat Menggunakan tindakan kebijakan IAM untuk mengelola cluster.

Menghubungkan ke klaster Anda menggunakan IAM

Untuk terhubung ke cluster Anda, gunakan IAM untuk otentikasi dan otorisasi:

Autentikasi IAM

Buat token otentikasi sementara menggunakan identitas IAM dengan otorisasi untuk terhubung ke cluster Anda. Untuk mempelajari selengkapnya, lihat Menghasilkan token otentikasi di Amazon Aurora DSQL.

Otorisasi IAM

Berikan tindakan kebijakan IAM berikut ke identitas IAM yang Anda gunakan untuk membuat koneksi ke titik akhir klaster Anda:

- Gunakan `dsql:DbConnectAdmin` jika Anda menggunakan `admin` peran. Aurora DSQL membuat dan mengelola peran ini untuk Anda. Contoh tindakan kebijakan IAM berikut memungkinkan `admin` untuk terhubung ke. *`my-cluster`*

```
{
  "Effect": "Allow",
  "Action": "dsql:DbConnectAdmin",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/my-cluster"
}
```

- Gunakan `dsql:DbConnect` jika Anda menggunakan peran database kustom. Anda membuat dan mengelola peran ini dengan menggunakan perintah SQL dalam database Anda. Contoh tindakan kebijakan IAM berikut memungkinkan peran database kustom untuk terhubung hingga *`my-cluster`* satu jam.

```
{
  "Effect": "Allow",
  "Action": "dsql:DbConnect",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/my-cluster"
}
```

Setelah Anda membuat koneksi, peran Anda diizinkan hingga satu jam untuk koneksi.

Berinteraksi dengan database Anda menggunakan peran database PostgreSQL dan peran IAM

PostgreSQL mengelola izin akses database menggunakan konsep peran. Peran dapat dianggap sebagai pengguna database, atau sekelompok pengguna database, tergantung pada bagaimana

peran tersebut diatur. Anda membuat peran PostgreSQL menggunakan perintah SQL. Untuk mengelola otorisasi tingkat database, berikan izin PostgreSQL ke peran database PostgreSQL Anda.

Aurora DSQL mendukung dua jenis peran database: peran dan `admin` peran khusus. Aurora DSQL secara otomatis membuat `admin` peran yang telah ditentukan untuk Anda di cluster Aurora DSQL Anda. Anda tidak dapat memodifikasi `admin` peran. Ketika Anda terhubung ke database Anda sebagai`admin`, Anda dapat mengeluarkan SQL untuk membuat peran tingkat database baru untuk dikaitkan dengan peran IAM Anda. Untuk membiarkan peran IAM terhubung ke database Anda, kaitkan peran basis data kustom Anda dengan peran IAM Anda.

Autentikasi

Gunakan `admin` peran untuk terhubung ke cluster Anda. Setelah Anda menghubungkan database Anda, gunakan perintah `AWS IAM GRANT` untuk mengaitkan peran database kustom dengan identitas IAM yang diotorisasi untuk terhubung ke cluster, seperti pada contoh berikut.

```
AWS IAM GRANT custom-db-role TO 'arn:aws:iam::account-id:role/iam-role-name';
```

Untuk mempelajari selengkapnya, lihat Mengotorisasi peran database untuk terhubung ke klaster Anda.

Otorisasi

Gunakan `admin` peran untuk terhubung ke cluster Anda. Jalankan perintah SQL untuk mengatur peran database kustom dan memberikan izin. Untuk mempelajari selengkapnya, lihat peran database PostgreSQL dan hak istimewa PostgreSQL dalam dokumentasi PostgreSQL.

Menggunakan tindakan kebijakan IAM dengan Aurora DSQL

Tindakan kebijakan IAM yang Anda gunakan bergantung pada peran yang Anda gunakan untuk menyambung ke klaster: salah satu `admin` atau peran basis data kustom. Kebijakan ini juga tergantung pada tindakan IAM yang diperlukan untuk peran ini.

Menggunakan tindakan kebijakan IAM untuk terhubung ke cluster

Saat Anda terhubung ke klaster Anda dengan peran database default`admin`, gunakan identitas IAM dengan otorisasi untuk melakukan tindakan kebijakan IAM berikut.

```
"dsql:DbConnectAdmin"
```

Saat Anda terhubung ke klaster Anda dengan peran database kustom, pertama-tama kaitkan peran IAM dengan peran database. Identitas IAM yang Anda gunakan untuk terhubung ke klaster Anda harus memiliki otorisasi untuk melakukan tindakan kebijakan IAM berikut.

```
"dsql:DbConnect"
```

Untuk mempelajari lebih lanjut tentang peran basis data kustom, lihatMenggunakan peran database dan otentikasi IAM.

Menggunakan tindakan kebijakan IAM untuk mengelola cluster

Saat mengelola klaster Aurora DSQL Anda, tentukan tindakan kebijakan hanya untuk tindakan yang perlu dilakukan peran Anda. Misalnya, jika peran Anda hanya perlu mendapatkan informasi kluster, Anda dapat membatasi izin peran hanya untuk `ListClusters` izin `GetCluster` dan, seperti dalam kebijakan contoh berikut

JSON

```
{
"Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "dsql:GetCluster",
        "dsql:ListClusters"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/my-cluster"
    }
  ]
}
```

Contoh kebijakan berikut menunjukkan semua tindakan kebijakan IAM yang tersedia untuk mengelola cluster.

JSON

```
{
```

```
"Version" : "2012-10-17",
  "Statement" : [
    {
      "Effect" : "Allow",
      "Action" : [
        "dsql:CreateCluster",
        "dsql:GetCluster",
        "dsql:UpdateCluster",
        "dsql:DeleteCluster",
        "dsql:ListClusters",
        "dsql:TagResource",
        "dsql:ListTagsForResource",
        "dsql:UntagResource"
      ],
      "Resource" : "*"
    }
  ]
}
```

Mencabut otorisasi menggunakan IAM dan PostgreSQL

Anda dapat mencabut izin untuk peran IAM untuk mengakses peran tingkat database Anda:

Mencabut otorisasi admin untuk terhubung ke cluster

Untuk mencabut otorisasi untuk terhubung ke klaster Anda dengan `admin` peran tersebut, cabut akses identitas IAM ke. `dsql:DbConnectAdmin` Edit kebijakan IAM atau lepaskan kebijakan dari identitas.

Setelah mencabut otorisasi koneksi dari identitas IAM, Aurora DSQL menolak semua upaya koneksi baru dari identitas IAM tersebut. Koneksi aktif apa pun yang menggunakan identitas IAM mungkin tetap diotorisasi selama durasi koneksi. Untuk informasi selengkapnya tentang durasi koneksi, lihat Kuota dan batas.

Mencabut otorisasi peran khusus untuk terhubung ke cluster

Untuk mencabut akses ke peran database selain`admin`, cabut akses identitas IAM ke. `dsql:DbConnect` Edit kebijakan IAM atau lepaskan kebijakan dari identitas.

Anda juga dapat menghapus asosiasi antara peran database dan IAM dengan menggunakan perintah `AWS IAM REVOKE` dalam database Anda. Untuk mempelajari lebih lanjut tentang mencabut akses dari peran database, lihat. Mencabut otorisasi database dari peran IAM

Anda tidak dapat mengelola izin dari peran `admin` database yang telah ditentukan sebelumnya. Untuk mempelajari cara mengelola izin untuk peran database kustom, lihat hak istimewa PostgreSQL. Modifikasi hak istimewa berlaku pada transaksi berikutnya setelah Aurora DSQL berhasil melakukan transaksi modifikasi.

Menghasilkan token otentikasi di Amazon Aurora DSQL

Untuk terhubung ke Amazon Aurora DSQL dengan klien SQL, buat token otentikasi untuk digunakan sebagai kata sandi. Token ini hanya digunakan untuk mengautentikasi koneksi. Setelah koneksi dibuat, koneksi tetap valid bahkan jika token otentikasi kedaluwarsa.

Jika Anda membuat token otentikasi menggunakan AWS konsol, token secara otomatis kedaluwarsa dalam satu jam secara default. Jika Anda menggunakan AWS CLI atau SDKs untuk membuat token, defaultnya adalah 15 menit. Durasi maksimum adalah 604.800 detik, yaitu satu minggu. Untuk terhubung ke Aurora DSQL dari klien Anda lagi, Anda dapat menggunakan token otentikasi yang sama jika belum kedaluwarsa, atau Anda dapat membuat yang baru.

Untuk memulai membuat token, buat kebijakan IAM dan cluster di Aurora DSQL. Kemudian gunakan AWS konsol, AWS CLI, atau AWS SDKs untuk menghasilkan token.

Minimal, Anda harus memiliki izin IAM yang tercantumMenghubungkan ke klaster Anda menggunakan IAM, tergantung pada peran database yang Anda gunakan untuk terhubung.

Topik

- Gunakan AWS konsol untuk menghasilkan token otentikasi di Aurora DSQL
- Gunakan AWS CloudShell untuk menghasilkan token otentikasi di Aurora DSQL
- Gunakan AWS CLI untuk menghasilkan token otentikasi di Aurora DSQL
- Gunakan SDKs untuk menghasilkan token di Aurora DSQL

Gunakan AWS konsol untuk menghasilkan token otentikasi di Aurora DSQL

Aurora DSQL mengautentikasi pengguna dengan token daripada kata sandi. Anda dapat menghasilkan token dari konsol.

Untuk menghasilkan token otentikasi

1. Masuk ke AWS Management Console dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql

2. Pilih ID cluster cluster yang ingin Anda buat token otentikasi. Jika Anda belum membuat cluster, ikuti langkah-langkah di Langkah 1: Buat cluster Aurora DSQL Single-region atauLangkah 4: Buat cluster Multi-region.
3. Pilih Connect dan kemudian pilih Get Token.
4. Pilih apakah Anda ingin terhubung sebagai `admin` atau dengan peran basis data kustom.
5. Salin token otentikasi yang dihasilkan dan gunakan untukMengakses Aurora DSQL menggunakan klien SQL.

Untuk mempelajari lebih lanjut tentang peran basis data kustom dan IAM di Aurora DSQL, lihat. Autentikasi dan otorisasi

Gunakan AWS CloudShell untuk menghasilkan token otentikasi di Aurora DSQL

Sebelum Anda dapat membuat token otentikasi menggunakan AWS CloudShell, pastikan bahwa Anda membuat cluster Aurora DSQL.

Untuk menghasilkan token otentikasi menggunakan AWS CloudShell

1. Masuk ke AWS Management Console dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql
2. Di kiri bawah AWS konsol, pilih AWS CloudShell.
3. Jalankan perintah berikut untuk menghasilkan token otentikasi untuk `admin` peran tersebut. Ganti *`us-east-1`* dengan Wilayah Anda dan *`your_cluster_endpoint`* dengan titik akhir cluster Anda sendiri.

> **Note**
>
> Jika Anda tidak terhubung sebagai`admin`, gunakan sebagai `generate-db-connect-auth-token` gantinya.

```
aws dsql generate-db-connect-admin-auth-token \
  --expires-in 3600 \
  --region us-east-1 \
  --hostname your_cluster_endpoint
```

Jika Anda mengalami masalah, lihat Memecahkan Masalah IAM dan Bagaimana cara memecahkan masalah akses ditolak atau kesalahan operasi yang tidak sah dengan kebijakan IAM? .

4. Gunakan perintah berikut untuk digunakan `psql` untuk memulai koneksi ke cluster Anda.

```
PGSSLMODE=require \
psql --dbname postgres \
  --username admin \
  --host cluster_endpoint
```

5. Anda akan melihat prompt untuk memberikan kata sandi. Salin token yang Anda buat, dan pastikan Anda tidak menyertakan spasi atau karakter tambahan. Tempelkan ke prompt berikut dari`psql`.

```
Password for user admin:
```

6. Tekan Enter. Anda akan melihat prompt PostgreSQL.

```
postgres=>
```

Jika Anda mendapatkan kesalahan akses ditolak, pastikan identitas IAM Anda memiliki `dsql:DbConnectAdmin` izin. Jika Anda memiliki izin dan terus mendapatkan kesalahan penolakan akses, lihat Memecahkan masalah IAM dan Bagaimana saya bisa memecahkan masalah kesalahan operasi yang ditolak atau tidak sah dengan kebijakan IAM? .

Untuk mempelajari lebih lanjut tentang peran basis data kustom dan IAM di Aurora DSQL, lihat. Autentikasi dan otorisasi

Gunakan AWS CLI untuk menghasilkan token otentikasi di Aurora DSQL

Ketika cluster Anda`ACTIVE`, Anda dapat menghasilkan token otentikasi pada CLI dengan menggunakan `aws dsql` perintah. Gunakan salah satu dari teknik berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan `generate-db-connect-admin-auth-token` opsi.
- Jika Anda terhubung dengan peran basis data kustom, gunakan `generate-db-connect-auth-token` opsi.

Contoh berikut menggunakan atribut berikut untuk menghasilkan token otentikasi untuk `admin` peran tersebut.

- *`your_cluster_endpoint`*— Titik akhir cluster. Ini mengikuti format*`your_cluster_identifier`*`.dsql.`*`region`*`.on.aws`, seperti pada contoh`01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`.
- *`region`*— The Wilayah AWS, seperti `us-east-2` atau`us-east-1`.

Contoh berikut mengatur waktu kedaluwarsa token untuk kedaluwarsa dalam 3600 detik (1 jam).

Linux and macOS

```
aws dsql generate-db-connect-admin-auth-token \
  --region region \
  --expires-in 3600 \
  --hostname your_cluster_endpoint
```

Windows

```
aws dsql generate-db-connect-admin-auth-token ^
  --region=region ^
  --expires-in=3600 ^
  --hostname=your_cluster_endpoint
```

Gunakan SDKs untuk menghasilkan token di Aurora DSQL

Anda dapat membuat token otentikasi untuk klaster Anda saat berada dalam `ACTIVE` status. Contoh SDK menggunakan atribut berikut untuk menghasilkan token autentikasi untuk peran tersebut`admin`:

- *`your_cluster_endpoint`*(atau*`yourClusterEndpoint`*) — Titik akhir cluster Aurora DSQL Anda. Format penamaan adalah*`your_cluster_identifier`*`.dsql.`*`region`*`.on.aws`, seperti pada contoh`01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`.
- *`region`*(atau*`RegionEndpoint`*) — Wilayah AWS Di mana cluster Anda berada, seperti `us-east-2` atau`us-east-1`.

Python SDK

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan`generate_db_connect_admin_auth_token`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`generate_connect_auth_token`.

```
def generate_token(your_cluster_endpoint, region):
    client = boto3.client("dsql", region_name=region)
    # use `generate_db_connect_auth_token` instead if you are not connecting as
 admin.
    token = client.generate_db_connect_admin_auth_token(your_cluster_endpoint,
 region)
    print(token)
    return token
```

C++ SDK

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan`GenerateDBConnectAdminAuthToken`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`GenerateDBConnectAuthToken`.

```
#include <aws/core/Aws.h>
#include <aws/dsql/DSQLClient.h>
#include <iostream>

using namespace Aws;
using namespace Aws::DSQL;

std::string generateToken(String yourClusterEndpoint, String region) {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQLClient client{clientConfig};
    std::string token = "";

    // If you are not using the admin role to connect, use
 GenerateDBConnectAuthToken instead
    const auto presignedString =
 client.GenerateDBConnectAdminAuthToken(yourClusterEndpoint, region);
    if (presignedString.IsSuccess()) {
        token = presignedString.GetResult();
    } else {
        std::cerr << "Token generation failed." << std::endl;
    }

    std::cout << token << std::endl;

    Aws::ShutdownAPI(options);
    return token;
}
```

JavaScript SDK

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan`getDbConnectAdminAuthToken`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`getDbConnectAuthToken`.

```
import { DsqlSigner } from "@aws-sdk/dsql-signer";

async function generateToken(yourClusterEndpoint, region) {
  const signer = new DsqlSigner({
    hostname: yourClusterEndpoint,
    region,
  });
  try {
    // Use `getDbConnectAuthToken` if you are _not_ logging in as the `admin` user
    const token = await signer.getDbConnectAdminAuthToken();
    console.log(token);
    return token;
  } catch (error) {
      console.error("Failed to generate token: ", error);
      throw error;
  }
}
```

Java SDK

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan`generateDbConnectAdminAuthToken`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`generateDbConnectAuthToken`.

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.services.dsql.DsqlUtilities;
import software.amazon.awssdk.regions.Region;

public class GenerateAuthToken {
    public static String generateToken(String yourClusterEndpoint, Region region) {
        DsqlUtilities utilities = DsqlUtilities.builder()
                .region(region)
                .credentialsProvider(DefaultCredentialsProvider.create())
                .build();

        // Use `generateDbConnectAuthToken` if you are _not_ logging in as `admin`
 user
        String token = utilities.generateDbConnectAdminAuthToken(builder -> {
            builder.hostname(yourClusterEndpoint)
                    .region(region);
        });

        System.out.println(token);
        return token;
    }
}
```

Rust SDK

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan`db_connect_admin_auth_token`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`db_connect_auth_token`.

```
use aws_config::{BehaviorVersion, Region};
use aws_sdk_dsql::auth_token::{AuthTokenGenerator, Config};

async fn generate_token(your_cluster_endpoint: String, region: String) -> String {
    let sdk_config = aws_config::load_defaults(BehaviorVersion::latest()).await;
    let signer = AuthTokenGenerator::new(
        Config::builder()
            .hostname(&your_cluster_endpoint)
            .region(Region::new(region))
            .build()
            .unwrap(),
    );

    // Use `db_connect_auth_token` if you are _not_ logging in as `admin` user
    let token = signer.db_connect_admin_auth_token(&sdk_config).await.unwrap();
    println!("{}", token);
    token.to_string()
}
```

Ruby SDK

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan`generate_db_connect_admin_auth_token`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`generate_db_connect_auth_token`.

```
require 'aws-sdk-dsql'

def generate_token(your_cluster_endpoint, region)
  credentials = Aws::SharedCredentials.new()

  begin
      token_generator = Aws::DSQL::AuthTokenGenerator.new({
          :credentials => credentials
      })

      # if you're not using admin role, use generate_db_connect_auth_token instead
      token = token_generator.generate_db_connect_admin_auth_token({
          :endpoint => your_cluster_endpoint,
```

```
          :region => region
      })
  rescue => error
    puts error.full_message
  end
end
```

.NET

> **Note**
>
> SDK resmi untuk .NET tidak menyertakan panggilan API bawaan untuk menghasilkan token otentikasi untuk Aurora DSQL. Sebaliknya, Anda harus menggunakan`DSQLAuthTokenGenerator`, yang merupakan kelas utilitas. Contoh kode berikut menunjukkan cara menghasilkan token otentikasi untuk.NET.

Anda dapat membuat token dengan cara berikut:

- Jika Anda terhubung dengan `admin` peran, gunakan`DbConnectAdmin`.
- Jika Anda terhubung dengan peran basis data kustom, gunakan`DbConnect`.

Contoh berikut menggunakan kelas `DSQLAuthTokenGenerator` utilitas untuk menghasilkan token otentikasi untuk pengguna dengan `admin` peran. Ganti *`insert-dsql-cluster-endpoint`* dengan titik akhir cluster Anda.

```
using Amazon;
using Amazon.DSQL.Util;
using Amazon.Runtime;

var yourClusterEndpoint = "insert-dsql-cluster-endpoint";

AWSCredentials credentials = FallbackCredentialsFactory.GetCredentials();

var token = DSQLAuthTokenGenerator.GenerateDbConnectAdminAuthToken(credentials,
 RegionEndpoint.USEast1, yourClusterEndpoint);

Console.WriteLine(token);
```

Golang

> **Note**
>
> Golang SDK tidak menyediakan metode bawaan untuk menghasilkan token yang telah ditandatangani sebelumnya. Anda harus secara manual membangun permintaan yang ditandatangani, seperti yang ditunjukkan dalam contoh kode berikut.

Dalam contoh kode berikut, tentukan `action` berdasarkan pengguna PostgreSQL:

- Jika Anda terhubung dengan `admin` peran, gunakan `DbConnectAdmin` tindakan.
- Jika Anda terhubung dengan peran basis data kustom, gunakan `DbConnect` tindakan tersebut.

Selain *`yourClusterEndpoint`* dan*`region`*, contoh berikut menggunakan*`action`*. Tentukan *`action`* berdasarkan pengguna PostgreSQL.

```
func GenerateDbConnectAdminAuthToken(yourClusterEndpoint string, region
 string, action string) (string, error) {
 // Fetch credentials
 sess, err := session.NewSession()
 if err != nil {
  return "", err
 }

 creds, err := sess.Config.Credentials.Get()
 if err != nil {
  return "", err
 }
 staticCredentials := credentials.NewStaticCredentials(
  creds.AccessKeyID,
  creds.SecretAccessKey,
  creds.SessionToken,
 )

 // The scheme is arbitrary and is only needed because validation of the URL
 requires one.
 endpoint := "https://" + yourClusterEndpoint
 req, err := http.NewRequest("GET", endpoint, nil)
 if err != nil {
  return "", err
 }
 values := req.URL.Query()
 values.Set("Action", action)
 req.URL.RawQuery = values.Encode()

 signer := v4.Signer{
  Credentials: staticCredentials,
 }
 _, err = signer.Presign(req, nil, "dsql", region, 15*time.Minute, time.Now())
 if err != nil {
  return "", err
 }

 url := req.URL.String()[len("https://"):]

 return url, nil
}
```

Menggunakan peran database dan otentikasi IAM

Aurora DSQL mendukung otentikasi menggunakan peran IAM dan pengguna IAM. Anda dapat menggunakan salah satu metode untuk mengautentikasi dan mengakses database Aurora DSQL.

Peran IAM

Peran IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin tertentu tetapi tidak terkait dengan orang tertentu. Menggunakan peran IAM memberikan kredensil keamanan sementara. Anda dapat mengambil peran IAM untuk sementara dalam beberapa cara:

- Dengan beralih peran di AWS Management Console
- Dengan memanggil operasi AWS CLI atau AWS API
- Dengan menggunakan URL kustom

Setelah mengambil peran, Anda dapat mengakses Aurora DSQL menggunakan kredensyal sementara peran. Untuk informasi selengkapnya tentang metode penggunaan peran, lihat Identitas IAM di panduan pengguna IAM.

Pengguna IAM

Pengguna IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus dan dikaitkan dengan satu orang atau aplikasi. Pengguna IAM memiliki kredensyal jangka panjang seperti kata sandi dan kunci akses yang dapat digunakan untuk mengakses Aurora DSQL.

> **Note**
>
> Untuk menjalankan perintah SQL dengan otentikasi IAM, Anda dapat menggunakan peran IAM ARNs atau pengguna ARNs IAM dalam contoh di bawah ini.

Mengotorisasi peran database untuk terhubung ke klaster Anda

Buat peran IAM dan berikan otorisasi koneksi dengan tindakan kebijakan IAM:. `dsql:DbConnect`

Kebijakan IAM juga harus memberikan izin untuk mengakses sumber daya klaster. Gunakan wildcard (*) atau ikuti petunjuk di Menggunakan kunci kondisi IAM dengan Amazon Aurora DSQL.

Mengotorisasi peran database untuk menggunakan SQL dalam database Anda

Anda harus menggunakan peran IAM dengan otorisasi untuk terhubung ke cluster Anda.

1. Connect ke cluster Aurora DSQL Anda menggunakan utilitas SQL.

   Gunakan peran `admin` database dengan identitas IAM yang diotorisasi untuk tindakan IAM untuk terhubung `dsql:DbConnectAdmin` ke cluster Anda.
2. Buat peran database baru, pastikan untuk menentukan `WITH LOGIN` opsi.

   ```
   CREATE ROLE example WITH LOGIN;
   ```
3. Kaitkan peran database dengan peran IAM ARN.

   ```
   AWS IAM GRANT example TO 'arn:aws:iam::012345678912:role/example';
   ```
4. Berikan izin tingkat database ke peran database

   Contoh berikut menggunakan `GRANT` perintah untuk memberikan otorisasi dalam database.

   ```
   GRANT USAGE ON SCHEMA myschema TO example;
   GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA myschema TO example;
   ```

Untuk informasi selengkapnya, lihat PostgreSQL dan `GRANT` PostgreSQL Privileges dalam dokumentasi PostgreSQL.

Melihat IAM ke pemetaan peran database

Untuk melihat pemetaan antara peran IAM dan peran database, kueri tabel sistem. `sys.iam_pg_role_mappings`

```
SELECT * FROM sys.iam_pg_role_mappings;
```

Contoh output:

```
 iam_oid |                  arn                   | pg_role_oid | pg_role_name |
grantor_pg_role_oid | grantor_pg_role_name
```

```
---------+----------------------------------------+-------------+--------------
+---------------------+----------------------
   26398 | arn:aws:iam::012345678912:role/example |       26396 | example      |
        15579 | admin
(1 row)
```

Tabel ini menunjukkan semua pemetaan antara peran IAM (diidentifikasi oleh ARN mereka) dan peran database PostgreSQL.

Mencabut otorisasi database dari peran IAM

Untuk mencabut otorisasi database, gunakan operasi. `AWS IAM REVOKE`

```
AWS IAM REVOKE example FROM 'arn:aws:iam::012345678912:role/example';
```

Untuk mempelajari lebih lanjut tentang mencabut otorisasi, lihat. Mencabut otorisasi menggunakan IAM dan PostgreSQL

Aurora DSQL dan PostgreSQL

Aurora DSQL adalah database relasional terdistribusi yang kompatibel dengan PostgreSQL yang dirancang untuk beban kerja transaksional. Aurora DSQL menggunakan komponen PostgreSQL inti seperti parser, planner, optimizer, dan sistem tipe.

Desain Aurora DSQL memastikan bahwa semua sintaks PostgreSQL yang didukung memberikan perilaku yang kompatibel dan menghasilkan hasil kueri yang identik. Misalnya, Aurora DSQL menyediakan konversi tipe, operasi aritmatika, dan presisi numerik dan skala yang identik dengan PostgreSQL. Setiap penyimpangan didokumentasikan.

Aurora DSQL juga memperkenalkan kemampuan canggih seperti kontrol konkurensi optimis dan manajemen skema terdistribusi. Dengan fitur-fitur ini, Anda dapat menggunakan perkakas PostgreSQL yang sudah dikenal sambil memanfaatkan kinerja dan skalabilitas yang diperlukan untuk aplikasi terdistribusi modern, cloud-native, dan terdistribusi.

Sorotan kompatibilitas PostgreSQL

Aurora DSQL saat ini didasarkan pada PostgreSQL versi 16. Kompatibilitas utama meliputi yang berikut:

Protokol kawat

Aurora DSQL menggunakan protokol kabel PostgreSQL v3 standar. Ini memungkinkan integrasi dengan klien PostgreSQL standar, driver, dan alat. Misalnya, Aurora DSQL kompatibel dengan`psql`,, dan. `pgjdbc` `psycopg`

Kompatibilitas SQL

Aurora DSQL mendukung berbagai ekspresi dan fungsi PostgreSQL standar yang biasa digunakan dalam beban kerja transaksional. Ekspresi SQL yang didukung menghasilkan hasil yang identik dengan PostgreSQL, termasuk yang berikut:

- Penanganan nulls
- Urutkan perilaku urutan
- Skala dan presisi untuk operasi numerik
- Kesetaraan untuk operasi string

Untuk informasi selengkapnya, lihat Kompatibilitas fitur SQL di Aurora DSQL.

Manajemen transaksi

Aurora DSQL mempertahankan karakteristik utama PostgreSQL, seperti transaksi ACID dan tingkat isolasi yang setara dengan PostgreSQL Repeatable Read. Untuk informasi selengkapnya, lihat Kontrol konkurensi di Aurora DSQL.

Perbedaan arsitektur utama

Desain Aurora DSQL yang didistribusikan dan tidak dibagikan menghasilkan beberapa perbedaan mendasar dari PostgreSQL tradisional. Perbedaan ini merupakan bagian integral dari arsitektur Aurora DSQL dan memberikan banyak manfaat kinerja dan skalabilitas. Perbedaan utama meliputi:

Kontrol Konkurensi Optimis (OCC)

Aurora DSQL menggunakan model kontrol konkurensi yang optimis. Pendekatan bebas kunci ini mencegah transaksi memblokir satu sama lain, menghilangkan kebuntuan, dan memungkinkan eksekusi paralel throughput tinggi. Fitur-fitur ini membuat Aurora DSQL sangat berharga untuk aplikasi yang membutuhkan kinerja yang konsisten pada skala besar. Untuk contoh lebih lanjut, lihatKontrol konkurensi di Aurora DSQL.

Operasi DDL asinkron

Aurora DSQL menjalankan operasi DDL secara asinkron, yang memungkinkan pembacaan dan penulisan tanpa gangguan selama perubahan skema. Arsitektur terdistribusi memungkinkan Aurora DSQL untuk melakukan tindakan berikut:

- Jalankan operasi DDL sebagai tugas latar belakang, meminimalkan gangguan.
- Mengkoordinasikan perubahan katalog sebagai transaksi terdistribusi yang sangat konsisten. Ini memastikan visibilitas atom di semua node, bahkan selama kegagalan atau operasi bersamaan.
- Beroperasi dengan cara yang terdistribusi penuh dan tanpa pemimpin di beberapa Availability Zone dengan lapisan komputasi dan penyimpanan terpisah.

Untuk informasi selengkapnya, lihat DDL dan transaksi terdistribusi di Aurora DSQL.

Kompatibilitas fitur SQL di Aurora DSQL

Aurora DSQL dan PostgreSQL mengembalikan hasil yang identik untuk semua query SQL. Perhatikan bahwa Aurora DSQL berbeda dari PostgreSQL tanpa klausa. `ORDER BY` Pada bagian berikut, pelajari tentang dukungan Aurora DSQL untuk tipe data PostgreSQL dan perintah SQL.

Topik

Tipe data yang didukung di Aurora DSQL

Aurora DSQL mendukung subset dari jenis PostgreSQL umum.

Topik

Jenis data numerik

Aurora DSQL mendukung tipe data numerik PostgreSQL berikut.

Nama	Alias	Rentang dan presisi	Ukuran penyimpanan	Dukungan indeks
`smallint`	`int2`	-32768 ke +32767	2 byte	Ya
`integer`	`int`, `int4`	-2147483648 ke +2147483647	4 byte	Ya

Nama	Alias	Rentang dan presisi	Ukuran penyimpanan	Dukungan indeks
`bigint`	`int8`	-9223372036854775808 ke +9223372036854775807	8 byte	Ya
`real`	`float4`	6 digit desimal presisi	4 byte	Ya
`double precision`	`float8`	15 digit desimal presisi	8 byte	Ya
`numeric [ (p, s) ]`	`decimal [ (p, s) ]` `dec[ (p,s)]`	Numerik yang tepat dari presisi yang dapat dipilih. Presisi maksimum adalah 38 dan skala maksimum adalah 37. [1] Defaultnya adalah`numeric (18,6)`.	8 byte+2 byte per digit presisi. Ukuran maksimum adalah 27 byte.	Tidak

[1]— Jika Anda tidak secara eksplisit menentukan ukuran saat Anda menjalankan `CREATE TABLE` atau, `ALTER TABLE ADD COLUMN` Aurora DSQL memberlakukan default. Aurora DSQL menerapkan batasan saat Anda menjalankan atau pernyataan. `INSERT UPDATE`

Tipe data karakter

Aurora DSQL mendukung tipe data karakter PostgreSQL berikut.

Nama	Alias	Deskripsi	Batas Aurora DSQL	Ukuran penyimpanan	Dukungan indeks
`character [ (n) ]`	`char [ (n) ]`	String karakter dengan panjang tetap	4096 byte [1]	Variabel hingga 4100 byte	Ya
`character varying [ (n) ]`	`varchar [ (n) ]`	String karakter panjang variabel	65535 byte [1]	Variabel hingga 65539 byte	Ya

Nama	Alias	Deskripsi	Batas Aurora DSQL	Ukuran penyimpanan	Dukungan indeks
`bpchar [ (n) ]`		Jika panjang tetap, ini adalah alias untuk`char`. Jika panjang variabel, ini adalah alias untuk`varchar`, di mana spasi tambahan secara semantik tidak signifikan.	4096 byte 1	Variabel hingga 4100 byte	Ya
`text`		String karakter panjang variabel	1 MiB 1	Variabel hingga 1 MiB	Ya

1— Jika Anda tidak secara eksplisit menentukan ukuran ketika Anda menjalankan `CREATE TABLE` atau`ALTER TABLE ADD COLUMN`, maka Aurora DSQL memberlakukan default. Aurora DSQL menerapkan batasan saat Anda menjalankan atau pernyataan. `INSERT UPDATE`

Jenis data tanggal dan waktu

Aurora DSQL mendukung tipe data tanggal dan waktu PostgreSQL berikut.

Nama	Alias	Deskripsi	Kisaran	Resolusi	Ukuran penyimpanan	Dukungan indeks
`date`		Tanggal kalender (tahun, bulan, hari)	4713 SM — 5874897 IKLAN	1 hari	4 byte	Ya
`time [ (p) ] [ without time zone ]`	`time`	Waktu hari, tanpa zona waktu	0 — 1	1 mikrodetik	8 byte	Ya

Nama	Alias	Deskripsi	Kisaran	Resolusi	Ukuran penyimpanan	Dukungan indeks
`time [ (p) ] with time zone`	`time`	waktu hari, termasuk zona waktu	00:00:00 +1559 — 24:00:00 —1559	1 mikrodetik	12 byte	Tidak
`timestamp [ (p) ] [ without time zone ]`		Tanggal dan waktu, tanpa zona waktu	4713 SM — 294276 IKLAN	1 mikrodetik	8 byte	Ya
`timestamp [ (p) ] with time zone`	`timestamptz`	Tanggal dan waktu, termasuk zona waktu	4713 SM — 294276 IKLAN	1 mikrodetik	8 byte	Ya
`interval [ fields ] [ (p) ]`		Rentang waktu	-178000000 tahun — 178000000 tahun	1 mikrodetik	16 byte	Tidak

Jenis data lain-lain

Aurora DSQL mendukung berbagai jenis data PostgreSQL berikut.

Nama	Alias	Deskripsi	Batas Aurora DSQL	Ukuran penyimpanan	Dukungan indeks
`boolean`	`bool`	Logis Boolean (benar/salah)		1 byte	Ya
`bytea`		Data biner (“array byte”)	1 MiB 1	Variabel hingga batas 1 MiB	Tidak

Nama	Alias	Deskripsi	Batas Aurora DSQL	Ukuran penyimpanan	Dukungan indeks
`UUID`		Pengidentifikasi unik secara universal		16 byte	Ya

[1] — Jika Anda tidak secara eksplisit menentukan ukuran ketika Anda menjalankan `CREATE TABLE` atau`ALTER TABLE ADD COLUMN`, maka Aurora DSQL memberlakukan default. Aurora DSQL menerapkan batasan saat Anda menjalankan atau pernyataan. `INSERT` `UPDATE`

Jenis data runtime kueri

Query tipe data runtime adalah tipe data internal yang digunakan pada waktu eksekusi query. Tipe ini berbeda dari tipe yang kompatibel dengan PostgreSQL seperti `varchar` dan `integer` yang Anda tentukan dalam skema Anda. Sebagai gantinya, jenis ini adalah representasi runtime yang digunakan Aurora DSQL saat memproses kueri.

Tipe data berikut hanya didukung selama runtime kueri:

Jenis array

Aurora DSQL mendukung array dari tipe data yang didukung. Misalnya, Anda dapat memiliki array bilangan bulat. Fungsi `string_to_array` membagi string ke dalam array gaya PostgreSQL dengan pembatas koma () `,` seperti yang ditunjukkan pada contoh berikut. Anda dapat menggunakan array dalam ekspresi, output fungsi, atau perhitungan sementara selama eksekusi kueri.

```
SELECT string_to_array('1,2', ',');
```

Fungsi mengembalikan respon yang mirip dengan berikut ini:

```
 string_to_array
-----------------
 {1,2}
(1 row)
```

jenis inet

Tipe data mewakili IPv4, alamat IPv6 host, dan subnetnya. Jenis ini berguna saat mengurai log, memfilter pada subnet IP, atau melakukan perhitungan jaringan dalam kueri. Untuk informasi selengkapnya, lihat inet di dokumentasi PostgreSQL.

SQL yang didukung untuk Aurora DSQL

Aurora DSQL mendukung berbagai fitur inti PostgreSQL SQL. Di bagian berikut, Anda dapat mempelajari tentang dukungan ekspresi PostgreSQL umum. Daftar ini bukanlah daftar lengkap.

Perintah **SELECT**

Aurora DSQL mendukung klausul perintah berikut. `SELECT`

Klausul utama	Klausul yang didukung
`FROM`	
`GROUP BY`	`ALL`, `DISTINCT`
`ORDER BY`	`ASC`, `DESC`, `NULLS`
`LIMIT`	
`DISTINCT`	
`HAVING`	
`USING`	
`WITH`(ekspresi tabel umum)	
`INNER JOIN`	`ON`
`OUTER JOIN`	`LEFT`, `RIGHT`, `FULL`, `ON`
`CROSS JOIN`	`ON`
`UNION`	`ALL`

Klausul utama	Klausul yang didukung
`INTERSECT`	`ALL`
`EXCEPT`	`ALL`
`OVER`	`RANK ()`, `PARTITION BY`
`FOR UPDATE`	

Bahasa Definisi Data (DDL)

Aurora DSQL mendukung perintah PostgreSQL DDL berikut.

Perintah	Klausul Utama	Klausul yang Didukung
`CREATE`	`TABLE`	Untuk informasi tentang sintaks `CREATE TABLE` perintah yang didukung, lihat`CREATE TABLE`.
`ALTER`	`TABLE`	Untuk informasi tentang sintaks `ALTER TABLE` perintah yang didukung, lihat`ALTER TABLE`.
`DROP`	`TABLE`	
`CREATE`	`[UNIQUE] INDEX ASYNC`	Anda dapat menggunakan perintah ini dengan parameter berikut:`ON`,`NULLS FIRST`,`NULLS LAST`. Untuk informasi tentang sintaks `CREATE INDEX ASYNC` perintah yang didukung, lihatIndeks asinkron di Aurora DSQL.
`DROP`	`INDEX`	

Perintah	Klausul Utama	Klausul yang Didukung
CREATE	VIEW	Untuk informasi selengkapnya tentang sintaks CREATE VIEW perintah yang didukung, lihatCREATE VIEW.
ALTER	VIEW	Untuk informasi tentang sintaks ALTER VIEW perintah yang didukung, lihatALTER VIEW.
DROP	VIEW	Untuk informasi tentang sintaks DROP VIEW perintah yang didukung, lihatDROP VIEW.
CREATE	ROLE, WITH	
CREATE	FUNCTION	LANGUAGE SQL
CREATE	DOMAIN	

Bahasa Manipulasi Data (DML)

Aurora DSQL mendukung perintah PostgreSQL DHTML berikut.

Perintah	Klausul utama	Klausul yang didukung
INSERT	INTO	VALUES SELECT
UPDATE	SET	WHERE (SELECT) FROM, WITH
DELETE	FROM	USING, WHERE

Bahasa Kontrol Data (DCL)

Aurora DSQL mendukung perintah PostgreSQL DCL berikut.

Perintah	Klausul yang didukung
`GRANT`	`ON`, `TO`
`REVOKE`	`ON`, `FROM`, `CASCADE`, `RESTRICT`

Bahasa Kontrol Transaksi (TCL)

Aurora DSQL mendukung perintah PostgreSQL TCL berikut.

Perintah	Klausul yang didukung
`COMMIT`	

`BEGIN`	`[WORK | TRANSACTION ]`
	
	`[READ ONLY | READ WRITE]`

Perintah utilitas

Aurora DSQL mendukung perintah utilitas PostgreSQL berikut:

- `EXPLAIN`
- `ANALYZE`(nama relasi saja)

Subset yang didukung dari perintah SQL di Aurora DSQL

Bagian PostgreSQL ini memberikan informasi rinci tentang ekspresi yang didukung, dengan fokus pada perintah dengan set parameter dan subperintah yang luas. Misalnya, CREATE TABLE di PostgreSQL menawarkan banyak klausa dan parameter. Bagian ini menjelaskan semua elemen sintaks PostgreSQL yang didukung Aurora DSQL untuk perintah ini.

Topik

- CREATE TABLE
- ALTER TABLE

- CREATE VIEW
- ALTER VIEW
- DROP VIEW

CREATE TABLE

CREATE TABLEmendefinisikan tabel baru.

```
CREATE TABLE [ IF NOT EXISTS ] table_name ( [
  { column_name data_type [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE source_table [ like_option ... ] }
    [, ... ]
] )

where column_constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression )|
  DEFAULT default_expr |
  GENERATED ALWAYS AS ( generation_expr ) STORED |
  UNIQUE [ NULLS [ NOT ] DISTINCT ] index_parameters |
  PRIMARY KEY index_parameters |

and table_constraint is:

[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
  UNIQUE [ NULLS [ NOT ] DISTINCT ] ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |

and like_option is:

{ INCLUDING | EXCLUDING } { COMMENTS | CONSTRAINTS | DEFAULTS | GENERATED | IDENTITY |
 INDEXES | STATISTICS | ALL }

index_parameters in UNIQUE, and PRIMARY KEY constraints are:
[ INCLUDE ( column_name [, ... ] ) ]
```

ALTER TABLE

`ALTER TABLE`mengubah definisi tabel.

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    action [, ... ]
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME [ COLUMN ] column_name TO new_column_name
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME CONSTRAINT constraint_name TO new_constraint_name
ALTER TABLE [ IF EXISTS ] name
    RENAME TO new_name
ALTER TABLE [ IF EXISTS ] name
    SET SCHEMA new_schema

where action is one of:

    ADD [ COLUMN ] [ IF NOT EXISTS ] column_name data_type
    OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER | SESSION_USER }
```

CREATE VIEW

`CREATE VIEW`mendefinisikan pandangan persisten baru. Aurora DSQL tidak mendukung tampilan sementara; hanya tampilan permanen yang didukung.

Sintaksis yang didukung

```
CREATE [ OR REPLACE ] [ RECURSIVE ] VIEW name [ ( column_name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ... ] ) ]
    AS query
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

Deskripsi

`CREATE VIEW`mendefinisikan tampilan kueri. Pandangan itu tidak terwujud secara fisik. Sebagai gantinya, kueri dijalankan setiap kali tampilan direferensikan dalam kueri.

`CREATE or REPLACE VIEW`mirip, tetapi jika tampilan dengan nama yang sama sudah ada, itu diganti. Kueri baru harus menghasilkan kolom yang sama yang dihasilkan oleh kueri tampilan yang ada (yaitu, nama kolom yang sama dalam urutan yang sama dan dengan tipe data yang sama), tetapi

mungkin menambahkan kolom tambahan ke akhir daftar. Perhitungan yang menimbulkan kolom output mungkin berbeda.

Jika nama skema diberikan, sepertiCREATE VIEW myschema.myview ...) maka tampilan dibuat dalam skema yang ditentukan. Jika tidak, itu dibuat dalam skema saat ini.

Nama tampilan harus berbeda dari nama relasi lainnya (tabel, indeks, tampilan) dalam skema yang sama.

Parameter

CREATE VIEWmendukung berbagai parameter untuk mengontrol perilaku tampilan yang dapat diperbarui secara otomatis.

RECURSIVE

Membuat tampilan rekursif. Sintaks: CREATE RECURSIVE VIEW [schema .] view_name (column_names) AS SELECT ...; setara dengan. CREATE VIEW [schema .] view_name AS WITH RECURSIVE view_name (column_names) AS (SELECT ...) SELECT column_names FROM view_name;

Daftar nama kolom tampilan harus ditentukan untuk tampilan rekursif.

name

Nama tampilan yang akan dibuat, yang mungkin secara opsional memenuhi syarat skema. Daftar nama kolom harus ditentukan untuk tampilan rekursif.

column_name

Daftar opsional nama yang akan digunakan untuk kolom tampilan. Jika tidak diberikan, nama kolom disimpulkan dari kueri.

WITH (view_option_name [= view_option_value] [, ...])

Klausa ini menentukan parameter opsional untuk tampilan; parameter berikut didukung.

- check_option (enum)— Parameter ini mungkin salah satu local ataucascaded, dan setara dengan menentukanWITH [CASCADED | LOCAL] CHECK OPTION.
- security_barrier (boolean)—Ini harus digunakan jika tampilan dimaksudkan untuk memberikan keamanan tingkat baris. Aurora DSQL saat ini tidak mendukung keamanan tingkat baris, tetapi opsi ini masih akan memaksa kondisi tampilan (dan WHERE kondisi apa pun yang menggunakan operator yang ditandai sebagaiLEAKPROOF) untuk dievaluasi terlebih dahulu.

- `security_invoker (boolean)`—Opsi ini menyebabkan hubungan dasar yang mendasarinya diperiksa terhadap hak istimewa pengguna tampilan daripada pemilik tampilan. Lihat catatan di bawah ini untuk detail selengkapnya.

Semua opsi di atas dapat diubah pada tampilan yang ada menggunakan`ALTER VIEW`.

`query`

A `SELECT` atau `VALUES` perintah yang akan menyediakan kolom dan baris tampilan.

- `WITH [ CASCADED | LOCAL ] CHECK OPTION`— Opsi ini mengontrol perilaku tampilan yang dapat diperbarui secara otomatis. Ketika opsi ini ditentukan, `INSERT` dan `UPDATE` perintah pada tampilan akan diperiksa untuk memastikan bahwa baris baru memenuhi kondisi penentuan tampilan (yaitu, baris baru diperiksa untuk memastikan bahwa mereka terlihat melalui tampilan). Jika tidak, pembaruan akan ditolak. Jika tidak `CHECK OPTION` ditentukan, `INSERT` dan `UPDATE` perintah pada tampilan diizinkan untuk membuat baris yang tidak terlihat melalui tampilan. Opsi pemeriksaan berikut didukung.
- `LOCAL`—Baris baru hanya diperiksa terhadap kondisi yang ditentukan secara langsung dalam tampilan itu sendiri. Setiap kondisi yang ditentukan pada tampilan dasar yang mendasarinya tidak dicentang (kecuali mereka juga menentukan`CHECK OPTION`).
- `CASCADED`—Baris baru diperiksa terhadap kondisi tampilan dan semua tampilan dasar yang mendasarinya. Jika `CHECK OPTION` ditentukan, dan tidak `LOCAL` juga `CASCADED` ditentukan, maka `CASCADED` diasumsikan.

> **Note**
>
> Ini tidak `CHECK OPTION` dapat digunakan dengan `RECURSIVE` tampilan. Hanya `CHECK OPTION` didukung pada tampilan yang dapat diperbarui secara otomatis.

Catatan

Gunakan `DROP VIEW` pernyataan untuk menghapus tampilan.

Nama dan tipe data kolom tampilan harus dipertimbangkan dengan cermat. Misalnya, CREATE VIEW vista AS SELECT 'Hello World'; tidak disarankan karena nama kolom default ke. `?column?;` Juga, tipe data kolom default `ketext`, yang mungkin bukan yang Anda inginkan.

Pendekatan yang lebih baik adalah secara eksplisit menentukan nama kolom dan tipe data, seperti:. `CREATE VIEW vista AS SELECT text 'Hello World' AS hello;`

Secara default, akses ke relasi dasar yang dirujuk dalam tampilan ditentukan oleh izin pemilik tampilan. Dalam beberapa kasus, ini dapat digunakan untuk menyediakan akses yang aman tetapi terbatas ke tabel yang mendasarinya. Namun, tidak semua pandangan aman terhadap gangguan.

- Jika tampilan memiliki `security_invoker` properti yang disetel ke true, akses ke relasi dasar yang mendasarinya ditentukan oleh izin pengguna yang mengeksekusi kueri, bukan pemilik tampilan. Dengan demikian, pengguna tampilan pemanggil keamanan harus memiliki izin yang relevan pada tampilan dan hubungan dasar yang mendasarinya.
- Jika salah satu hubungan dasar yang mendasarinya adalah tampilan pemanggil keamanan, itu akan diperlakukan seolah-olah telah diakses langsung dari kueri asli. Dengan demikian, tampilan pemanggil keamanan akan selalu memeriksa hubungan dasar yang mendasarinya menggunakan izin pengguna saat ini, bahkan jika itu diakses dari tampilan tanpa properti. `security_invoker`
- Fungsi yang dipanggil dalam tampilan diperlakukan sama seperti jika mereka telah dipanggil langsung dari kueri menggunakan tampilan. Oleh karena itu, pengguna tampilan harus memiliki izin untuk memanggil semua fungsi yang digunakan oleh tampilan. Fungsi dalam tampilan dijalankan dengan hak istimewa pengguna yang mengeksekusi kueri atau pemilik fungsi, tergantung pada apakah fungsi didefinisikan sebagai `SECURITY INVOKER` atau. `SECURITY DEFINER` Misalnya, menelepon `CURRENT_USER` langsung dalam tampilan akan selalu mengembalikan pengguna yang memanggil, bukan pemilik tampilan. Ini tidak terpengaruh oleh `security_invoker` pengaturan tampilan, sehingga tampilan dengan `security_invoker` disetel ke false tidak setara dengan `SECURITY DEFINER` fungsi.
- Pengguna yang membuat atau mengganti tampilan harus memiliki `USAGE` hak istimewa pada skema apa pun yang dirujuk dalam kueri tampilan, untuk mencari objek yang direferensikan dalam skema tersebut. Perhatikan, bagaimanapun, bahwa pencarian ini hanya terjadi ketika tampilan dibuat atau diganti. Oleh karena itu, pengguna tampilan hanya memerlukan `USAGE` hak istimewa pada skema yang berisi tampilan, bukan pada skema yang dirujuk dalam kueri tampilan, bahkan untuk tampilan pemanggil keamanan.
- Ketika `CREATE OR REPLACE VIEW` digunakan pada tampilan yang ada, hanya `SELECT` aturan penentu tampilan, ditambah `WITH ( ... )` parameter apa pun dan `CHECK OPTION` yang diubah. Properti tampilan lainnya, termasuk kepemilikan, izin, dan aturan non-Pilih, tetap tidak berubah. Anda harus memiliki tampilan untuk menggantinya (ini termasuk menjadi anggota peran pemilik).

Tampilan yang dapat diperbarui

Tampilan sederhana dapat diperbarui secara otomatis: sistem akan memungkinkan`INSERT`,`UPDATE`, dan `DELETE` pernyataan yang akan digunakan pada tampilan dengan cara yang sama seperti pada tabel biasa. Tampilan dapat diperbarui secara otomatis jika memenuhi semua kondisi berikut:

- Tampilan harus memiliki tepat satu entri dalam `FROM` daftarnya, yang harus berupa tabel atau tampilan lain yang dapat diperbarui.
- Definisi tampilan tidak boleh berisi`WITH`,,`DISTINCT`,`GROUP BY`, `HAVING`,`LIMIT`, atau `OFFSET` klausa di tingkat atas.
- Definisi tampilan tidak boleh berisi operasi set (`UNION`,`INTERSECT`, atau`EXCEPT`) di tingkat atas.
- Daftar pilih tampilan tidak boleh berisi agregat, fungsi jendela, atau fungsi set-return.

Tampilan yang dapat diperbarui secara otomatis mungkin berisi campuran kolom yang dapat diperbarui dan yang tidak dapat diperbarui. Kolom dapat diperbarui jika itu adalah referensi sederhana ke kolom yang dapat diperbarui dari hubungan dasar yang mendasarinya. Jika tidak, kolom tersebut hanya-baca, dan kesalahan terjadi jika `UPDATE` pernyataan `INSERT` atau mencoba untuk menetapkan nilai untuk itu.

Untuk tampilan yang dapat diperbarui secara otomatis, sistem mengonversi pernyataan apa pun `INSERT`,`UPDATE`, atau pada tampilan menjadi `DELETE` pernyataan yang sesuai pada relasi dasar yang mendasarinya. `INSERT`pernyataan dengan `ON CONFLICT UPDATE` klausa didukung sepenuhnya.

Jika tampilan yang dapat diperbarui secara otomatis berisi `WHERE` kondisi, kondisi membatasi baris relasi dasar mana yang tersedia untuk dimodifikasi oleh `UPDATE` dan `DELETE` pernyataan pada tampilan. Namun, an `UPDATE` dapat mengubah baris sehingga tidak lagi memenuhi `WHERE` kondisi, membuatnya tidak terlihat melalui tampilan. Demikian pula, sebuah `INSERT` perintah berpotensi menyisipkan baris hubungan dasar yang tidak memenuhi `WHERE` kondisi, membuatnya tidak terlihat melalui tampilan. `ON CONFLICT UPDATE`juga dapat mempengaruhi baris yang ada yang tidak terlihat melalui tampilan.

Anda dapat menggunakan `CHECK OPTION` untuk mencegah `INSERT` dan `UPDATE` perintah membuat baris yang tidak terlihat melalui tampilan.

Jika tampilan yang dapat diperbarui secara otomatis ditandai dengan properti security_barrier, semua kondisi tampilan (dan `WHERE` kondisi apa pun yang menggunakan operator yang ditandai

sebagaiLEAKPROOF) selalu dievaluasi sebelum kondisi apa pun yang ditambahkan oleh pengguna tampilan. Perhatikan bahwa karena ini, baris yang pada akhirnya tidak dikembalikan (karena tidak melewati WHERE kondisi pengguna) mungkin masih terkunci. Anda dapat menggunakan EXPLAIN untuk melihat kondisi mana yang diterapkan pada tingkat relasi (dan karenanya tidak mengunci baris) dan mana yang tidak.

Tampilan yang lebih kompleks yang tidak memenuhi semua kondisi ini adalah hanya-baca secara default: sistem tidak mengizinkan penyisipan, pembaruan, atau penghapusan pada tampilan.

> **Note**
>
> Pengguna yang melakukan penyisipan, pembaruan, atau penghapusan pada tampilan harus memiliki hak sisipan, pembaruan, atau hapus yang sesuai pada tampilan. Secara default, pemilik tampilan harus memiliki hak istimewa yang relevan pada relasi dasar yang mendasarinya, sementara pengguna yang melakukan pembaruan tidak memerlukan izin apa pun pada relasi dasar yang mendasarinya. Namun, jika tampilan memiliki security_invoker disetel ke true, pengguna yang melakukan pembaruan, bukan pemilik tampilan, harus memiliki hak istimewa yang relevan pada relasi dasar yang mendasarinya.

Contoh

Untuk membuat tampilan yang terdiri dari semua film komedi.

```
CREATE VIEW comedies AS
    SELECT *
    FROM films
    WHERE kind = 'Comedy';
```

Ini akan membuat tampilan yang berisi kolom yang ada di film tabel pada saat pembuatan tampilan. Meskipun * digunakan untuk membuat tampilan, kolom yang ditambahkan kemudian ke tabel tidak akan menjadi bagian dari tampilan.

Buat tampilan denganLOCAL CHECK OPTION.

```
CREATE VIEW pg_comedies AS
    SELECT *
    FROM comedies
    WHERE classification = 'PG'
```

```
    WITH CASCADED CHECK OPTION;
```

Ini akan membuat tampilan yang memeriksa baris `kind` dan `classification` baris baru.

Buat tampilan dengan campuran kolom yang dapat diperbarui dan tidak dapat diperbarui.

```
CREATE VIEW comedies AS
    SELECT f.*,
           country_code_to_name(f.country_code) AS country,
           (SELECT avg(r.rating)
            FROM user_ratings r
            WHERE r.film_id = f.id) AS avg_rating
    FROM films f
    WHERE f.kind = 'Comedy';
```

Pandangan ini akan mendukung`INSERT`,`UPDATE`, dan`DELETE`. Semua kolom dari tabel film akan dapat diperbarui, sedangkan kolom yang dihitung `country` dan hanya `avg_rating` akan dibaca.

```
CREATE RECURSIVE VIEW public.nums_1_100 (n) AS
    VALUES (1)
UNION ALL
    SELECT n+1 FROM nums_1_100 WHERE n < 100;
```

> **Note**
>
> Meskipun nama tampilan rekursif memenuhi syarat skema dalam hal ini`CREATE`, referensi diri internalnya tidak memenuhi syarat skema. Ini karena nama Common Table Expression (CTE) yang dibuat secara implisit tidak dapat memenuhi syarat skema.

Kompatibilitas

`CREATE OR REPLACE VIEW`adalah ekstensi bahasa PostgreSQL. `WITH ( ... )`Klausul ini juga merupakan perpanjangan, seperti juga pandangan penghalang keamanan dan pandangan pemanggil keamanan. Aurora DSQL mendukung ekstensi bahasa ini.

ALTER VIEW

`ALTER VIEW`Pernyataan ini memungkinkan mengubah berbagai properti dari tampilan yang ada, dan Aurora DSQL mendukung semua sintaks PostgreSQL untuk perintah ini.

Sintaksis yang didukung

```
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name SET DEFAULT expression
ALTER VIEW [ IF EXISTS ] name ALTER [ COLUMN ] column_name DROP DEFAULT
ALTER VIEW [ IF EXISTS ] name OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER |
 SESSION_USER }
ALTER VIEW [ IF EXISTS ] name RENAME [ COLUMN ] column_name TO new_column_name
ALTER VIEW [ IF EXISTS ] name RENAME TO new_name
ALTER VIEW [ IF EXISTS ] name SET SCHEMA new_schema
ALTER VIEW [ IF EXISTS ] name SET ( view_option_name [= view_option_value] [, ... ] )
ALTER VIEW [ IF EXISTS ] name RESET ( view_option_name [, ... ] )
```

Deskripsi

`ALTER VIEW`mengubah berbagai properti tambahan tampilan. (Jika Anda ingin mengubah kueri penentu tampilan, gunakan`CREATE OR REPLACE VIEW`.) Anda harus memiliki pandangan untuk digunakan`ALTER VIEW`. Untuk mengubah skema tampilan, Anda juga harus memiliki `CREATE` hak istimewa pada skema baru. Untuk mengubah pemilik, Anda harus `SET ROLE` dapat memiliki peran baru, dan peran itu harus memiliki `CREATE` hak istimewa pada skema tampilan. Pembatasan ini memberlakukan bahwa mengubah pemilik tidak melakukan apa pun yang tidak dapat Anda lakukan dengan menjatuhkan dan membuat ulang tampilan.)

Parameter

Parameter `ALTER VIEW`

`name`

Nama (opsional schema-qualified) dari tampilan yang ada.

`column_name`

Nama baru untuk kolom yang ada.

`IF EXISTS`

Jangan melempar kesalahan jika tampilan tidak ada. Pemberitahuan dikeluarkan dalam kasus ini.

`SET/DROP DEFAULT`

Formulir ini mengatur atau menghapus nilai default untuk kolom. Nilai default untuk kolom tampilan diganti ke salah satu `INSERT` atau `UPDATE` perintah di mana target adalah tampilan. Default untuk tampilan akan diutamakan daripada nilai default apa pun dari relasi yang mendasarinya.

new_owner

Nama pengguna dari pemilik tampilan yang baru.

new_name

Nama baru untuk tampilan.

new_schema

Skema baru untuk tampilan.

`SET ( view_option_name [= view_option_value] [, ... ] )`, **`RESET ( view_option_name [, ... ] )`**

Menyetel atau mengatur ulang opsi tampilan. Berikut ini adalah opsi yang didukung.

- `check_option (enum)`- Mengubah opsi centang tampilan. Nilainya harus `local` atau`cascaded`.
- `security_barrier (boolean)`- Mengubah properti penghalang keamanan tampilan. Nilai harus berupa nilai Boolean, seperti `true` atau`false`.
- `security_invoker (boolean)`- Mengubah properti penghalang keamanan tampilan. Nilai harus berupa nilai Boolean, seperti `true` atau`false`.

Catatan

Untuk alasan PostgreSQL historis`ALTER TABLE`, dapat digunakan dengan tampilan juga; tetapi satu-satunya varian `ALTER TABLE` yang diizinkan dengan tampilan setara dengan yang ditunjukkan sebelumnya.

Contoh

Mengganti nama tampilan `foo` menjadi`bar`.

```
ALTER VIEW foo RENAME TO bar;
```

Melampirkan nilai kolom default ke tampilan yang dapat diperbarui.

```
CREATE TABLE base_table (id int, ts timestamptz);
CREATE VIEW a_view AS SELECT * FROM base_table;
ALTER VIEW a_view ALTER COLUMN ts SET DEFAULT now();
INSERT INTO base_table(id) VALUES(1);  -- ts will receive a NULL
INSERT INTO a_view(id) VALUES(2);  -- ts will receive the current time
```

Kompatibilitas

`ALTER VIEW`adalah ekstensi PostgreSQL dari standar SQL yang didukung Aurora DSQL.

DROP VIEW

`DROP VIEW`Pernyataan menghapus tampilan yang ada. Aurora DSQL mendukung sintaks PostgreSQL lengkap untuk perintah ini.

Sintaksis yang didukung

```
DROP VIEW [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

Deskripsi

`DROP VIEW`menjatuhkan tampilan yang ada. Untuk menjalankan perintah ini, Anda harus menjadi pemilik tampilan.

Parameter

`IF EXISTS`

Jangan melempar kesalahan jika tampilan tidak ada. Pemberitahuan dikeluarkan dalam kasus ini.

`name`

Nama (opsional schema-qualified) dari tampilan yang akan dihapus.

`CASCADE`

Secara otomatis menjatuhkan objek yang bergantung pada tampilan (seperti tampilan lain), dan pada gilirannya semua objek yang bergantung pada objek tersebut.

`RESTRICT`

Menolak untuk menjatuhkan tampilan jika ada objek yang bergantung padanya. Ini adalah opsi default.

Contoh

```
DROP VIEW kinds;
```

Kompatibilitas

Perintah ini sesuai dengan standar SQL, kecuali bahwa standar hanya memungkinkan satu tampilan untuk dijatuhkan per perintah, dan terlepas dari `IF EXISTS` opsi, yang merupakan ekstensi PostgreSQL yang didukung Aurora DSQL.

Fitur PostgreSQL yang tidak didukung di Aurora DSQL

Aurora DSQL kompatibel dengan PostgreSQL. Ini berarti bahwa Aurora DSQL mendukung fitur relasional inti seperti transaksi ACID, indeks sekunder, bergabung, menyisipkan, dan pembaruan. Untuk ikhtisar fitur SQL yang didukung, lihat Ekspresi SQL yang didukung.

Bagian berikut menyoroti fitur PostgreSQL mana yang saat ini tidak didukung di Aurora DSQL.

Objek yang tidak didukung

Objek yang tidak didukung oleh Aurora DSQL meliputi:

- Beberapa database pada satu cluster Aurora DSQL
- Tabel Sementara
- Pemicu
- Jenis (dukungan sebagian)
- Ruang Meja
- Fungsi yang ditulis dalam bahasa selain SQL
- Urutan
- Partisi

Kendala yang tidak didukung

- Kunci asing
- Kendala pengecualian

Perintah tidak didukung

- `ALTER SYSTEM`
- `TRUNCATE`

- `SAVEPOINT`
- `VACUUM`

> **Note**
>
> Aurora DSQL tidak memerlukan penyedot debu. Sistem memelihara statistik dan mengelola pengoptimalan penyimpanan secara otomatis tanpa perintah vakum manual.

Ekstensi yang tidak didukung

Aurora DSQL tidak mendukung ekstensi PostgreSQL. Tabel berikut menunjukkan ekstensi yang tidak didukung:

- `PL/pgSQL`
- `PostGIS`
- `PGVector`
- `PGAudit`
- `Postgres_FDW`
- `PGCron`
- `pg_stat_statements`

Ekspresi SQL yang tidak didukung

Tabel berikut menjelaskan klausa yang tidak didukung di Aurora DSQL.

Kategori	Klausul Utama	Klausul Tidak Didukung
`CREATE`	`INDEX ASYNC`	`ASC | DESC`
`CREATE`	`INDEX`[1]	
`TRUNCATE`		
`ALTER`	`SYSTEM`	Semua `ALTER SYSTEM` perintah diblokir.

Kategori	Klausul Utama	Klausul Tidak Didukung
CREATE	TABLE	COLLATE, AS SELECT, INHERITS, PARTITION
CREATE	FUNCTION	LANGUAGE *non-sql-lang* Dimana *non-sql-lang* ada bahasa lain selain SQL
CREATE	TEMPORARY	TABLES
CREATE	EXTENSION	
CREATE	SEQUENCE	
CREATE	MATERIALIZED	VIEW
CREATE	TABLESPACE	
CREATE	TRIGGER	
CREATE	TYPE	
CREATE	DATABASE	Anda tidak dapat membuat database tambahan.

[1] Lihat Indeks asinkron di Aurora DSQL untuk membuat indeks pada kolom tabel tertentu.

Pertimbangan Aurora DSQL untuk kompatibilitas PostgreSQL

Pertimbangkan batasan kompatibilitas berikut saat menggunakan Aurora DSQL. Untuk pertimbangan umum, lihatPertimbangan untuk bekerja dengan Amazon Aurora DSQL. Untuk kuota dan batasan, lihatKuota cluster dan batas database di Amazon Aurora DSQL.

- Aurora DSQL menggunakan database built-in tunggal bernama. `postgres` Anda tidak dapat membuat database tambahan atau mengganti nama atau menjatuhkan database. `postgres`
- `postgres`Basis data menggunakan pengkodean karakter UTF-8. Anda tidak dapat mengubah pengkodean.

- Basis data hanya C menggunakan pemeriksaan.
- Aurora DSQL digunakan UTC sebagai zona waktu sistem. Anda tidak dapat mengubah zona waktu menggunakan parameter atau pernyataan SQL seperti. `SET TIMEZONE`
- Tingkat isolasi transaksi ditetapkan di PostgreSQL`Repeatable Read`.
- Transaksi memiliki kendala sebagai berikut:
  - Transaksi tidak dapat mencampur operasi DDL dan DML
  - Transaksi hanya dapat mencakup 1 pernyataan DDL
  - Transaksi dapat memodifikasi hingga 3.000 baris, terlepas dari jumlah indeks sekunder
  - Batas 3.000 baris berlaku untuk semua pernyataan DHTML (,,) `INSERT UPDATE DELETE`
- Waktu koneksi database habis setelah 1 jam.
- Aurora DSQL saat ini tidak membiarkan Anda menjalankan. `GRANT [permission] ON DATABASE` Jika Anda mencoba menjalankan pernyataan itu, Aurora DSQL mengembalikan pesan kesalahan. `ERROR: unsupported object type in GRANT`
- Aurora DSQL tidak mengizinkan peran pengguna non-admin untuk menjalankan perintah. `CREATE SCHEMA` Anda tidak dapat menjalankan `GRANT [permission] on DATABASE` perintah dan memberikan `CREATE` izin pada database. Jika peran pengguna non-admin mencoba membuat skema, Aurora DSQL kembali dengan pesan kesalahan. `ERROR: permission denied for database postgres`
- Pengguna non-admin tidak dapat membuat objek dalam skema publik. Hanya pengguna admin yang dapat membuat objek dalam skema publik. Peran pengguna admin memiliki izin untuk memberikan akses baca, tulis, dan modifikasi ke objek ini kepada pengguna non-admin, tetapi tidak dapat memberikan `CREATE` izin ke skema publik itu sendiri. Pengguna non-admin harus menggunakan skema yang dibuat pengguna yang berbeda untuk pembuatan objek.
- Aurora DSQL tidak mendukung perintah. `ALTER ROLE [] CONNECTION LIMIT` Hubungi AWS dukungan jika Anda memerlukan peningkatan batas koneksi.
- Aurora DSQL tidak mendukung asyncpg, driver database PostgreSQL asinkron untuk Python.

Kontrol konkurensi di Aurora DSQL

Concurrency memungkinkan beberapa sesi untuk mengakses dan memodifikasi data secara bersamaan tanpa mengorbankan integritas dan konsistensi data. Aurora DSQL menyediakan kompatibilitas PostgreSQL sambil menerapkan mekanisme kontrol konkurensi modern yang bebas

kunci. Ini mempertahankan kepatuhan ACID penuh melalui isolasi snapshot, memastikan konsistensi dan keandalan data.

Keuntungan utama Aurora DSQL adalah arsitekturnya yang bebas kunci, yang menghilangkan kemacetan kinerja database umum. Aurora DSQL mencegah transaksi lambat memblokir operasi lain dan menghilangkan risiko kebuntuan. Pendekatan ini membuat Aurora DSQL sangat berharga untuk aplikasi throughput tinggi di mana kinerja dan skalabilitas sangat penting.

Konflik transaksi

Aurora DSQL menggunakan kontrol konkurensi optimis (OCC), yang bekerja secara berbeda dari sistem berbasis kunci tradisional. Alih-alih menggunakan kunci, OCC mengevaluasi konflik pada waktu komit. Ketika beberapa transaksi bertentangan saat memperbarui baris yang sama, Aurora DSQL mengelola transaksi sebagai berikut:

- Transaksi dengan waktu komit paling awal diproses oleh Aurora DSQL.
- Transaksi yang bertentangan menerima kesalahan serialisasi PostgreSQL, yang menunjukkan perlunya dicoba lagi.

Rancang aplikasi Anda untuk menerapkan logika coba lagi untuk menangani konflik. Pola desain yang ideal adalah idempoten, memungkinkan percobaan ulang transaksi sebagai jalan pertama bila memungkinkan. Logika yang direkomendasikan mirip dengan logika batalkan dan coba lagi dalam batas waktu kunci PostgreSQL standar atau situasi kebuntuan. Namun, OCC mengharuskan aplikasi Anda untuk menggunakan logika ini lebih sering.

Pedoman untuk mengoptimalkan kinerja transaksi

Untuk mengoptimalkan kinerja, minimalkan pertengkaran tinggi pada tombol tunggal atau rentang kunci kecil. Untuk mencapai tujuan ini, rancang skema Anda untuk menyebarkan pembaruan pada rentang kunci klaster Anda dengan menggunakan pedoman berikut:

- Pilih kunci primer acak untuk tabel Anda.
- Hindari pola yang meningkatkan pertengkaran pada satu tombol. Pendekatan ini memastikan kinerja optimal bahkan ketika volume transaksi tumbuh.

DDL dan transaksi terdistribusi di Aurora DSQL

Bahasa definisi data (DDL) berperilaku berbeda di Aurora DSQL dari PostgreSQL. Aurora DSQL memiliki lapisan database multi-AZ yang didistribusikan dan tidak dibagikan yang dibangun di atas armada komputasi dan penyimpanan multi-tenant. Karena tidak ada node atau pemimpin basis data primer tunggal, katalog database didistribusikan. Dengan demikian, Aurora DSQL mengelola perubahan skema DDL sebagai transaksi terdistribusi.

Secara khusus, DDL berperilaku berbeda di Aurora DSQL sebagai berikut:

Kesalahan kontrol konkurensi

Aurora DSQL mengembalikan kesalahan pelanggaran kontrol konkurensi jika Anda menjalankan satu transaksi sementara transaksi lain memperbarui sumber daya. Misalnya, pertimbangkan urutan tindakan berikut:

1. Di sesi 1, pengguna menambahkan kolom ke tabel`mytable`.
2. Di sesi 2, pengguna mencoba memasukkan baris ke dalam`mytable`.

   Aurora DSQL mengembalikan kesalahan `SQL Error [40001]: ERROR: schema has been updated by another transaction, please retry: (OC001).`

DDL dan DML dalam transaksi yang sama

Transaksi di Aurora DSQL hanya dapat berisi satu pernyataan DDL dan tidak dapat memiliki pernyataan DDL dan DHTML. Pembatasan ini berarti Anda tidak dapat membuat tabel dan menyisipkan data ke dalam tabel yang sama dalam transaksi yang sama. Misalnya, Aurora DSQL mendukung transaksi berurutan berikut.

```
BEGIN;
  CREATE TABLE mytable (ID_col integer);
COMMIT;

BEGIN;
  INSERT into FOO VALUES (1);
COMMIT;
```

Aurora DSQL tidak mendukung transaksi berikut, yang mencakup keduanya dan pernyataan. `CREATE INSERT`

```
BEGIN;
  CREATE TABLE FOO (ID_col integer);
  INSERT into FOO VALUES (1);
COMMIT;
```

DDL asinkron

Dalam PostgreSQL standar, operasi DDL `CREATE INDEX` seperti mengunci tabel yang terpengaruh, membuatnya tidak tersedia untuk dibaca dan ditulis dari sesi lain. Di Aurora DSQL, pernyataan DDL ini berjalan secara asinkron menggunakan pengelola latar belakang. Akses ke tabel yang terpengaruh tidak diblokir. Dengan demikian, DDL pada tabel besar dapat berjalan tanpa downtime atau dampak kinerja. Untuk informasi selengkapnya tentang manajer pekerjaan asinkron di Aurora DSQL, lihat. Indeks asinkron di Aurora DSQL

Kunci utama di Aurora DSQL

Dalam Aurora DSQL, kunci utama adalah fitur yang secara fisik mengatur data tabel. Ini mirip dengan `CLUSTER` operasi di PostgreSQL atau indeks berkerumun di database lain. Saat Anda menentukan kunci primer, Aurora DSQL membuat indeks yang mencakup semua kolom dalam tabel. Struktur kunci utama di Aurora DSQL memastikan akses dan manajemen data yang efisien.

Struktur dan penyimpanan data

Saat Anda menentukan kunci primer, Aurora DSQL menyimpan data tabel dalam urutan kunci primer. Struktur terorganisir indeks ini memungkinkan pencarian kunci primer untuk mengambil semua nilai kolom secara langsung, alih-alih mengikuti penunjuk ke data seperti dalam indeks B-tree tradisional. Berbeda dengan `CLUSTER` operasi di PostgreSQL, yang mengatur ulang data hanya sekali, Aurora DSQL mempertahankan urutan ini secara otomatis dan terus menerus. Pendekatan ini meningkatkan kinerja kueri yang bergantung pada akses kunci primer.

Aurora DSQL juga menggunakan kunci utama untuk menghasilkan kunci unik di seluruh cluster untuk setiap baris dalam tabel dan indeks. Kunci unik ini juga mendukung manajemen data terdistribusi. Ini memungkinkan partisi otomatis data di beberapa node, mendukung penyimpanan yang dapat diskalakan dan konkurensi tinggi. Akibatnya, struktur kunci utama membantu Aurora DSQL menskalakan secara otomatis dan mengelola beban kerja bersamaan secara efisien.

Pedoman untuk memilih kunci utama

Saat memilih dan menggunakan kunci primer di Aurora DSQL, pertimbangkan pedoman berikut:

- Tentukan kunci utama saat Anda membuat tabel. Anda tidak dapat mengubah kunci ini atau menambahkan kunci utama baru nanti. Kunci primer menjadi bagian dari kunci seluruh cluster yang digunakan untuk partisi data dan penskalaan otomatis throughput tulis. Jika Anda tidak menentukan kunci utama, Aurora DSQL menetapkan ID tersembunyi sintetis.
- Untuk tabel dengan volume tulis tinggi, hindari penggunaan bilangan bulat yang meningkat secara monoton sebagai kunci utama. Hal ini dapat menyebabkan masalah kinerja dengan mengarahkan semua sisipan baru ke satu partisi. Sebagai gantinya, gunakan kunci primer dengan distribusi acak untuk memastikan distribusi penulisan yang merata di seluruh partisi penyimpanan.
- Untuk tabel yang jarang berubah atau hanya-baca, Anda dapat menggunakan tombol naik. Contoh kunci naik adalah stempel waktu atau nomor urut. Kunci padat memiliki banyak nilai yang berjarak dekat atau duplikat. Anda dapat menggunakan tombol naik meskipun padat karena kinerja tulis kurang kritis.
- Jika pemindaian tabel lengkap tidak memenuhi persyaratan kinerja Anda, pilih metode akses yang lebih efisien. Dalam kebanyakan kasus, ini berarti menggunakan kunci utama yang cocok dengan kunci gabungan dan pencarian Anda yang paling umum dalam kueri.
- Ukuran gabungan maksimum kolom dalam kunci primer adalah 1 kibibyte. Untuk informasi selengkapnya, lihat Batas basis data di Aurora DSQL dan tipe data yang didukung di Aurora DSQL.
- Anda dapat menyertakan hingga 8 kolom dalam kunci primer atau indeks sekunder. Untuk informasi selengkapnya, lihat Batas basis data di Aurora DSQL dan tipe data yang didukung di Aurora DSQL.

Indeks asinkron di Aurora DSQL

`CREATE INDEX ASYNC`Perintah membuat indeks pada satu atau lebih kolom dari tabel tertentu. Perintah ini adalah operasi DDL asinkron yang tidak memblokir transaksi lain. Ketika Anda menjalankan`CREATE INDEX ASYNC`, Aurora DSQL segera mengembalikan file. `job_id`

Anda dapat memantau status pekerjaan asinkron ini menggunakan tampilan sistem. `sys.jobs` Saat pekerjaan pembuatan indeks sedang berlangsung, Anda dapat menggunakan prosedur dan perintah ini:

`sys.wait_for_job(job_id)`*`'your_index_creation_job_id'`*

Memblokir sesi saat ini sampai pekerjaan yang ditentukan selesai atau gagal. Mengembalikan nilai Boolean yang menunjukkan keberhasilan atau kegagalan.

`DROP INDEX`

Membatalkan pekerjaan pembuatan indeks yang sedang berlangsung.

Ketika pembuatan indeks asinkron selesai, Aurora DSQL memperbarui katalog sistem untuk menandai indeks sebagai aktif.

> **Note**
>
> Perhatikan bahwa transaksi bersamaan yang mengakses objek di namespace yang sama selama pembaruan ini mungkin mengalami kesalahan konkurensi.

Ketika Aurora DSQL menyelesaikan tugas indeks asinkron, Aurora akan memperbarui katalog sistem untuk menunjukkan bahwa indeks aktif. Jika transaksi lain mereferensikan objek di namespace yang sama saat ini, Anda mungkin melihat kesalahan konkurensi.

Sintaksis

`CREATE INDEX ASYNC`menggunakan sintaks berikut.

```
CREATE [ UNIQUE ] INDEX ASYNC [ IF NOT EXISTS ] name ON table_name
     ( { column_name } [ NULLS { FIRST | LAST } ] )
     [ INCLUDE ( column_name [, ...] ) ]
     [ NULLS [ NOT ] DISTINCT ]
```

Parameter

`UNIQUE`

Menunjukkan ke Aurora DSQL untuk memeriksa nilai duplikat dalam tabel saat membuat indeks dan setiap kali Anda menambahkan data. Jika Anda menentukan parameter ini, menyisipkan dan memperbarui operasi yang akan menghasilkan entri duplikat menghasilkan kesalahan.

`IF NOT EXISTS`

Menunjukkan bahwa Aurora DSQL seharusnya tidak membuang pengecualian jika indeks dengan nama yang sama sudah ada. Dalam situasi ini, Aurora DSQL tidak membuat indeks baru. Perhatikan bahwa indeks yang Anda coba buat bisa memiliki struktur yang sangat berbeda dari indeks yang ada. Jika Anda menentukan parameter ini, nama indeks diperlukan.

name

Nama indeks. Anda tidak dapat menyertakan nama skema Anda dalam parameter ini.

Aurora DSQL membuat indeks dalam skema yang sama dengan tabel induknya. Nama indeks harus berbeda dari nama objek lain, seperti tabel atau indeks, dalam skema.

Jika Anda tidak menentukan nama, Aurora DSQL menghasilkan nama secara otomatis berdasarkan nama tabel induk dan kolom yang diindeks. Misalnya, jika Anda menjalankan`CREATE INDEX ASYNC on table1 (col1, col2)`, Aurora DSQL secara otomatis memberi nama indeks. `table1_col1_col2_idx`

`NULLS FIRST | LAST`

Urutan urutan kolom null dan non-null. `FIRST`menunjukkan bahwa Aurora DSQL harus mengurutkan kolom nol sebelum kolom non-null. `LAST`menunjukkan bahwa Aurora DSQL harus mengurutkan kolom null setelah kolom non-null.

`INCLUDE`

Daftar kolom untuk disertakan dalam indeks sebagai kolom non-kunci. Anda tidak dapat menggunakan kolom non-kunci dalam kualifikasi pencarian pemindaian indeks. Aurora DSQL mengabaikan kolom dalam hal keunikan untuk indeks.

`NULLS DISTINCT | NULLS NOT DISTINCT`

Menentukan apakah Aurora DSQL harus mempertimbangkan nilai null sebagai berbeda dalam indeks unik. Defaultnya adalah`DISTINCT`, yang berarti bahwa indeks unik dapat berisi beberapa nilai null dalam kolom. `NOT DISTINCT`menunjukkan bahwa indeks tidak dapat berisi beberapa nilai null dalam kolom.

Catatan penggunaan

Pertimbangkan panduan berikut ini:

- `CREATE INDEX ASYNC`Perintah tidak memperkenalkan kunci. Ini juga tidak mempengaruhi tabel dasar yang Aurora DSQL gunakan untuk membuat indeks.
- Selama operasi migrasi skema, `sys.wait_for_job(job_id)`*`'your_index_creation_job_id'`* prosedur ini berguna. Ini memastikan bahwa operasi DDL dan DML berikutnya menargetkan indeks yang baru dibuat.

- Setiap kali Aurora DSQL menjalankan tugas asinkron baru, ia memeriksa `sys.jobs` tampilan dan menghapus tugas yang memiliki status atau selama lebih dari 30 menit. `completed` `failed` Jadi, `sys.jobs` terutama menunjukkan tugas yang sedang berlangsung dan tidak berisi informasi tentang tugas-tugas lama.
- Jika Aurora DSQL gagal membangun indeks asinkron, indeks tetap ada. `INVALID` Untuk indeks unik, operasi DHTML tunduk pada kendala keunikan sampai Anda menjatuhkan indeks. Kami menyarankan Anda menjatuhkan indeks yang tidak valid dan membuatnya kembali.

Membuat indeks: contoh

Contoh berikut menunjukkan cara membuat skema, tabel, dan kemudian indeks.

1. Buat tabel bernama`test.departments`.

```
CREATE SCHEMA test;

CREATE TABLE test.departments (name varchar(255) primary key NOT null,
     manager varchar(255),
     size varchar(4));
```

2. Masukkan baris ke dalam tabel.

```
INSERT INTO test.departments VALUES ('Human Resources', 'John Doe', '10')
```

3. Buat indeks asinkron.

```
CREATE INDEX ASYNC test_index on test.departments(name, manager, size);
```

   `CREATE INDEX ASYNC`Perintah mengembalikan ID pekerjaan, seperti yang ditunjukkan di bawah ini.

```
job_id
-------------------------
jh2gbtx4mzhgfkbimtgwn5j45y
```

   `job_id`Ini menunjukkan bahwa Aurora DSQL telah mengirimkan pekerjaan baru untuk membuat indeks. Anda dapat menggunakan prosedur `sys.wait_for_job(job_id)` *`'your_index_creation_job_id'`* untuk memblokir pekerjaan lain pada sesi sampai pekerjaan selesai atau habis waktu.

Menanyakan status pembuatan indeks: contoh

Kueri tampilan `sys.jobs` sistem untuk memeriksa status pembuatan indeks Anda, seperti yang ditunjukkan pada contoh berikut.

```
SELECT * FROM sys.jobs
```

Aurora DSQL mengembalikan respon yang mirip dengan berikut ini.

```
           job_id           |   status   | details
----------------------------+------------+---------
 vs3kcl3rt5ddpk3a6xcq57cmcy | completed  |
 ihbyw2aoirfnrdfoc4ojnlamoq | processing |
```

Kolom status dapat menjadi salah satu nilai berikut.

submitted	**processing**	**failed**	**completed**
Tugas dikirimkan, tetapi Aurora DSQL belum mulai memprosesnya.	Aurora DSQL sedang memproses tugas.	Tugas gagal. Lihat kolom detail untuk informasi lebih lanjut. Jika Aurora DSQL gagal membangun indeks, Aurora DSQL tidak secara otomatis menghapus definisi indeks. Anda harus menghapus indeks secara manual dengan `DROP INDEX` perintah.	Aurora DSQL

Anda juga dapat menanyakan status indeks melalui tabel katalog `pg_index` dan`pg_class`. Secara khusus, atribut `indisvalid` dan `indisimmediate` dapat memberi tahu Anda status indeks Anda. Sementara Aurora DSQL membuat indeks Anda, ia memiliki status awal. `INVALID` `indisvalid`Bendera untuk indeks mengembalikan `FALSE` atau`f`, yang menunjukkan bahwa indeks tidak valid. Jika bendera kembali `TRUE` atau`t`, indeks siap.

```
SELECT relname AS index_name, indisvalid as is_valid, pg_get_indexdef(indexrelid) AS
 index_definition
from pg_index, pg_class
WHERE pg_class.oid = indexrelid AND indrelid = 'test.departments'::regclass;
```

```
    index_name    | is_valid |
 index_definition
------------------+----------
+-------------------------------------------------------------------------------------------
 department_pkey  |     t    | CREATE UNIQUE INDEX department_pkey ON test.departments
 USING btree_index (title) INCLUDE (name, manager, size)
 test_index1      |     t    | CREATE INDEX test_index1 ON test.departments USING
 btree_index (name, manager, size)
```

Kegagalan pembuatan indeks unik

Jika pekerjaan pembuatan indeks unik asinkron Anda menunjukkan status gagal dengan detailFound `duplicate key while validating index for UCVs`, ini menunjukkan bahwa indeks unik tidak dapat dibuat karena pelanggaran batasan keunikan.

Untuk mengatasi kegagalan pembuatan indeks unik

1. Hapus baris apa pun di tabel utama Anda yang memiliki entri duplikat untuk kunci yang ditentukan dalam indeks sekunder unik Anda.
2. Jatuhkan indeks yang gagal.
3. Keluarkan perintah buat indeks baru.

Mendeteksi pelanggaran keunikan di tabel utama

Kueri SQL berikut membantu Anda mengidentifikasi nilai duplikat dalam kolom tertentu dari tabel Anda. Ini sangat berguna ketika Anda perlu menerapkan keunikan pada kolom yang saat ini tidak ditetapkan sebagai kunci utama atau tidak memiliki kendala unik, seperti alamat email dalam tabel pengguna.

Contoh di bawah ini menunjukkan cara membuat tabel pengguna sampel, mengisinya dengan data uji yang berisi duplikat yang diketahui, dan kemudian menjalankan kueri deteksi.

Tentukan skema tabel

```
-- Drop the table if it exists
DROP TABLE IF EXISTS users;

-- Create the users table with a simple integer primary key
CREATE TABLE users (
    user_id INTEGER PRIMARY KEY,
    email VARCHAR(255),
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Masukkan data sampel yang mencakup kumpulan alamat email duplikat

```
-- Insert sample data with explicit IDs
INSERT INTO users (user_id, email, first_name, last_name) VALUES
    (1, 'john.doe@example.com', 'John', 'Doe'),
    (2, 'jane.smith@example.com', 'Jane', 'Smith'),
    (3, 'john.doe@example.com', 'Johnny', 'Doe'),
    (4, 'alice.wong@example.com', 'Alice', 'Wong'),
    (5, 'bob.jones@example.com', 'Bob', 'Jones'),
    (6, 'alice.wong@example.com', 'Alicia', 'Wong'),
    (7, 'bob.jones@example.com', 'Robert', 'Jones');
```

Jalankan kueri deteksi duplikat

```
-- Query to find duplicates
WITH duplicates AS (
    SELECT email, COUNT(*) as duplicate_count
    FROM users
    GROUP BY email
    HAVING COUNT(*) > 1
)
SELECT u.*, d.duplicate_count
FROM users u
INNER JOIN duplicates d ON u.email = d.email
ORDER BY u.email, u.user_id;
```

Lihat semua catatan dengan alamat email duplikat

```
 user_id |         email          | first_name | last_name |         created_at
 | duplicate_count
```

```
---------+-------------------------+------------+-----------
+---------------------------+-----------------
       4 | akua.mansa@example.com | Akua       | Mansa     | 2025-05-21 20:55:53.714432
 |               2
       6 | akua.mansa@example.com | Akua       | Mansa     | 2025-05-21 20:55:53.714432
 |               2
       1 | john.doe@example.com   | John       | Doe       | 2025-05-21 20:55:53.714432
 |               2
       3 | john.doe@example.com   | Johnny     | Doe       | 2025-05-21 20:55:53.714432
 |               2
(4 rows)
```

Jika kita mencoba pernyataan pembuatan indeks sekarang, itu akan gagal:

```
postgres=> CREATE UNIQUE INDEX ASYNC idx_users_email ON users(email);
      job_id
----------------------------
 ve32upmjz5dgdknpbleeca5tri
(1 row)

postgres=> select * from sys.jobs;
           job_id           |  status   |                       details
       |  job_type   | class_id | object_id |      object_name       |       start_time
       |      update_time
----------------------------+-----------
+-----------------------------------------------------+-------------+----------
+-----------+-------------------------+-------------------------+-----------------------
 qpn6aqlkijgmzilyidcpwrpova | completed |
          | DROP        |     1259 |     26384 |                       | 2025-05-20
 00:47:10+00 | 2025-05-20 00:47:32+00
 ve32upmjz5dgdknpbleeca5tri | failed    | Found duplicate key while validating index
 for UCVs | INDEX_BUILD |     1259 |     26396 | public.idx_users_email | 2025-05-20
 00:49:49+00 | 2025-05-20 00:49:56+00
(2 rows)
```

Tabel dan perintah sistem di Aurora DSQL

Lihat bagian berikut untuk mempelajari tentang tabel dan katalog sistem yang didukung di Aurora DSQL.

Tabel sistem

Aurora DSQL kompatibel dengan PostgreSQL, begitu banyak tabel katalog sistem dan tampilan dari PostgreSQL juga ada di Aurora DSQL.

Tabel dan tampilan katalog PostgreSQL penting

Tabel berikut menjelaskan tabel dan tampilan paling umum yang mungkin Anda gunakan di Aurora DSQL.

Nama	Penjelasan
`pg_namespace`	Informasi tentang semua skema
`pg_tables`	Informasi tentang semua tabel
`pg_attribute`	Informasi tentang semua atribut
`pg_views`	Informasi tentang (pra-) tampilan yang ditentukan
`pg_class`	Menjelaskan semua tabel, kolom, indeks, dan objek serupa
`pg_stats`	Pandangan tentang statistik perencana
`pg_user`	Informasi tentang pengguna
`pg_roles`	Informasi tentang pengguna dan grup
`pg_indexes`	Daftar semua indeks
`pg_constraint`	Daftar kendala pada tabel

Tabel katalog yang didukung dan tidak didukung

Tabel berikut menunjukkan tabel mana yang didukung dan tidak didukung di Aurora DSQL.

Nama	Berlaku untuk Aurora DSQL
`pg_aggregate`	Tidak

Nama	Berlaku untuk Aurora DSQL
pg_am	Ya
pg_amop	Tidak
pg_amproc	Tidak
pg_attrdef	Ya
pg_attribute	Ya
pg_authid	Tidak (gunakanpg_roles)
pg_auth_members	Ya
pg_cast	Ya
pg_class	Ya
pg_collation	Ya
pg_constraint	Ya
pg_conversion	Tidak
pg_database	Tidak
pg_db_role_setting	Ya
pg_default_acl	Ya
pg_depend	Ya
pg_description	Ya
pg_enum	Tidak
pg_event_trigger	Tidak
pg_extension	Tidak

Nama	Berlaku untuk Aurora DSQL
pg_foreign_data_wrapper	Tidak
pg_foreign_server	Tidak
pg_foreign_table	Tidak
pg_index	Ya
pg_inherits	Ya
pg_init_privs	Tidak
pg_language	Tidak
pg_largeobject	Tidak
pg_largeobject_metadata	Ya
pg_namespace	Ya
pg_opclass	Tidak
pg_operator	Ya
pg_opfamily	Tidak
pg_parameter_acl	Ya
pg_partitioned_table	Tidak
pg_policy	Tidak
pg_proc	Tidak
pg_publication	Tidak
pg_publication_namespace	Tidak
pg_publication_rel	Tidak

Nama	Berlaku untuk Aurora DSQL
pg_range	Ya
pg_replication_origin	Tidak
pg_rewrite	Tidak
pg_seclabel	Tidak
pg_sequence	Tidak
pg_shdepend	Ya
pg_shdescription	Ya
pg_shseclabel	Tidak
pg_statistic	Ya
pg_statistic_ext	Tidak
pg_statistic_ext_data	Tidak
pg_subscription	Tidak
pg_subscription_rel	Tidak
pg_tablespace	Tidak
pg_transform	Tidak
pg_trigger	Tidak
pg_ts_config	Ya
pg_ts_config_map	Ya
pg_ts_dict	Ya
pg_ts_parser	Ya

Nama	Berlaku untuk Aurora DSQL
pg_ts_template	Ya
pg_type	Ya
pg_user_mapping	Tidak

Tampilan sistem yang didukung dan tidak didukung

Tabel berikut menunjukkan tampilan mana yang didukung dan tidak didukung di Aurora DSQL.

Nama	Berlaku untuk Aurora DSQL
pg_available_extensions	Tidak
pg_available_extension_versions	Tidak
pg_backend_memory_contexts	Ya
pg_config	Tidak
pg_cursors	Tidak
pg_file_settings	Tidak
pg_group	Ya
pg_hba_file_rules	Tidak
pg_ident_file_mappings	Tidak
pg_indexes	Ya
pg_locks	Tidak
pg_matviews	Tidak
pg_policies	Tidak

Nama	Berlaku untuk Aurora DSQL
pg_prepared_statements	Tidak
pg_prepared_xacts	Tidak
pg_publication_tables	Tidak
pg_replication_origin_status	Tidak
pg_replication_slots	Tidak
pg_roles	Ya
pg_rules	Tidak
pg_seclabels	Tidak
pg_sequences	Tidak
pg_settings	Ya
pg_shadow	Ya
pg_shmem_allocations	Ya
pg_stats	Ya
pg_stats_ext	Tidak
pg_stats_ext_exprs	Tidak
pg_tables	Ya
pg_timezone_abbrevs	Ya
pg_timezone_names	Ya
pg_user	Ya
pg_user_mappings	Tidak

Nama	Berlaku untuk Aurora DSQL
pg_views	Ya
pg_stat_activity	Tidak
pg_stat_replication	Tidak
pg_stat_replication_slots	Tidak
pg_stat_wal_receiver	Tidak
pg_stat_recovery_prefetch	Tidak
pg_stat_subscription	Tidak
pg_stat_subscription_stats	Tidak
pg_stat_ssl	Ya
pg_stat_gssapi	Tidak
pg_stat_archiver	Tidak
pg_stat_io	Tidak
pg_stat_bgwriter	Tidak
pg_stat_wal	Tidak
pg_stat_database	Tidak
pg_stat_database_conflicts	Tidak
pg_stat_all_tables	Tidak
pg_stat_all_indexes	Tidak
pg_statio_all_tables	Tidak
pg_statio_all_indexes	Tidak

Nama	Berlaku untuk Aurora DSQL
pg_statio_all_sequences	Tidak
pg_stat_slru	Tidak
pg_statio_user_tables	Tidak
pg_statio_user_sequences	Tidak
pg_stat_user_functions	Tidak
pg_stat_user_indexes	Tidak
pg_stat_progress_analyze	Tidak
pg_stat_progress_basebackup	Tidak
pg_stat_progress_cluster	Tidak
pg_stat_progress_create_index	Tidak
pg_stat_progress_vacuum	Tidak
pg_stat_sys_indexes	Tidak
pg_stat_sys_tables	Tidak
pg_stat_xact_all_tables	Tidak
pg_stat_xact_sys_tables	Tidak
pg_stat_xact_user_functions	Tidak
pg_stat_xact_user_tables	Tidak
pg_statio_sys_indexes	Tidak
pg_statio_sys_sequences	Tidak
pg_statio_sys_tables	Tidak

Nama	Berlaku untuk Aurora DSQL
pg_statio_user_indexes	Tidak

Tampilan sys.jobs dan sys.iam_pg_role_mappings

Aurora DSQL mendukung tampilan sistem berikut:

sys.jobs

`sys.jobs`memberikan informasi status tentang pekerjaan asinkron. Misalnya, setelah Anda membuat indeks asinkron, Aurora DSQL mengembalikan file. `job_uuid` Anda dapat menggunakan ini `job_uuid` `sys.jobs` untuk mencari status pekerjaan.

```
SELECT * FROM sys.jobs WHERE job_id = 'example_job_uuid';

        job_id        |   status   | details
----------------------+------------+---------
 example_job_uuid | processing |
(1 row)
```

sys.iam_pg_role_mappings

Tampilan `sys.iam_pg_role_mappings` memberikan informasi tentang izin yang diberikan kepada pengguna IAM. Misalnya, jika `DQSLDBConnect` peran IAM yang memberikan Aurora DSQL akses ke non-admin dan pengguna `testuser` bernama diberikan `DQSLDBConnect` peran dan izin yang sesuai, Anda dapat menanyakan tampilan untuk melihat pengguna mana yang diberikan izin `sys.iam_pg_role_mappings` mana.

```
SELECT * FROM sys.iam_pg_role_mappings;
```

Tabel pg_class

`pg_class`Tabel menyimpan metadata tentang objek database. Untuk mendapatkan perkiraan hitungan berapa banyak baris dalam tabel, jalankan perintah berikut.

```
SELECT reltuples FROM pg_class WHERE relname = 'table_name';
```

Perintah ini menghasilkan output serupa dengan berikut:

```
  reltuples
--------------
 9.993836e+08
```

ANALYZEPerintah

`ANALYZE` Perintah mengumpulkan statistik tentang isi tabel dalam database dan menyimpan hasilnya dalam tampilan `pg_stats` sistem. Selanjutnya, perencana kueri menggunakan statistik ini untuk membantu menentukan rencana eksekusi yang paling efisien untuk kueri.

Di Aurora DSQL, Anda tidak dapat menjalankan `ANALYZE` perintah dalam transaksi eksplisit. `ANALYZE`tidak tunduk pada batas batas waktu transaksi database.

Untuk mengurangi kebutuhan intervensi manual dan menjaga statistik tetap up to date, Aurora DSQL secara otomatis berjalan `ANALYZE` sebagai proses latar belakang. Pekerjaan latar belakang ini dipicu secara otomatis berdasarkan tingkat perubahan yang diamati dalam tabel. Ini terkait dengan jumlah baris (tupel) yang telah dimasukkan, diperbarui, atau dihapus sejak analisis terakhir.

`ANALYZE`berjalan secara asinkron di latar belakang dan aktivitasnya dapat dipantau dalam tampilan sistem sys.jobs dengan kueri berikut:

```
SELECT * FROM sys.jobs WHERE job_type = 'ANALYZE';
```

Pertimbangan utama

> **Note**
>
> `ANALYZE`pekerjaan ditagih seperti pekerjaan asinkron lainnya di Aurora DSQL. Saat Anda memodifikasi tabel, ini secara tidak langsung dapat memicu pekerjaan pengumpulan statistik latar belakang otomatis, yang dapat mengakibatkan biaya pengukuran karena aktivitas tingkat sistem terkait.

`ANALYZE`Pekerjaan latar belakang, dipicu secara otomatis, mengumpulkan jenis statistik yang sama dengan manual `ANALYZE` dan menerapkannya secara default ke tabel pengguna. Tabel sistem dan katalog dikecualikan dari proses otomatis ini.

Mengelola klaster Aurora DSQL

Aurora DSQL menyediakan beberapa opsi konfigurasi untuk membantu Anda membangun infrastruktur database yang tepat untuk kebutuhan Anda. Untuk menyiapkan infrastruktur cluster Aurora DSQL Anda, tinjau bagian berikut.

Topik

Fitur dan fungsionalitas yang dibahas dalam panduan ini memastikan bahwa lingkungan Aurora DSQL Anda lebih tangguh, responsif, dan mampu mendukung aplikasi Anda saat mereka tumbuh dan berkembang.

Mengkonfigurasi kluster wilayah tunggal

Membuat klaster

Buat cluster menggunakan create-cluster perintah.

> **Note**
>
> Pembuatan cluster adalah operasi asinkron. Panggil GetCluster API hingga status berubah menjadi`ACTIVE`. Anda dapat terhubung ke cluster Anda setelah menjadi aktif.

Example Perintah

```
aws dsql create-cluster --region us-east-1
```

> **Note**
>
> Untuk menonaktifkan perlindungan penghapusan selama pembuatan, sertakan bendera. `--no-deletion-protection-enabled`

Example Respons

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "CREATING",
    "creationTime": "2024-05-25T16:56:49.784000-07:00",
    "deletionProtectionEnabled": true,
    "tag": {},
    "encryptionDetails": {
        "encryptionType": "AWS_OWNED_KMS_KEY",
        "encryptionStatus": "ENABLED"
    }
}
```

Menggambarkan sebuah cluster

Dapatkan informasi tentang cluster menggunakan get-cluster perintah.

Example Perintah

```
aws dsql get-cluster \
  --region us-east-1 \
  --identifier your_cluster_id
```

Example Respons

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "ACTIVE",
    "creationTime": "2024-11-27T00:32:14.434000-08:00",
    "deletionProtectionEnabled": false,
    "encryptionDetails": {
        "encryptionType": "CUSTOMER_MANAGED_KMS_KEY",
        "kmsKeyArn": "arn:aws:kms:us-east-1:111122223333:key/123a456b-c789-01de-2f34-
g5hi6j7k8lm9",
        "encryptionStatus": "ENABLED"
    }
}
```

Memperbarui klaster

Perbarui cluster yang ada menggunakan update-cluster perintah.

> **Note**
>
> Pembaruan adalah operasi asinkron. Panggil GetCluster API hingga status berubah `ACTIVE` untuk melihat perubahan Anda.

Example Perintah

```
aws dsql update-cluster \
  --region us-east-1 \
  --no-deletion-protection-enabled \
  --identifier your_cluster_id
```

Example Respons

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "UPDATING",
    "creationTime": "2024-05-24T09:15:32.708000-07:00"
}
```

Menghapus klaster

Hapus cluster yang ada menggunakan delete-cluster perintah.

> **Note**
>
> Anda hanya dapat menghapus cluster yang memiliki perlindungan penghapusan dinonaktifkan. Secara default, perlindungan penghapusan diaktifkan saat Anda membuat cluster baru.

Example Perintah

```
aws dsql delete-cluster \
```

```
  --region us-east-1 \
  --identifier your_cluster_id
```

Example Respons

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "DELETING",
    "creationTime": "2024-05-24T09:16:43.778000-07:00"
}
```

Daftar cluster

Buat daftar cluster Anda menggunakan list-clusters perintah.

Example Perintah

```
aws dsql list-clusters --region us-east-1
```

Example Respons

```
{
    "clusters": [
        {
            "identifier": "abc0def1baz2quux3quux4quuux",
            "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/
abc0def1baz2quux3quux4quuux"
        },
        {
            "identifier": "abc0def1baz2quux3quux5quuuux",
            "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/
abc0def1baz2quux3quux5quuuux"
        }
    ]
}
```

Mengkonfigurasi klaster Multi-wilayah

Bab ini menjelaskan cara mengkonfigurasi dan mengelola cluster di beberapa Wilayah AWS.

Menghubungkan ke klaster Multi-region

Cluster peered Multi-Region menyediakan dua titik akhir regional, satu di setiap cluster peered. Wilayah AWS Kedua titik akhir menyajikan database logis tunggal yang mendukung operasi baca dan tulis bersamaan dengan konsistensi data yang kuat. Selain cluster peered, cluster Multi-region juga memiliki Wilayah saksi yang menyimpan jendela terbatas log transaksi terenkripsi, yang digunakan untuk meningkatkan daya tahan dan ketersediaan Multi-region. Saksi Multi-Wilayah Wilayah tidak memiliki titik akhir.

Membuat cluster Multi-region

Untuk membuat klaster Multi-wilayah, pertama-tama Anda membuat klaster dengan Wilayah saksi. Kemudian Anda mengintip cluster ini dengan cluster kedua yang berbagi Wilayah saksi yang sama dengan cluster pertama Anda. Contoh berikut menunjukkan cara membuat cluster di AS Timur (Virginia N.) dan AS Timur (Ohio) dengan US West (Oregon) sebagai Wilayah saksi.

Langkah 1: Buat cluster satu di AS Timur (Virginia N.)

Untuk membuat cluster di AS Timur (Virginia N.) Wilayah AWS dengan properti Multi-region, gunakan perintah di bawah ini.

```
aws dsql create-cluster \
--region us-east-1 \
--multi-region-properties '{"witnessRegion":"us-west-2"}'
```

Example Respons:

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "UPDATING",
    "encryptionDetails": {
        "encryptionType": "AWS_OWNED_KMS_KEY",
        "encryptionStatus": "ENABLED"
   }
   "creationTime": "2024-05-24T09:15:32.708000-07:00"
}
```

> **Note**
>
> Ketika operasi API berhasil, cluster memasuki `PENDING_SETUP` status. Pembuatan cluster tetap ada `PENDING_SETUP` sampai Anda memperbarui cluster dengan ARN dari peer cluster.

Langkah 2: Buat cluster dua di US East (Ohio)

Untuk membuat cluster di US East (Ohio) Wilayah AWS dengan properti Multi-region, gunakan perintah di bawah ini.

```
aws dsql create-cluster \
--region us-east-2 \
--multi-region-properties '{"witnessRegion":"us-west-2"}'
```

Example Respons:

```
{
    "identifier": "foo0bar1baz2quux3quuxquux5",
    "arn": "arn:aws:dsql:us-east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5",
    "status": "PENDING_SETUP",
    "creationTime": "2025-05-06T06:51:16.145000-07:00",
    "deletionProtectionEnabled": true,
    "multiRegionProperties": {
        "witnessRegion": "us-west-2",
        "clusters": [
            "arn:aws:dsql:us-east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5"
        ]
    }
}
```

Ketika operasi API berhasil, klaster bertransisi ke `PENDING_SETUP` status. Pembuatan cluster tetap dalam `PENDING_SETUP` status sampai Anda memperbaruinya dengan ARN cluster lain untuk mengintip.

Langkah 3: Peer cluster di AS Timur (Virginia N.) dengan US East (Ohio)

Untuk mengintip cluster AS East (Virginia N.) Anda dengan cluster US East (Ohio) Anda, gunakan perintah. `update-cluster` Tentukan nama cluster US East (Virginia N.) Anda dan string JSON dengan ARN cluster US East (Ohio).

```
aws dsql update-cluster \
--region us-east-1 \
--identifier 'foo0bar1baz2quux3quuxquux4' \
--multi-region-properties '{"witnessRegion": "us-west-2","clusters": ["arn:aws:dsql:us-
east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5"]}'
```

Example Respons

```
{
    "identifier": "foo0bar1baz2quux3quuxquux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4",
    "status": "UPDATING",
    "creationTime": "2025-05-06T06:46:10.745000-07:00"
}
```

Langkah 4: Peer cluster di AS Timur (Ohio) dengan AS Timur (Virginia N.)

Untuk mengintip cluster US East (Ohio) Anda dengan cluster US East (Virginia N.) Anda, gunakan perintah. `update-cluster` Tentukan nama cluster US East (Ohio) Anda dan string JSON dengan ARN dari cluster US East (N. Virginia).

Example

```
aws dsql update-cluster \
--region us-east-2 \
--identifier 'foo0bar1baz2quux3quuxquux5' \
--multi-region-properties '{"witnessRegion": "us-west-2", "clusters":
 ["arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4"]}'
```

Example Respons

```
{
    "identifier": "foo0bar1baz2quux3quuxquux5",
    "arn": "arn:aws:dsql:us-east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5",
    "status": "UPDATING",
    "creationTime": "2025-05-06T06:51:16.145000-07:00"
}
```

> **Note**
>
> Setelah berhasil mengintip, kedua cluster beralih dari “PENDING_SETUP” ke “CREATING” dan akhirnya ke status “ACTIVE” saat siap digunakan.

Melihat properti cluster Multi-region

Saat mendeskripsikan sebuah klaster, Anda dapat melihat properti Multi-region untuk cluster secara berbeda. Wilayah AWS

Example

```
aws dsql get-cluster \
--region us-east-1 \
--identifier 'foo0bar1baz2quux3quuxquux4'
```

Example Respons

```
{
    "identifier": "foo0bar1baz2quux3quuxquux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4",
    "status": "PENDING_SETUP",
    "encryptionDetails": {
    "encryptionType": "AWS_OWNED_KMS_KEY",
    "encryptionStatus": "ENABLED"
},
    "creationTime": "2024-11-27T00:32:14.434000-08:00",
    "deletionProtectionEnabled": false,
    "multiRegionProperties": {
       "witnessRegion": "us-west-2",
       "clusters": [
          "arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4",
          "arn:aws:dsql:us-east-2:111122223333:cluster/foo0bar1baz2quux3quuxquux5"
       ]
    }
}
```

Peer cluster selama pembuatan

Anda dapat mengurangi jumlah langkah dengan memasukkan informasi peering selama pembuatan cluster. Setelah membuat cluster pertama Anda di US East (N. Virginia) (Langkah 1), Anda dapat

membuat cluster kedua Anda di US East (Ohio) sambil secara bersamaan memulai proses peering dengan memasukkan ARN dari cluster pertama.

Example

```
aws dsql create-cluster \
--region us-east-2 \
--multi-region-properties '{"witnessRegion":"us-west-2","clusters": ["arn:aws:dsql:us-east-1:111122223333:cluster/foo0bar1baz2quux3quuxquux4"]}'
```

Ini menggabungkan Langkah 2 dan 4, tetapi Anda masih harus menyelesaikan Langkah 3 (memperbarui cluster pertama dengan ARN dari cluster kedua) untuk membangun hubungan peering. Setelah semua langkah selesai, kedua cluster akan bertransisi melalui status yang sama seperti dalam proses standar: dari PENDING_SETUP ke CREATING, dan akhirnya ke ACTIVE saat siap digunakan.

Menghapus cluster Multi-region

Untuk menghapus cluster Multi-region, Anda harus menyelesaikan dua langkah.

1. Matikan perlindungan penghapusan untuk setiap cluster.
2. Hapus setiap cluster peered secara terpisah di masing-masing Wilayah AWS

Perbarui dan hapus cluster di AS Timur (Virginia N.)

1. Matikan perlindungan penghapusan menggunakan perintah. `update-cluster`

   ```
   aws dsql update-cluster \
     --region us-east-1 \
     --identifier 'foo0bar1baz2quux3quuxquux4' \
     --no-deletion-protection-enabled
   ```

2. Hapus cluster menggunakan `delete-cluster` perintah.

   ```
   aws dsql delete-cluster \
     --region us-east-1 \
     --identifier 'foo0bar1baz2quux3quuxquux4'
   ```

   Perintah mengembalikan respons berikut.

```
{
    "identifier": "foo0bar1baz2quux3quuxquux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/
foo0bar1baz2quux3quuxquux4",
    "status": "PENDING_DELETE",
    "creationTime": "2025-05-06T06:46:10.745000-07:00"
}
```

> **Note**
>
> Transisi cluster ke PENDING_DELETE status. Penghapusan tidak selesai sampai Anda menghapus cluster peered di US East (Ohio).

Perbarui dan hapus cluster di AS Timur (Ohio)

1. Matikan perlindungan penghapusan menggunakan perintah. `update-cluster`

```
aws dsql update-cluster \
--region us-east-2 \
--identifier 'foo0bar1baz2quux3quux4quuux' \
--no-deletion-protection-enabled
```

2. Hapus cluster menggunakan `delete-cluster` perintah.

```
aws dsql delete-cluster \
--region us-east-2 \
--identifier 'foo0bar1baz2quux3quuxquux5'
```

Perintah mengembalikan respon berikut:

```
{
    "identifier": "foo0bar1baz2quux3quuxquux5",
    "arn": "arn:aws:dsql:us-east-2:111122223333:cluster/
foo0bar1baz2quux3quuxquux5",
    "status": "PENDING_DELETE",
    "creationTime": "2025-05-06T06:46:10.745000-07:00"
}
```

> **Note**
>
> Transisi cluster ke `PENDING_DELETE` status. Setelah beberapa detik, sistem secara otomatis mentransisikan kedua cluster peered ke `DELETING` status setelah validasi.

Mengkonfigurasi cluster Multi-region menggunakan AWS CloudFormation

Anda dapat menggunakan AWS CloudFormation sumber daya yang sama `AWS::DSQL::Cluster` untuk menyebarkan dan mengelola klaster DSQL Aurora wilayah tunggal dan Multi-wilayah.

Lihat referensi jenis sumber daya Amazon Aurora DSQL untuk mengetahui lebih lanjut tentang cara membuat, memodifikasi, dan mengelola klaster menggunakan sumber daya. `AWS::DSQL::Cluster`

Membuat Konfigurasi Cluster Awal

Pertama, buat AWS CloudFormation template untuk menentukan cluster Multi-region Anda:

```
---
Resources:
  MRCluster:
    Type: AWS::DSQL::Cluster
    Properties:
      DeletionProtectionEnabled: true
      MultiRegionProperties:
        WitnessRegion: us-west-2
```

Buat tumpukan di kedua Wilayah menggunakan perintah AWS CLI berikut:

```
aws cloudformation create-stack --region us-east-2 \
    --stack-name MRCluster \
    --template-body file://mr-cluster.yaml
```

```
aws cloudformation create-stack --region us-east-1 \
    --stack-name MRCluster \
    --template-body file://mr-cluster.yaml
```

Menemukan Pengidentifikasi Cluster

Ambil sumber daya fisik IDs untuk cluster Anda:

```
aws cloudformation describe-stack-resources -region us-east-2 \
    --stack-name MRCluster \
    --query 'StackResources[].PhysicalResourceId'
[
  "auabudrks5jwh4mjt6o5xxhr4y"
]
```

```
aws cloudformation describe-stack-resources -region us-east-1 \
    --stack-name MRCluster \
    --query 'StackResources[].PhysicalResourceId'
[
  "imabudrfon4p2z3nv2jo4rlajm"
]
```

Memperbarui Konfigurasi Cluster

Perbarui AWS CloudFormation template Anda untuk menyertakan kedua cluster ARNs:

```
---
Resources:
  MRCluster:
    Type: AWS::DSQL::Cluster
    Properties:
      DeletionProtectionEnabled: true
      MultiRegionProperties:
        WitnessRegion: us-west-2
        Clusters:
        - arn:aws:dsql:us-east-2:123456789012:cluster/auabudrks5jwh4mjt6o5xxhr4y
        - arn:aws:dsql:us-east-1:123456789012:cluster/imabudrfon4p2z3nv2jo4rlajm
```

Terapkan konfigurasi yang diperbarui ke kedua Wilayah:

```
aws cloudformation update-stack --region us-east-2 \
    --stack-name MRCluster \
    --template-body file://mr-cluster.yaml
```

```
aws cloudformation update-stack --region us-east-1 \
    --stack-name MRCluster \
    --template-body file://mr-cluster.yaml
```

Pemrograman dengan Aurora DSQL

Aurora DSQL memberi Anda alat berikut untuk mengelola sumber daya Aurora DSQL Anda secara terprogram.

AWS Command Line Interface (AWS CLI)

Anda dapat membuat dan mengelola sumber daya Anda dengan menggunakan AWS CLI shell dalam baris perintah. AWS CLI Menyediakan akses langsung ke APIs for Layanan AWS, seperti Aurora DSQL. Untuk sintaks dan contoh perintah untuk Aurora DSQL, lihat dsql di Command Reference.AWS CLI

AWS kit pengembangan perangkat lunak () SDKs

AWS SDKs menyediakan banyak teknologi populer dan bahasa pemrograman. Mereka memudahkan Anda untuk menelepon Layanan AWS dari dalam aplikasi Anda dalam bahasa atau teknologi itu. Untuk informasi selengkapnya tentang ini SDKs, lihat Alat untuk mengembangkan dan mengelola aplikasi di AWS.

Aurora DSQL API

API ini adalah antarmuka pemrograman lain untuk Aurora DSQL. Saat menggunakan API ini, Anda harus memformat setiap permintaan HTTPS dengan benar dan menambahkan tanda tangan digital yang valid ke setiap permintaan. Untuk informasi selengkapnya, lihat Referensi API.

AWS CloudFormation

AWS::DSQL::ClusterIni adalah AWS CloudFormation sumber daya yang memungkinkan Anda untuk membuat dan mengelola cluster Aurora DSQL sebagai bagian dari infrastruktur Anda sebagai kode. AWS CloudFormation membantu Anda menentukan seluruh AWS lingkungan Anda dalam kode, membuatnya lebih mudah untuk menyediakan, memperbarui, dan mereplikasi infrastruktur Anda dengan cara yang konsisten dan andal.

Bila Anda menggunakan AWS::DSQL::Cluster sumber daya dalam AWS CloudFormation template Anda, Anda dapat secara deklaratif menyediakan klaster Aurora DSQL bersama sumber daya cloud Anda yang lain. Ini membantu memastikan bahwa infrastruktur data Anda menyebarkan dan mengelola bersama sisa tumpukan aplikasi Anda.

Amazon Aurora DSQL SDKs, driver, dan kode sampel

AWS kit pengembangan perangkat lunak (SDKs) tersedia untuk banyak bahasa pemrograman populer. Setiap SDK menyediakan API, contoh kode, dan dokumentasi yang memudahkan developer untuk membangun aplikasi dalam bahasa pilihan mereka.

Adaptor dan dialek

Tabel berikut menunjukkan adaptor ORM yang tersedia dan dialek database untuk Aurora DSQL.

Bahasa pemrograman	Kerangka Kerja	Tautan repositori
Java	Hibernasi	https://github.com/awslabs/aurora-dsql-hibernate/
Python	Django	https://github.com/awslabs/aurora-dsql-django/
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-sqlalchemy/

Sampel Kode

Manajemen cluster menggunakan AWS SDK

Tabel berikut menunjukkan contoh kode manajemen cluster untuk bahasa pemrograman yang berbeda menggunakan AWS SDKs.

Bahasa pemrograman	Contoh tautan repositori
C++	https://github.com/aws-samples/aurora-dsql-samples/tree/main/cpp/cluster_manajemen
C# (.NET)	https://github.com/aws-samples/aurora-dsql-samples/tree/main/dotnet/cluster_manajemen
Go	https://github.com/aws-samples/aurora-dsql-samples/tree/main/go/cluster_manajemen

Bahasa pemrograman	Contoh tautan repositori
Java	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/cluster_manajemen
JavaScript	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/cluster_manajemen
Python	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/cluster_manajemen
Ruby	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/cluster_manajemen
Rust	https://github.com/aws-samples/aurora-dsql-samples/tree/main/rust/cluster_manajemen

Driver dan Object-Relational Mapping () sampel ORMs

Tabel berikut menunjukkan driver database dan sampel kode kerangka ORM untuk bahasa pemrograman yang berbeda.

Bahasa pemrograman	Pengemudi atau kerangka kerja	Contoh tautan repositori
C++	libpq	https://github.com/aws-samples/aurora-dsql-samples/tree/main/cpp/libpq
C# (.NET)	Npgsql	https://github.com/aws-samples/aurora-dsql-samples/tree/main/dotnet/npgsql
Go	pgx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/go/pgx
Java	Hibernasi	https://github.com/awslabs/aurora-dsql-hibernate/tree/

Bahasa pemrograman	Pengemudi atau kerangka kerja	Contoh tautan repositori
		main/examples/pet-klinik-aplikasi
Java	PgJdBC	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/pgjdbc
JavaScript	simpul-postgres	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/node-postgres
JavaScript	Postgres.js	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/postgres-js
Python	Psycopg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/psycopg
Python	Psycopg2	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/psycopg2
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-sqlalchemy/tree/main/examples/pet-klinik-aplikasi
Ruby	Rel	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/rails

Bahasa pemrograman	Pengemudi atau kerangka kerja	Contoh tautan repositori
Ruby	pg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/ruby-pg
Rust	SQLx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/rust/sqlx
TypeScript	Sequelize	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/sequelize
TypeScript	TypeORM	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/type-orm

Aurora DSQL dengan AWS CLI

Lihat bagian berikut untuk mempelajari cara mengelola cluster Anda dengan. AWS CLI

CreateCluster

Untuk membuat cluster, gunakan `create-cluster` perintah.

> **Note**
>
> Pembuatan cluster terjadi secara asinkron. Panggil `GetCluster` API sampai statusnya`ACTIVE`. Anda dapat terhubung ke cluster setelah menjadi`ACTIVE`.

Perintah sampel

```
aws dsql create-cluster --region us-east-1
```

> **Note**
>
> Jika Anda ingin menonaktifkan perlindungan penghapusan saat pembuatan, sertakan bendera. `--no-deletion-protection-enabled`

Sampel respon

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "CREATING",
    "creationTime": "2025-05-22T14:03:26.631000-07:00",
    "encryptionDetails": {
        "encryptionType": "AWS_OWNED_KMS_KEY",
        "encryptionStatus": "ENABLED"
    },
    "deletionProtectionEnabled": true
}
```

GetCluster

Untuk mendeskripsikan sebuah cluster, gunakan `get-cluster` perintah.

Perintah sampel

```
aws dsql get-cluster \
  --region us-east-1 \
  --identifier <your_cluster_id>
```

Sampel respon

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "ACTIVE",
    "creationTime": "2025-05-22T14:03:26.631000-07:00",
    "deletionProtectionEnabled": true,
    "tags": {},
    "encryptionDetails": {
        "encryptionType": "AWS_OWNED_KMS_KEY",
```

```
        "encryptionStatus": "ENABLED"
    }
}
```

UpdateCluster

Untuk memperbarui cluster yang ada, gunakan `update-cluster` perintah.

> **Note**
>
> Pembaruan terjadi secara asinkron. Panggil `GetCluster` API sampai statusnya `ACTIVE` dan Anda akan mengamati perubahannya.

Perintah sampel

```
aws dsql update-cluster \
  --region us-east-1 \
  --no-deletion-protection-enabled \
  --identifier your_cluster_id
```

Sampel respon

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "UPDATING",
    "creationTime": "2024-05-24T09:15:32.708000-07:00"
}
```

DeleteCluster

Untuk menghapus cluster yang ada, gunakan `delete-cluster` perintah.

> **Note**
>
> Anda hanya dapat menghapus klaster yang memiliki perlindungan penghapusan dinonaktifkan. Perlindungan penghapusan diaktifkan secara default saat membuat cluster baru.

Perintah sampel

```
aws dsql delete-cluster \
  --region us-east-1 \
  --identifier your_cluster_id
```

Sampel respon

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "DELETING",
    "creationTime": "2024-05-24T09:16:43.778000-07:00"
}
```

ListClusters

Untuk mendapatkan cluster, gunakan `list-clusters` perintah.

Perintah sampel

```
aws dsql list-clusters --region us-east-1
```

Sampel respon

```
{
    "clusters": [
        {
            "identifier": "abc0def1baz2quux3quux4quuux",
            "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/
abc0def1baz2quux3quux4quuux"
        },
        {
            "identifier": "abc0def1baz2quux3quux4quuuux",
            "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/
abc0def1baz2quux3quux4quuuux"
        }
    ]
}
```

GetCluster pada klaster Multi-wilayah

Untuk mendapatkan informasi tentang cluster Multi-region, gunakan `get-cluster` perintah. Untuk klaster Multi-region, responsnya akan menyertakan klaster yang ditautkan. ARNs

Perintah sampel

```
aws dsql get-cluster \
  --region us-east-1 \
  --identifier your_cluster_id
```

Sampel respon

```
{
    "identifier": "abc0def1baz2quux3quuux4",
    "arn": "arn:aws:dsql:us-east-1:111122223333:cluster/abc0def1baz2quux3quuux4",
    "status": "ACTIVE",
    "creationTime": "2025-05-22T13:56:18.716000-07:00",
    "deletionProtectionEnabled": true,
    "multiRegionProperties": {
        "witnessRegion": "us-west-2",
        "clusters": [
            "arn:aws:dsql:us-east-1:842685632318:cluster/fuabuc7d3szkr37uqd5znkjynu"
        ]
    },
    "tags": {},
    "encryptionDetails": {
        "encryptionType": "AWS_OWNED_KMS_KEY",
        "encryptionStatus": "ENABLED"
    }
}
```

Buat, Baca, Perbarui, Hapus cluster Aurora DSQL

Contoh Create, Read, Update, Delete (CRUD) disediakan untuk penerapan Single-region dan Multi-region. Termasuk adalah `cluster_management` bagian khusus untuk setiap bahasa pemrograman yang menunjukkan tugas-tugas manajemen utama ini.

Penyebaran Single-Region ideal untuk aplikasi yang melayani pengguna di wilayah geografis tertentu, menawarkan manajemen yang disederhanakan dan latensi yang lebih rendah. Penerapan

Multi-Wilayah membantu Anda mencapai ketersediaan yang lebih tinggi dan kemampuan pemulihan bencana dengan mendistribusikan database Anda ke beberapa. Wilayah AWS

Pilih jenis penerapan yang sesuai dengan persyaratan aplikasi Anda untuk ketersediaan, kinerja, dan distribusi geografis.

Topik

Membuat klaster

Lihat informasi berikut untuk mempelajari cara membuat kluster Single-region dan Multi-region di Aurora DSQL.

Python

Untuk membuat cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
import boto3


def create_cluster(region):
    try:
        client = boto3.client("dsql", region_name=region)
        tags = {"Name": "Python single region cluster"}
        cluster = client.create_cluster(tags=tags, deletionProtectionEnabled=True)
        print(f"Initiated creation of cluster: {cluster["identifier"]}")

        print(f"Waiting for {cluster["arn"]} to become ACTIVE")
        client.get_waiter("cluster_active").wait(
            identifier=cluster["identifier"],
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )
```

```
        return cluster
    except:
        print("Unable to create cluster")
        raise


def main():
    region = "us-east-1"
    response = create_cluster(region)
    print(f"Created cluster: {response["arn"]}")


if __name__ == "__main__":
    main()
```

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
import boto3


def create_multi_region_clusters(region_1, region_2, witness_region):
    try:
        client_1 = boto3.client("dsql", region_name=region_1)
        client_2 = boto3.client("dsql", region_name=region_2)

        # We can only set the witness region for the first cluster
        cluster_1 = client_1.create_cluster(
            deletionProtectionEnabled=True,
            multiRegionProperties={"witnessRegion": witness_region},
            tags={"Name": "Python multi region cluster"}
        )
        print(f"Created {cluster_1["arn"]}")

        # For the second cluster we can set witness region and designate cluster_1
 as a peer
        cluster_2 = client_2.create_cluster(
            deletionProtectionEnabled=True,
            multiRegionProperties={"witnessRegion": witness_region, "clusters":
 [cluster_1["arn"]]},
            tags={"Name": "Python multi region cluster"}
        )

        print(f"Created {cluster_2["arn"]}")
        # Now that we know the cluster_2 arn we can set it as a peer of cluster_1
        client_1.update_cluster(
            identifier=cluster_1["identifier"],
            multiRegionProperties={"witnessRegion": witness_region, "clusters":
 [cluster_2["arn"]]}
        )
        print(f"Added {cluster_2["arn"]} as a peer of {cluster_1["arn"]}")

        # Now that multiRegionProperties is fully defined for both clusters
        # they'll begin the transition to ACTIVE
        print(f"Waiting for {cluster_1["arn"]} to become ACTIVE")
        client_1.get_waiter("cluster_active").wait(
            identifier=cluster_1["identifier"],
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

        print(f"Waiting for {cluster_2["arn"]} to become ACTIVE")
        client_2.get_waiter("cluster_active").wait(
            identifier=cluster_2["identifier"],
            WaiterConfig={
```

C++

Contoh berikut memungkinkan Anda membuat cluster dalam satu Wilayah AWS.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/CreateClusterRequest.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>
#include <thread>
#include <chrono>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Creates a single-region cluster in Amazon Aurora DSQL
 */
CreateClusterResult CreateCluster(const Aws::String& region) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Create the cluster
    CreateClusterRequest createClusterRequest;
    createClusterRequest.SetDeletionProtectionEnabled(true);
    createClusterRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    // Add tags
    Aws::Map<Aws::String, Aws::String> tags;
    tags["Name"] = "cpp single region cluster";
    createClusterRequest.SetTags(tags);

    auto createOutcome = client.CreateCluster(createClusterRequest);
    if (!createOutcome.IsSuccess()) {
        std::cerr << "Failed to create cluster in " << region << ": "
                  << createOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to create cluster in " + region);
    }

    auto cluster = createOutcome.GetResult();
```

```
    std::cout << "Created " << cluster.GetArn() << std::endl;

    return cluster;
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region for the single-region setup
            Aws::String region = "us-east-1";

            auto cluster = CreateCluster(region);

            std::cout << "Created single region cluster:" << std::endl;
            std::cout << "Cluster ARN: " << cluster.GetArn() << std::endl;
            std::cout << "Cluster Status: " <<
 ClusterStatusMapper::GetNameForClusterStatus(cluster.GetStatus()) << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/CreateClusterRequest.h>
#include <aws/dsql/model/UpdateClusterRequest.h>
#include <aws/dsql/model/MultiRegionProperties.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>
#include <thread>
#include <chrono>

using namespace Aws;
```

```
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Creates multi-region clusters in Amazon Aurora DSQL
 */
std::pair<CreateClusterResult, CreateClusterResult> CreateMultiRegionClusters(
    const Aws::String& region1,
    const Aws::String& region2,
    const Aws::String& witnessRegion) {

    // Create clients for each region
    DSQL::DSQLClientConfiguration clientConfig1;
    clientConfig1.region = region1;
    DSQL::DSQLClient client1(clientConfig1);

    DSQL::DSQLClientConfiguration clientConfig2;
    clientConfig2.region = region2;
    DSQL::DSQLClient client2(clientConfig2);

    // We can only set the witness region for the first cluster
    std::cout << "Creating cluster in " << region1 << std::endl;

    CreateClusterRequest createClusterRequest1;
    createClusterRequest1.SetDeletionProtectionEnabled(true);

    // Set multi-region properties with witness region
    MultiRegionProperties multiRegionProps1;
    multiRegionProps1.SetWitnessRegion(witnessRegion);
    createClusterRequest1.SetMultiRegionProperties(multiRegionProps1);

    // Add tags
    Aws::Map<Aws::String, Aws::String> tags;
    tags["Name"] = "cpp multi region cluster 1";
    createClusterRequest1.SetTags(tags);
    createClusterRequest1.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto createOutcome1 = client1.CreateCluster(createClusterRequest1);
    if (!createOutcome1.IsSuccess()) {
        std::cerr << "Failed to create cluster in " << region1 << ": "
                  << createOutcome1.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to create multi-region clusters");
    }
```

```
    auto cluster1 = createOutcome1.GetResult();
    std::cout << "Created " << cluster1.GetArn() << std::endl;

    // For the second cluster we can set witness region and designate cluster1 as a
peer
    std::cout << "Creating cluster in " << region2 << std::endl;

    CreateClusterRequest createClusterRequest2;
    createClusterRequest2.SetDeletionProtectionEnabled(true);

    // Set multi-region properties with witness region and cluster1 as peer
    MultiRegionProperties multiRegionProps2;
    multiRegionProps2.SetWitnessRegion(witnessRegion);

    Aws::Vector<Aws::String> clusters;
    clusters.push_back(cluster1.GetArn());
    multiRegionProps2.SetClusters(clusters);

    tags["Name"] = "cpp multi region cluster 2";
    createClusterRequest2.SetMultiRegionProperties(multiRegionProps2);
    createClusterRequest2.SetTags(tags);
    createClusterRequest2.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto createOutcome2 = client2.CreateCluster(createClusterRequest2);
    if (!createOutcome2.IsSuccess()) {
        std::cerr << "Failed to create cluster in " << region2 << ": "
                  << createOutcome2.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to create multi-region clusters");
    }

    auto cluster2 = createOutcome2.GetResult();
    std::cout << "Created " << cluster2.GetArn() << std::endl;

    // Now that we know the cluster2 arn we can set it as a peer of cluster1
    UpdateClusterRequest updateClusterRequest;
    updateClusterRequest.SetIdentifier(cluster1.GetIdentifier());

    MultiRegionProperties updatedProps;
    updatedProps.SetWitnessRegion(witnessRegion);

    Aws::Vector<Aws::String> updatedClusters;
    updatedClusters.push_back(cluster2.GetArn());
    updatedProps.SetClusters(updatedClusters);
```

```
    updateClusterRequest.SetMultiRegionProperties(updatedProps);
    updateClusterRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto updateOutcome = client1.UpdateCluster(updateClusterRequest);
    if (!updateOutcome.IsSuccess()) {
        std::cerr << "Failed to update cluster in " << region1 << ": "
                  << updateOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to update multi-region clusters");
    }

    std::cout << "Added " << cluster2.GetArn() << " as a peer of " <<
 cluster1.GetArn() << std::endl;

    return std::make_pair(cluster1, cluster2);
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define regions for the multi-region setup
            Aws::String region1 = "us-east-1";
            Aws::String region2 = "us-east-2";
            Aws::String witnessRegion = "us-west-2";

            auto [cluster1, cluster2] = CreateMultiRegionClusters(region1, region2,
 witnessRegion);

            std::cout << "Created multi region clusters:" << std::endl;
            std::cout << "Cluster 1 ARN: " << cluster1.GetArn() << std::endl;
            std::cout << "Cluster 2 ARN: " << cluster2.GetArn() << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Untuk membuat cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
import { DSQLClient, CreateClusterCommand, waitUntilClusterActive } from "@aws-sdk/
client-dsql";

async function createCluster(region) {

    const client = new DSQLClient({ region });

    try {
        const createClusterCommand = new CreateClusterCommand({
            deletionProtectionEnabled: true,
            tags: {
                Name: "javascript single region cluster"
            },
        });
        const response = await client.send(createClusterCommand);

        console.log(`Waiting for cluster ${response.identifier} to become ACTIVE`);
        await waitUntilClusterActive(
            {
                client: client,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response.identifier
            }
        );
        console.log(`Cluster Id ${response.identifier} is now active`);
        return;
    } catch (error) {
        console.error(`Unable to create cluster in ${region}: `, error.message);
        throw error;
    }
}

async function main() {
    const region = "us-east-1";

    await createCluster(region);
}

main();
```

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
import { DSQLClient, CreateClusterCommand, UpdateClusterCommand,
 waitUntilClusterActive } from "@aws-sdk/client-dsql";

async function createMultiRegionCluster(region1, region2, witnessRegion) {

    const client1 = new DSQLClient({ region: region1 });
    const client2 = new DSQLClient({ region: region2 });

    try {
        // We can only set the witness region for the first cluster
        console.log(`Creating cluster in ${region1}`);
        const createClusterCommand1 = new CreateClusterCommand({
            deletionProtectionEnabled: true,
            tags: {
                Name: "javascript multi region cluster 1"
            },
            multiRegionProperties: {
                witnessRegion: witnessRegion
            }
        });

        const response1 = await client1.send(createClusterCommand1);
        console.log(`Created ${response1.arn}`);

        // For the second cluster we can set witness region and designate the first
 cluster as a peer
        console.log(`Creating cluster in ${region2}`);
        const createClusterCommand2 = new CreateClusterCommand({
            deletionProtectionEnabled: true,
            tags: {
                Name: "javascript multi region cluster 2"
            },
            multiRegionProperties: {
                witnessRegion: witnessRegion,
                clusters: [response1.arn]
            }
        });

        const response2 = await client2.send(createClusterCommand2);
        console.log(`Created ${response2.arn}`);
```

```
        // Now that we know the second cluster arn we can set it as a peer of the
first cluster
        const updateClusterCommand1 = new UpdateClusterCommand(
            {
                identifier: response1.identifier,
                multiRegionProperties: {
                    witnessRegion: witnessRegion,
                    clusters: [response2.arn]
                }
            }
        );

        await client1.send(updateClusterCommand1);
        console.log(`Added ${response2.arn} as a peer of ${response1.arn}`);

        // Now that multiRegionProperties is fully defined for both clusters
        // they'll begin the transition to ACTIVE
        console.log(`Waiting for cluster 1 ${response1.identifier} to become
ACTIVE`);

        await waitUntilClusterActive(
            {
                client: client1,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response1.identifier
            }
        );
        console.log(`Cluster 1 is now active`);

        console.log(`Waiting for cluster 2 ${response2.identifier} to become
ACTIVE`);
        await waitUntilClusterActive(
            {
                client: client2,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response2.identifier
            }
        );
        console.log(`Cluster 2 is now active`);
```

```
        console.log("The multi region clusters are now active");
        return;
    } catch (error) {
        console.error("Failed to create cluster: ", error.message);
        throw error;
    }
}

async function main() {
    const region1 = "us-east-1";
    const region2 = "us-east-2";
    const witnessRegion = "us-west-2";

    await createMultiRegionCluster(region1, region2, witnessRegion);
}

main();
```

Java

Gunakan contoh berikut untuk membuat cluster dalam satu Wilayah AWS.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.retries.api.BackoffStrategy;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.CreateClusterRequest;
import software.amazon.awssdk.services.dsql.model.CreateClusterResponse;
import software.amazon.awssdk.services.dsql.model.GetClusterResponse;

import java.time.Duration;
import java.util.Map;

public class CreateCluster {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;

        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
```

```
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            CreateClusterRequest request = CreateClusterRequest.builder()
                    .deletionProtectionEnabled(true)
                    .tags(Map.of("Name", "java single region cluster"))
                    .build();
            CreateClusterResponse cluster = client.createCluster(request);
            System.out.println("Created " + cluster.arn());

            // The DSQL SDK offers a built-in waiter to poll for a cluster's
            // transition to ACTIVE.
            System.out.println("Waiting for cluster to become ACTIVE");
            WaiterResponse<GetClusterResponse> waiterResponse =
 client.waiter().waitUntilClusterActive(
                    getCluster -> getCluster.identifier(cluster.identifier()),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            );
            waiterResponse.matched().response().ifPresent(System.out::println);
        }
    }
}
```

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.retries.api.BackoffStrategy;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.DsqlClientBuilder;
import software.amazon.awssdk.services.dsql.model.CreateClusterRequest;
import software.amazon.awssdk.services.dsql.model.CreateClusterResponse;
import software.amazon.awssdk.services.dsql.model.GetClusterResponse;
import software.amazon.awssdk.services.dsql.model.UpdateClusterRequest;

import java.time.Duration;
```

```
import java.util.Map;

public class CreateMultiRegionCluster {

    public static void main(String[] args) {
        Region region1 = Region.US_EAST_1;
        Region region2 = Region.US_EAST_2;
        Region witnessRegion = Region.US_WEST_2;

        DsqlClientBuilder clientBuilder = DsqlClient.builder()
                .credentialsProvider(DefaultCredentialsProvider.create());

        try (
            DsqlClient client1 = clientBuilder.region(region1).build();
            DsqlClient client2 = clientBuilder.region(region2).build()
        ) {
            // We can only set the witness region for the first cluster
            System.out.println("Creating cluster in " + region1);
            CreateClusterRequest request1 = CreateClusterRequest.builder()
                    .deletionProtectionEnabled(true)
                    .multiRegionProperties(mrp ->
 mrp.witnessRegion(witnessRegion.toString()))
                    .tags(Map.of("Name", "java multi region cluster"))
                    .build();
            CreateClusterResponse cluster1 = client1.createCluster(request1);
            System.out.println("Created " + cluster1.arn());

            // For the second cluster we can set the witness region and designate
            // cluster1 as a peer.
            System.out.println("Creating cluster in " + region2);
            CreateClusterRequest request2 = CreateClusterRequest.builder()
                    .deletionProtectionEnabled(true)
                    .multiRegionProperties(mrp ->

 mrp.witnessRegion(witnessRegion.toString()).clusters(cluster1.arn())
                    )
                    .tags(Map.of("Name", "java multi region cluster"))
                    .build();
            CreateClusterResponse cluster2 = client2.createCluster(request2);
            System.out.println("Created " + cluster2.arn());

            // Now that we know the cluster2 ARN we can set it as a peer of cluster1
            UpdateClusterRequest updateReq = UpdateClusterRequest.builder()
                    .identifier(cluster1.identifier())
```

```
                    .multiRegionProperties(mrp ->

 mrp.witnessRegion(witnessRegion.toString()).clusters(cluster2.arn())
                    )
                    .build();
            client1.updateCluster(updateReq);
            System.out.printf("Added %s as a peer of %s%n", cluster2.arn(),
 cluster1.arn());

            // Now that MultiRegionProperties is fully defined for both clusters
 they'll begin
            // the transition to ACTIVE.
            System.out.printf("Waiting for cluster %s to become ACTIVE%n",
 cluster1.arn());
            GetClusterResponse activeCluster1 =
 client1.waiter().waitUntilClusterActive(
                    getCluster -> getCluster.identifier(cluster1.identifier()),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            ).matched().response().orElseThrow();

            System.out.printf("Waiting for cluster %s to become ACTIVE%n",
 cluster2.arn());
            GetClusterResponse activeCluster2 =
 client2.waiter().waitUntilClusterActive(
                    getCluster -> getCluster.identifier(cluster2.identifier()),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            ).matched().response().orElseThrow();

            System.out.println("Created multi region clusters:");
            System.out.println(activeCluster1);
            System.out.println(activeCluster2);
        }
    }
}
```

Rust

Gunakan contoh berikut untuk membuat cluster dalam satu Wilayah AWS.

```
use aws_config::load_defaults;
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};
use std::collections::HashMap;

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Create a cluster without delete protection and a name
pub async fn create_cluster(region: &'static str) -> GetClusterOutput {
    let client = dsql_client(region).await;
    let tags = HashMap::from([(
        String::from("Name"),
        String::from("rust single region cluster"),
    )]);

    let create_cluster_output = client
        .create_cluster()
        .set_tags(Some(tags))
        .deletion_protection_enabled(true)
```

```
        .send()
        .await
        .unwrap();
    println!("Created {}", create_cluster_output.arn);

    println!("Waiting for cluster to become ACTIVE");
    client
        .wait_until_cluster_active()
        .identifier(&create_cluster_output.identifier)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap()
        .into_result()
        .unwrap()
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";
    let output = create_cluster(region).await;
    println!("{:#?}", output);
    Ok(())
}
```

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
use aws_config::{BehaviorVersion, Region, load_defaults};
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::types::MultiRegionProperties;
use aws_sdk_dsql::{Client, Config};
use std::collections::HashMap;

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();
```

```
    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Create a cluster without delete protection and a name
pub async fn create_multi_region_clusters(
    region_1: &'static str,
    region_2: &'static str,
    witness_region: &'static str,
) -> (GetClusterOutput, GetClusterOutput) {
    let client_1 = dsql_client(region_1).await;
    let client_2 = dsql_client(region_2).await;

    let tags = HashMap::from([(
        String::from("Name"),
        String::from("rust multi region cluster"),
    )]);

    // We can only set the witness region for the first cluster
    println!("Creating cluster in {region_1}");
    let cluster_1 = client_1
        .create_cluster()
        .set_tags(Some(tags.clone()))
        .deletion_protection_enabled(true)
        .multi_region_properties(
            MultiRegionProperties::builder()
                .witness_region(witness_region)
                .build(),
        )
        .send()
        .await
        .unwrap();
    let cluster_1_arn = &cluster_1.arn;
    println!("Created {cluster_1_arn}");

    // For the second cluster we can set witness region and designate cluster_1 as a
 peer
    println!("Creating cluster in {region_2}");
```

```
    let cluster_2 = client_2
        .create_cluster()
        .set_tags(Some(tags))
        .deletion_protection_enabled(true)
        .multi_region_properties(
            MultiRegionProperties::builder()
                .witness_region(witness_region)
                .clusters(&cluster_1.arn)
                .build(),
        )
        .send()
        .await
        .unwrap();
    let cluster_2_arn = &cluster_2.arn;
    println!("Created {cluster_2_arn}");

    // Now that we know the cluster_2 arn we can set it as a peer of cluster_1
    client_1
        .update_cluster()
        .identifier(&cluster_1.identifier)
        .multi_region_properties(
            MultiRegionProperties::builder()
                .witness_region(witness_region)
                .clusters(&cluster_2.arn)
                .build(),
        )
        .send()
        .await
        .unwrap();
    println!("Added {cluster_2_arn} as a peer of {cluster_1_arn}");

    // Now that the multi-region properties are fully defined for both clusters
    // they'll begin the transition to ACTIVE
    println!("Waiting for {cluster_1_arn} to become ACTIVE");
    let cluster_1_output = client_1
        .wait_until_cluster_active()
        .identifier(&cluster_1.identifier)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap()
        .into_result()
        .unwrap();

    println!("Waiting for {cluster_2_arn} to become ACTIVE");
```

```
    let cluster_2_output = client_2
        .wait_until_cluster_active()
        .identifier(&cluster_2.identifier)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap()
        .into_result()
        .unwrap();

    (cluster_1_output, cluster_2_output)
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region_1 = "us-east-1";
    let region_2 = "us-east-2";
    let witness_region = "us-west-2";

    let (cluster_1, cluster_2) =
        create_multi_region_clusters(region_1, region_2, witness_region).await;

    println!("Created multi region clusters:");
    println!("{:#?}", cluster_1);
    println!("{:#?}", cluster_2);

    Ok(())
}
```

Ruby

Gunakan contoh berikut untuk membuat cluster dalam satu Wilayah AWS.

```
require "aws-sdk-dsql"
require "pp"

def create_cluster(region)
  client = Aws::DSQL::Client.new(region: region)
  cluster = client.create_cluster(
    deletion_protection_enabled: true,
    tags: {
      Name: "ruby single region cluster"
    }
```

```
  )
  puts "Created #{cluster.arn}"

  # The DSQL SDK offers built-in waiters to poll for a cluster's
  # transition to ACTIVE.
  puts "Waiting for cluster to become ACTIVE"
  client.wait_until(:cluster_active, identifier: cluster.identifier) do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
rescue Aws::Errors::ServiceError => e
  abort "Unable to create cluster in #{region}: #{e.message}"
end

def main
  region = "us-east-1"
  cluster = create_cluster(region)
  pp cluster
end

main if $PROGRAM_NAME == __FILE__
```

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
require "aws-sdk-dsql"
require "pp"

def create_multi_region_clusters(region_1, region_2, witness_region)
  client_1 = Aws::DSQL::Client.new(region: region_1)
  client_2 = Aws::DSQL::Client.new(region: region_2)

  # We can only set the witness region for the first cluster
  puts "Creating cluster in #{region_1}"
  cluster_1 = client_1.create_cluster(
    deletion_protection_enabled: true,
    multi_region_properties: {
      witness_region: witness_region
    },
    tags: {
```

```
    Name: "ruby multi region cluster"
  }
)
puts "Created #{cluster_1.arn}"

# For the second cluster we can set witness region and designate cluster_1 as a
peer
puts "Creating cluster in #{region_2}"
cluster_2 = client_2.create_cluster(
  deletion_protection_enabled: true,
  multi_region_properties: {
    witness_region: witness_region,
    clusters: [ cluster_1.arn ]
  },
  tags: {
    Name: "ruby multi region cluster"
  }
)
puts "Created #{cluster_2.arn}"

# Now that we know the cluster_2 arn we can set it as a peer of cluster_1
client_1.update_cluster(
  identifier: cluster_1.identifier,
  multi_region_properties: {
    witness_region: witness_region,
    clusters: [ cluster_2.arn ]
  }
)
puts "Added #{cluster_2.arn} as a peer of #{cluster_1.arn}"

# Now that multi_region_properties is fully defined for both clusters
# they'll begin the transition to ACTIVE
puts "Waiting for #{cluster_1.arn} to become ACTIVE"
cluster_1 = client_1.wait_until(:cluster_active, identifier: cluster_1.identifier)
do |w|
  # Wait for 5 minutes
  w.max_attempts = 30
  w.delay = 10
end

puts "Waiting for #{cluster_2.arn} to become ACTIVE"
cluster_2 = client_2.wait_until(:cluster_active, identifier: cluster_2.identifier)
do |w|
  w.max_attempts = 30
```

```
    w.delay = 10
  end

  [ cluster_1, cluster_2 ]
rescue Aws::Errors::ServiceError => e
  abort "Failed to create multi-region clusters: #{e.message}"
end

def main
  region_1 = "us-east-1"
  region_2 = "us-east-2"
  witness_region = "us-west-2"

  cluster_1, cluster_2 = create_multi_region_clusters(region_1, region_2,
 witness_region)

  puts "Created multi region clusters:"
  pp cluster_1
  pp cluster_2
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Gunakan contoh berikut untuk membuat cluster dalam satu Wilayah AWS.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
    public class CreateSingleRegionCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
```

```
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
region)
        {
            var awsCredentials = await
DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Create a cluster without delete protection and a name.
        /// </summary>
        public static async Task<CreateClusterResponse> Create(RegionEndpoint
region)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var tags = new Dictionary<string, string>
                {
                    { "Name", "csharp single region cluster" }
                };

                var createClusterRequest = new CreateClusterRequest
                {
                    DeletionProtectionEnabled = true,
                    Tags = tags
                };

                CreateClusterResponse response = await
client.CreateClusterAsync(createClusterRequest);
                Console.WriteLine($"Initiated creation of {response.Arn}");

                return response;
            }
        }

        private static async Task Main()
        {
            var region = RegionEndpoint.USEast1;

            await Create(region);
```

```
        }
    }
}
```

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;
using Amazon.Runtime.Endpoints;

namespace DSQLExamples.examples
{
    public class CreateMultiRegionClusters
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region,
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Create multi-region clusters with a witness region.
        /// </summary>
        public static async Task<(CreateClusterResponse, CreateClusterResponse)>
 Create(
            RegionEndpoint region1,
            RegionEndpoint region2,
```

```
            RegionEndpoint witnessRegion)
        {
            using (var client1 = await CreateDSQLClient(region1))
            using (var client2 = await CreateDSQLClient(region2))
            {
                var tags = new Dictionary<string, string>
                {
                    { "Name", "csharp multi region cluster" }
                };

                // We can only set the witness region for the first cluster
                var createClusterRequest1 = new CreateClusterRequest
                {
                    DeletionProtectionEnabled = true,
                    Tags = tags,
                    MultiRegionProperties = new MultiRegionProperties
                    {
                        WitnessRegion = witnessRegion.SystemName
                    }
                };

                var cluster1 = await
client1.CreateClusterAsync(createClusterRequest1);
                var cluster1Arn = cluster1.Arn;
                Console.WriteLine($"Initiated creation of {cluster1Arn}");

                // For the second cluster we can set witness region and designate
cluster1 as a peer
                var createClusterRequest2 = new CreateClusterRequest
                {
                    DeletionProtectionEnabled = true,
                    Tags = tags,
                    MultiRegionProperties = new MultiRegionProperties
                    {
                        WitnessRegion = witnessRegion.SystemName,
                        Clusters = new List<string> { cluster1.Arn }
                    }
                };

                var cluster2 = await
client2.CreateClusterAsync(createClusterRequest2);
                var cluster2Arn = cluster2.Arn;
                Console.WriteLine($"Initiated creation of {cluster2Arn}");
```

```
                // Now that we know the cluster2 arn we can set it as a peer of
 cluster1
                var updateClusterRequest = new UpdateClusterRequest
                {
                    Identifier = cluster1.Identifier,
                    MultiRegionProperties = new MultiRegionProperties
                    {
                        WitnessRegion = witnessRegion.SystemName,
                        Clusters = new List<string> { cluster2.Arn }
                    }
                };

                await client1.UpdateClusterAsync(updateClusterRequest);
                Console.WriteLine($"Added {cluster2Arn} as a peer of
 {cluster1Arn}");

                return (cluster1, cluster2);
            }
        }

        private static async Task Main()
        {
            var region1 = RegionEndpoint.USEast1;
            var region2 = RegionEndpoint.USEast2;
            var witnessRegion = RegionEndpoint.USWest2;

            var (cluster1, cluster2) = await Create(region1, region2,
 witnessRegion);

            Console.WriteLine("Created multi region clusters:");
            Console.WriteLine($"Cluster 1: {cluster1.Arn}");
            Console.WriteLine($"Cluster 2: {cluster2.Arn}");
        }
    }
}
```

Golang

Gunakan contoh berikut untuk membuat cluster dalam satu Wilayah AWS.

```
package main

import (
```

```
 "context"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func CreateCluster(ctx context.Context, region string) error {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Create a DSQL client
 client := dsql.NewFromConfig(cfg)

 deleteProtect := true

 input := dsql.CreateClusterInput{
  DeletionProtectionEnabled: &deleteProtect,
  Tags: map[string]string{
   "Name": "go single region cluster",
  },
 }

 clusterProperties, err := client.CreateCluster(context.Background(), &input)

 if err != nil {
  return fmt.Errorf("error creating cluster: %w", err)
 }

 fmt.Printf("Created cluster: %s\n", *clusterProperties.Arn)

 // Create the waiter with our custom options
 waiter := dsql.NewClusterActiveWaiter(client, func(o
 *dsql.ClusterActiveWaiterOptions) {
  o.MaxDelay = 30 * time.Second
  o.MinDelay = 10 * time.Second
  o.LogWaitAttempts = true
 })
```

```
 id := clusterProperties.Identifier

 // Create the input for the clusterProperties
 getInput := &dsql.GetClusterInput{
  Identifier: id,
 }

 // Wait for the cluster to become active
 fmt.Println("Waiting for cluster to become ACTIVE")
 err = waiter.Wait(ctx, getInput, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for cluster to become active: %w", err)
 }

 fmt.Printf("Cluster %s is now active\n", *id)
 return nil
}

// Example usage in main function
func main() {

 region := "us-east-1"

 // Set up context with timeout
 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()

 if err := CreateCluster(ctx, region); err != nil {
  log.Fatalf("Failed to create cluster: %v", err)
 }
}
```

Untuk membuat cluster Multi-region, gunakan contoh berikut. Membuat klaster Multi-wilayah mungkin membutuhkan waktu.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/dsql"
 dtypes "github.com/aws/aws-sdk-go-v2/service/dsql/types"
)

func CreateMultiRegionClusters(ctx context.Context, witness, region1, region2
 string) error {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region1))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Create a DSQL region 1 client
 client := dsql.NewFromConfig(cfg)

 cfg2, err := config.LoadDefaultConfig(ctx, config.WithRegion(region2))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Create a DSQL region 2 client
 client2 := dsql.NewFromConfig(cfg2, func(o *dsql.Options) {
  o.Region = region2
 })

 // Create cluster
 deleteProtect := true

 // We can only set the witness region for the first cluster
 input := &dsql.CreateClusterInput{
  DeletionProtectionEnabled: &deleteProtect,
  MultiRegionProperties: &dtypes.MultiRegionProperties{
   WitnessRegion: aws.String(witness),
  },
  Tags: map[string]string{
   "Name": "go multi-region cluster",
  },
 }

 clusterProperties, err := client.CreateCluster(context.Background(), input)
```

```
if err != nil {
 return fmt.Errorf("failed to create first cluster: %v", err)
}

// create second cluster
cluster2Arns := []string{*clusterProperties.Arn}

// For the second cluster we can set witness region and designate the first cluster
as a peer
input2 := &dsql.CreateClusterInput{
 DeletionProtectionEnabled: &deleteProtect,
 MultiRegionProperties: &dtypes.MultiRegionProperties{
  WitnessRegion: aws.String("us-west-2"),
  Clusters:      cluster2Arns,
 },
 Tags: map[string]string{
  "Name": "go multi-region cluster",
 },
}

clusterProperties2, err := client2.CreateCluster(context.Background(), input2)

if err != nil {
 return fmt.Errorf("failed to create second cluster: %v", err)
}

// link initial cluster to second cluster
cluster1Arns := []string{*clusterProperties2.Arn}

// Now that we know the second cluster arn we can set it as a peer of the first
cluster
input3 := dsql.UpdateClusterInput{
 Identifier: clusterProperties.Identifier,
 MultiRegionProperties: &dtypes.MultiRegionProperties{
  WitnessRegion: aws.String("us-west-2"),
  Clusters:      cluster1Arns,
 }}

_, err = client.UpdateCluster(context.Background(), &input3)

if err != nil {
 return fmt.Errorf("failed to update cluster to associate with first cluster. %v",
err)
}
```

```
// Create the waiter with our custom options for first cluster
waiter := dsql.NewClusterActiveWaiter(client, func(o
*dsql.ClusterActiveWaiterOptions) {
 o.MaxDelay = 30 * time.Second // Creating a multi-region cluster can take a few
minutes
 o.MinDelay = 10 * time.Second
 o.LogWaitAttempts = true
})

// Now that multiRegionProperties is fully defined for both clusters
// they'll begin the transition to ACTIVE

// Create the input for the clusterProperties to monitor for first cluster
getInput := &dsql.GetClusterInput{
 Identifier: clusterProperties.Identifier,
}

// Wait for the first cluster to become active
fmt.Printf("Waiting for first cluster %s to become active...\n",
*clusterProperties.Identifier)
err = waiter.Wait(ctx, getInput, 5*time.Minute)
if err != nil {
 return fmt.Errorf("error waiting for first cluster to become active: %w", err)
}

// Create the waiter with our custom options
waiter2 := dsql.NewClusterActiveWaiter(client2, func(o
*dsql.ClusterActiveWaiterOptions) {
 o.MaxDelay = 30 * time.Second // Creating a multi-region cluster can take a few
minutes
 o.MinDelay = 10 * time.Second
 o.LogWaitAttempts = true
})

// Create the input for the clusterProperties to monitor for second
getInput2 := &dsql.GetClusterInput{
 Identifier: clusterProperties2.Identifier,
}

// Wait for the second cluster to become active
fmt.Printf("Waiting for second cluster %s to become active...\n",
*clusterProperties2.Identifier)
err = waiter2.Wait(ctx, getInput2, 5*time.Minute)
```

```
 if err != nil {
  return fmt.Errorf("error waiting for second cluster to become active: %w", err)
 }

 fmt.Printf("Cluster %s is now active\n", *clusterProperties.Identifier)
 fmt.Printf("Cluster %s is now active\n", *clusterProperties2.Identifier)
 return nil
}

// Example usage in main function
func main() {
 // Set up context with timeout
 ctx, cancel := context.WithTimeout(context.Background(), 10*time.Minute)
 defer cancel()

 err := CreateMultiRegionClusters(ctx, "us-west-2", "us-east-1", "us-east-2")
 if err != nil {
  fmt.Printf("failed to create multi-region clusters: %v", err)
  panic(err)
 }

}
```

Dapatkan cluster

Lihat informasi berikut untuk mempelajari cara mengembalikan informasi cluster di Aurora DSQL.

Python

Untuk mendapatkan informasi tentang cluster tunggal atau Multi-region, gunakan contoh berikut.

```
import boto3
from datetime import datetime
import json


def get_cluster(region, identifier):
    try:
        client = boto3.client("dsql", region_name=region)
        return client.get_cluster(identifier=identifier)
    except:
        print(f"Unable to get cluster {identifier} in region {region}")
```

```
        raise


def main():
    region = "us-east-1"
    cluster_id = "<your cluster id>"
    response = get_cluster(region, cluster_id)

    print(json.dumps(response, indent=2, default=lambda obj: obj.isoformat() if
 isinstance(obj, datetime) else None))


if __name__ == "__main__":
    main()
```

C++

Gunakan contoh berikut untuk mendapatkan informasi tentang cluster tunggal atau Multi-region.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Retrieves information about a cluster in Amazon Aurora DSQL
 */
GetClusterResult GetCluster(const Aws::String& region, const Aws::String&
 identifier) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Get the cluster
    GetClusterRequest getClusterRequest;
    getClusterRequest.SetIdentifier(identifier);
```

```
    auto getOutcome = client.GetCluster(getClusterRequest);
    if (!getOutcome.IsSuccess()) {
        std::cerr << "Failed to retrieve cluster " << identifier << " in " << region
 << ": "
                  << getOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to retrieve cluster " + identifier + " in
 region " + region);
    }

    return getOutcome.GetResult();
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region and cluster ID
            Aws::String region = "us-east-1";
            Aws::String clusterId = "<your cluster id>";

            auto cluster = GetCluster(region, clusterId);

            // Print cluster details
            std::cout << "Cluster Details:" << std::endl;
            std::cout << "ARN: " << cluster.GetArn() << std::endl;
            std::cout << "Status: " <<
 ClusterStatusMapper::GetNameForClusterStatus(cluster.GetStatus()) << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Untuk mendapatkan informasi tentang cluster tunggal atau Multi-region, gunakan contoh berikut.

```
import { DSQLClient, GetClusterCommand } from "@aws-sdk/client-dsql";
```

```
async function getCluster(region, clusterId) {

  const client = new DSQLClient({ region });

  const getClusterCommand = new GetClusterCommand({
    identifier: clusterId,
  });

  try {
    return await client.send(getClusterCommand);
  } catch (error) {
    if (error.name === "ResourceNotFoundException") {
      console.log("Cluster ID not found or deleted");
    }
    throw error;
  }
}

async function main() {
  const region = "us-east-1";
  const clusterId = "<CLUSTER_ID>";

  const response = await getCluster(region, clusterId);
  console.log("Cluster: ", response);
}

main();
```

Java

Contoh berikut memungkinkan Anda mendapatkan informasi tentang cluster tunggal atau Multi-region.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetClusterResponse;
import software.amazon.awssdk.services.dsql.model.ResourceNotFoundException;
```

```
public class GetCluster {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        String clusterId = "<your cluster id>";

        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            GetClusterResponse cluster = client.getCluster(r ->
 r.identifier(clusterId));
            System.out.println(cluster);
        } catch (ResourceNotFoundException e) {
            System.out.printf("Cluster %s not found in %s%n", clusterId, region);
        }
    }
}
```

Rust

Contoh berikut memungkinkan Anda mendapatkan informasi tentang cluster tunggal atau Multi-region.

```
use aws_config::load_defaults;
use aws_sdk_dsql::operation::get_cluster::GetClusterOutput;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();
```

```
    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Get a ClusterResource from DSQL cluster identifier
pub async fn get_cluster(region: &'static str, identifier: &'static str) ->
 GetClusterOutput {
    let client = dsql_client(region).await;
    client
        .get_cluster()
        .identifier(identifier)
        .send()
        .await
        .unwrap()
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";

    let cluster = get_cluster(region, "<your cluster id>").await;
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

Contoh berikut memungkinkan Anda mendapatkan informasi tentang cluster tunggal atau Multi-region.

```
require "aws-sdk-dsql"
require "pp"

def get_cluster(region, identifier)
  client = Aws::DSQL::Client.new(region: region)
```

```
  client.get_cluster(identifier: identifier)
rescue Aws::Errors::ServiceError => e
  abort "Unable to retrieve cluster #{identifier} in region #{region}: #{e.message}"
end

def main
  region = "us-east-1"
  cluster_id = "<your cluster id>"
  cluster = get_cluster(region, cluster_id)
  pp cluster
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Contoh berikut memungkinkan Anda mendapatkan informasi tentang cluster tunggal atau Multi-region.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
    public class GetCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region
```

```
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Get information about a DSQL cluster.
        /// </summary>
        public static async Task<GetClusterResponse> Get(RegionEndpoint region,
 string identifier)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var getClusterRequest = new GetClusterRequest
                {
                    Identifier = identifier
                };

                return await client.GetClusterAsync(getClusterRequest);
            }
        }

        private static async Task Main()
        {
            var region = RegionEndpoint.USEast1;
            var clusterId = "<your cluster id>";

            var response = await Get(region, clusterId);
            Console.WriteLine($"Cluster ARN: {response.Arn}");
        }
    }
}
```

Golang

Contoh berikut memungkinkan Anda mendapatkan informasi tentang cluster tunggal atau Multi-region.

```
package main

import (
 "context"
 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/config"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/service/dsql"
)

func GetCluster(ctx context.Context, region, identifier string) (clusterStatus
 *dsql.GetClusterOutput, err error) {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Initialize the DSQL client
 client := dsql.NewFromConfig(cfg)

 input := &dsql.GetClusterInput{
  Identifier: aws.String(identifier),
 }
 clusterStatus, err = client.GetCluster(context.Background(), input)

 if err != nil {
  log.Fatalf("Failed to get cluster: %v", err)
 }

 log.Printf("Cluster ARN: %s", *clusterStatus.Arn)

 return clusterStatus, nil
}

func main() {
 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()

 // Example cluster identifier
 identifier := "<CLUSTER_ID>"
 region := "us-east-1"

 _, err := GetCluster(ctx, region, identifier)
 if err != nil {
  log.Fatalf("Failed to get cluster: %v", err)
 }
```

```
}
```

Perbarui klaster

Lihat informasi berikut untuk mempelajari cara memperbarui cluster di Aurora DSQL. Memperbarui cluster bisa memakan waktu satu atau dua menit. Kami menyarankan Anda menunggu beberapa saat dan kemudian menjalankan get cluster untuk mendapatkan status cluster.

Python

Untuk memperbarui cluster tunggal atau Multi-region, gunakan contoh berikut.

```
import boto3


def update_cluster(region, cluster_id, deletion_protection_enabled):
    try:
        client = boto3.client("dsql", region_name=region)
        return client.update_cluster(identifier=cluster_id,
 deletionProtectionEnabled=deletion_protection_enabled)
    except:
        print("Unable to update cluster")
        raise


def main():
    region = "us-east-1"
    cluster_id = "<your cluster id>"
    deletion_protection_enabled = False
    response = update_cluster(region, cluster_id, deletion_protection_enabled)
    print(f"Updated {response["arn"]} with deletion_protection_enabled:
 {deletion_protection_enabled}")


if __name__ == "__main__":
    main()
```

C++

Gunakan contoh berikut untuk memperbarui cluster tunggal atau Multi-wilayah.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/UpdateClusterRequest.h>
#include <iostream>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Updates a cluster in Amazon Aurora DSQL
 */
UpdateClusterResult UpdateCluster(const Aws::String& region, const
 Aws::Map<Aws::String, Aws::String>& updateParams) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Create update request
    UpdateClusterRequest updateRequest;
    updateRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    // Set identifier (required)
    if (updateParams.find("identifier") != updateParams.end()) {
        updateRequest.SetIdentifier(updateParams.at("identifier"));
    } else {
        throw std::runtime_error("Cluster identifier is required for update
 operation");
    }

    // Set deletion protection if specified
    if (updateParams.find("deletion_protection_enabled") != updateParams.end()) {
        bool deletionProtection = (updateParams.at("deletion_protection_enabled") ==
 "true");
        updateRequest.SetDeletionProtectionEnabled(deletionProtection);
    }

    // Execute the update
    auto updateOutcome = client.UpdateCluster(updateRequest);
    if (!updateOutcome.IsSuccess()) {
```

```
        std::cerr << "Failed to update cluster: " <<
 updateOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to update cluster");
    }

    return updateOutcome.GetResult();
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region and update parameters
            Aws::String region = "us-east-1";
            Aws::String clusterId = "<your cluster id>";

            // Create parameter map
            Aws::Map<Aws::String, Aws::String> updateParams;
            updateParams["identifier"] = clusterId;
            updateParams["deletion_protection_enabled"] = "false";

            auto updatedCluster = UpdateCluster(region, updateParams);

            std::cout << "Updated " << updatedCluster.GetArn() << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Untuk memperbarui cluster tunggal atau Multi-region, gunakan contoh berikut.

```
import { DSQLClient, UpdateClusterCommand } from "@aws-sdk/client-dsql";

export async function updateCluster(region, clusterId, deletionProtectionEnabled) {

  const client = new DSQLClient({ region });
```

```
  const updateClusterCommand = new UpdateClusterCommand({
    identifier: clusterId,
    deletionProtectionEnabled: deletionProtectionEnabled
  });

  try {
    return await client.send(updateClusterCommand);
  } catch (error) {
    console.error("Unable to update cluster", error.message);
    throw error;
  }
}

async function main() {
  const region = "us-east-1";
  const clusterId = "<CLUSTER_ID>";
  const deletionProtectionEnabled = false;

  const response = await updateCluster(region, clusterId,
 deletionProtectionEnabled);
  console.log(`Updated ${response.arn}`);
}

main();
```

Java

Gunakan contoh berikut untuk memperbarui cluster tunggal atau Multi-region.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.UpdateClusterRequest;
import software.amazon.awssdk.services.dsql.model.UpdateClusterResponse;

public class UpdateCluster {

    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        String clusterId = "<your cluster id>";
```

```
        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            UpdateClusterRequest request = UpdateClusterRequest.builder()
                    .identifier(clusterId)
                    .deletionProtectionEnabled(false)
                    .build();
            UpdateClusterResponse cluster = client.updateCluster(request);
            System.out.println("Updated " + cluster.arn());
        }
    }
}
```

Rust

Gunakan contoh berikut untuk memperbarui cluster tunggal atau Multi-region.

```
use aws_config::load_defaults;
use aws_sdk_dsql::operation::update_cluster::UpdateClusterOutput;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
```

```
}

/// Update a DSQL cluster and set delete protection to false. Also add new tags.
pub async fn update_cluster(region: &'static str, identifier: &'static str) ->
 UpdateClusterOutput {
    let client = dsql_client(region).await;
    // Update delete protection
    let update_response = client
        .update_cluster()
        .identifier(identifier)
        .deletion_protection_enabled(false)
        .send()
        .await
        .unwrap();

    update_response
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";

    let cluster = update_cluster(region, "<your cluster id>").await;
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

Gunakan contoh berikut untuk memperbarui cluster tunggal atau Multi-region.

```
require "aws-sdk-dsql"

def update_cluster(region, update_params)
  client = Aws::DSQL::Client.new(region: region)
  client.update_cluster(update_params)
rescue Aws::Errors::ServiceError => e
  abort "Unable to update cluster: #{e.message}"
end

def main
  region = "us-east-1"
  cluster_id = "<your cluster id>"
```

```
  updated_cluster = update_cluster(region, {
    identifier: cluster_id,
    deletion_protection_enabled: false
  })
  puts "Updated #{updated_cluster.arn}"
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Gunakan contoh berikut untuk memperbarui cluster tunggal atau Multi-region.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
    public class UpdateCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Update a DSQL cluster and set delete protection to false.
        /// </summary>
```

```
        public static async Task<UpdateClusterResponse> Update(RegionEndpoint
 region, string identifier)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var updateClusterRequest = new UpdateClusterRequest
                {
                    Identifier = identifier,
                    DeletionProtectionEnabled = false
                };

                UpdateClusterResponse response = await
 client.UpdateClusterAsync(updateClusterRequest);
                Console.WriteLine($"Updated {response.Arn}");

                return response;
            }
        }

        private static async Task Main()
        {
            var region = RegionEndpoint.USEast1;
            var clusterId = "<your cluster id>";

            await Update(region, clusterId);
        }
    }
}
```

Golang

Gunakan contoh berikut untuk memperbarui cluster tunggal atau Multi-region.

```
package main

import (
 "context"
 "github.com/aws/aws-sdk-go-v2/config"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/service/dsql"
)
```

```
func UpdateCluster(ctx context.Context, region, id string, deleteProtection bool)
 (clusterStatus *dsql.UpdateClusterOutput, err error) {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Initialize the DSQL client
 client := dsql.NewFromConfig(cfg)

 input := dsql.UpdateClusterInput{
  Identifier:                &id,
  DeletionProtectionEnabled: &deleteProtection,
 }

 clusterStatus, err = client.UpdateCluster(context.Background(), &input)

 if err != nil {
  log.Fatalf("Failed to update cluster: %v", err)
 }

 log.Printf("Cluster updated successfully: %v", clusterStatus.Status)
 return clusterStatus, nil
}

func main() {
 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()

 // Example cluster identifier
 identifier := "<CLUSTER_ID>"
 region := "us-east-1"
 deleteProtection := false

 _, err := UpdateCluster(ctx, region, identifier, deleteProtection)
 if err != nil {
  log.Fatalf("Failed to update cluster: %v", err)
 }
}
```

Hapus klaster

Lihat informasi berikut untuk mempelajari cara menghapus cluster di Aurora DSQL.

Python

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
import boto3


def delete_cluster(region, identifier):
    try:
        client = boto3.client("dsql", region_name=region)
        cluster = client.delete_cluster(identifier=identifier)
        print(f"Initiated delete of {cluster["arn"]}")

        print("Waiting for cluster to finish deletion")
        client.get_waiter("cluster_not_exists").wait(
            identifier=cluster["identifier"],
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )
    except:
        print("Unable to delete cluster " + identifier)
        raise


def main():
    region = "us-east-1"
    cluster_id = "<cluster id>"  # Use a placeholder in docs
    delete_cluster(region, cluster_id)
    print(f"Deleted {cluster_id}")


if __name__ == "__main__":
    main()
```

Untuk menghapus cluster Multi-region, gunakan contoh berikut.

```
import boto3


def delete_multi_region_clusters(region_1, cluster_id_1, region_2, cluster_id_2):
    try:

        client_1 = boto3.client("dsql", region_name=region_1)
        client_2 = boto3.client("dsql", region_name=region_2)

        client_1.delete_cluster(identifier=cluster_id_1)
        print(f"Deleting cluster {cluster_id_1} in {region_1}")

        # cluster_1 will stay in PENDING_DELETE state until cluster_2 is deleted

        client_2.delete_cluster(identifier=cluster_id_2)
        print(f"Deleting cluster {cluster_id_2} in {region_2}")

        # Now that both clusters have been marked for deletion they will transition
        # to DELETING state and finalize deletion
        print(f"Waiting for {cluster_id_1} to finish deletion")
        client_1.get_waiter("cluster_not_exists").wait(
            identifier=cluster_id_1,
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

        print(f"Waiting for {cluster_id_2} to finish deletion")
        client_2.get_waiter("cluster_not_exists").wait(
            identifier=cluster_id_2,
            WaiterConfig={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

    except:
        print("Unable to delete cluster")
        raise


def main():
```

```
    region_1 = "us-east-1"
    cluster_id_1 = "<cluster 1 id>"
    region_2 = "us-east-2"
    cluster_id_2 = "<cluster 2 id>"

    delete_multi_region_clusters(region_1, cluster_id_1, region_2, cluster_id_2)
    print(f"Deleted {cluster_id_1} in {region_1} and {cluster_id_2} in {region_2}")


if __name__ == "__main__":
    main()
```

C++

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/DeleteClusterRequest.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>
#include <thread>
#include <chrono>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Deletes a single-region cluster in Amazon Aurora DSQL
 */
void DeleteCluster(const Aws::String& region, const Aws::String& identifier) {
    // Create client for the specified region
    DSQL::DSQLClientConfiguration clientConfig;
    clientConfig.region = region;
    DSQL::DSQLClient client(clientConfig);

    // Delete the cluster
    DeleteClusterRequest deleteRequest;
    deleteRequest.SetIdentifier(identifier);
    deleteRequest.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto deleteOutcome = client.DeleteCluster(deleteRequest);
```

```
    if (!deleteOutcome.IsSuccess()) {
        std::cerr << "Failed to delete cluster " << identifier << " in " << region
 << ": "
                  << deleteOutcome.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Unable to delete cluster " + identifier + " in " +
 region);
    }

    auto cluster = deleteOutcome.GetResult();
    std::cout << "Initiated delete of " << cluster.GetArn() << std::endl;
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            // Define region and cluster ID
            Aws::String region = "us-east-1";
            Aws::String clusterId = "<your cluster id>";

            DeleteCluster(region, clusterId);

            std::cout << "Deleted " << clusterId << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
#include <aws/core/Aws.h>
#include <aws/core/utils/Outcome.h>
#include <aws/dsql/DSQLClient.h>
#include <aws/dsql/model/DeleteClusterRequest.h>
#include <aws/dsql/model/GetClusterRequest.h>
#include <iostream>
#include <thread>
```

```
#include <chrono>

using namespace Aws;
using namespace Aws::DSQL;
using namespace Aws::DSQL::Model;

/**
 * Deletes multi-region clusters in Amazon Aurora DSQL
 */
void DeleteMultiRegionClusters(
    const Aws::String& region1,
    const Aws::String& clusterId1,
    const Aws::String& region2,
    const Aws::String& clusterId2) {

    // Create clients for each region
    DSQL::DSQLClientConfiguration clientConfig1;
    clientConfig1.region = region1;
    DSQL::DSQLClient client1(clientConfig1);

    DSQL::DSQLClientConfiguration clientConfig2;
    clientConfig2.region = region2;
    DSQL::DSQLClient client2(clientConfig2);

    // Delete the first cluster
    std::cout << "Deleting cluster " << clusterId1 << " in " << region1 <<
 std::endl;

    DeleteClusterRequest deleteRequest1;
    deleteRequest1.SetIdentifier(clusterId1);
    deleteRequest1.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto deleteOutcome1 = client1.DeleteCluster(deleteRequest1);
    if (!deleteOutcome1.IsSuccess()) {
        std::cerr << "Failed to delete cluster " << clusterId1 << " in " << region1
 << ": "
                  << deleteOutcome1.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to delete multi-region clusters");
    }

    // cluster1 will stay in PENDING_DELETE state until cluster2 is deleted
    std::cout << "Deleting cluster " << clusterId2 << " in " << region2 <<
 std::endl;
```

```
    DeleteClusterRequest deleteRequest2;
    deleteRequest2.SetIdentifier(clusterId2);
    deleteRequest2.SetClientToken(Aws::Utils::UUID::RandomUUID());

    auto deleteOutcome2 = client2.DeleteCluster(deleteRequest2);
    if (!deleteOutcome2.IsSuccess()) {
        std::cerr << "Failed to delete cluster " << clusterId2 << " in " << region2
 << ": "
                  << deleteOutcome2.GetError().GetMessage() << std::endl;
        throw std::runtime_error("Failed to delete multi-region clusters");
    }
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    {
        try {
            Aws::String region1 = "us-east-1";
            Aws::String clusterId1 = "<your cluster id 1>";
            Aws::String region2 = "us-east-2";
            Aws::String clusterId2 = "<your cluster id 2>";

            DeleteMultiRegionClusters(region1, clusterId1, region2, clusterId2);

            std::cout << "Deleted " << clusterId1 << " in " << region1
                      << " and " << clusterId2 << " in " << region2 << std::endl;
        }
        catch (const std::exception& e) {
            std::cerr << "Error: " << e.what() << std::endl;
        }
    }
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
import { DSQLClient, DeleteClusterCommand, waitUntilClusterNotExists } from "@aws-
sdk/client-dsql";

async function deleteCluster(region, clusterId) {
```

```
  const client = new DSQLClient({ region });

  try {
    const deleteClusterCommand = new DeleteClusterCommand({
      identifier: clusterId,
    });
    const response = await client.send(deleteClusterCommand);

    console.log(`Waiting for cluster ${response.identifier} to finish deletion`);

    await waitUntilClusterNotExists(
      {
        client: client,
        maxWaitTime: 300 // Wait for 5 minutes
      },
      {
        identifier: response.identifier
      }
    );
    console.log(`Cluster Id ${response.identifier} is now deleted`);
    return;
  } catch (error) {
    if (error.name === "ResourceNotFoundException") {
      console.log("Cluster ID not found or already deleted");
    } else {
      console.error("Unable to delete cluster: ", error.message);
    }
    throw error;
  }
}

async function main() {
  const region = "us-east-1";
  const clusterId = "<CLUSTER_ID>";

  await deleteCluster(region, clusterId);
}

main();
```

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
import { DSQLClient, DeleteClusterCommand, waitUntilClusterNotExists } from "@aws-
sdk/client-dsql";

async function deleteMultiRegionClusters(region1, cluster1_id, region2, cluster2_id)
 {

    const client1 = new DSQLClient({ region: region1 });
    const client2 = new DSQLClient({ region: region2 });

    try {
        const deleteClusterCommand1 = new DeleteClusterCommand({
            identifier: cluster1_id,
        });
        const response1 = await client1.send(deleteClusterCommand1);

        const deleteClusterCommand2 = new DeleteClusterCommand({
            identifier: cluster2_id,
        });
        const response2 = await client2.send(deleteClusterCommand2);

        console.log(`Waiting for cluster1 ${response1.identifier} to finish
 deletion`);
        await waitUntilClusterNotExists(
            {
                client: client1,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response1.identifier
            }
        );
        console.log(`Cluster1 Id ${response1.identifier} is now deleted`);

        console.log(`Waiting for cluster2 ${response2.identifier} to finish
 deletion`);
        await waitUntilClusterNotExists(
            {
                client: client2,
                maxWaitTime: 300 // Wait for 5 minutes
            },
            {
                identifier: response2.identifier
            }
```

```
        );
        console.log(`Cluster2 Id ${response2.identifier} is now deleted`);
        return;
    } catch (error) {
        if (error.name === "ResourceNotFoundException") {
            console.log("Some or all Cluster ARNs not found or already deleted");
        } else {
            console.error("Unable to delete multi-region clusters: ",
 error.message);
        }
        throw error;
    }
}

async function main() {
    const region1 = "us-east-1";
    const cluster1_id = "<CLUSTER_ID_1>";
    const region2 = "us-east-2";
    const cluster2_id = "<CLUSTER_ID_2>";

    const response = await deleteMultiRegionClusters(region1, cluster1_id, region2,
 cluster2_id);
    console.log(`Deleted ${cluster1_id} in ${region1} and ${cluster2_id} in
 ${region2}`);
}

main();
```

Java

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.retries.api.BackoffStrategy;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.DeleteClusterResponse;
import software.amazon.awssdk.services.dsql.model.ResourceNotFoundException;

import java.time.Duration;

public class DeleteCluster {
```

```
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        String clusterId = "<your cluster id>";


        try (
            DsqlClient client = DsqlClient.builder()
                    .region(region)
                    .credentialsProvider(DefaultCredentialsProvider.create())
                    .build()
        ) {
            DeleteClusterResponse cluster = client.deleteCluster(r ->
 r.identifier(clusterId));
            System.out.println("Initiated delete of " + cluster.arn());

            // The DSQL SDK offers a built-in waiter to poll for deletion.
            System.out.println("Waiting for cluster to finish deletion");
            client.waiter().waitUntilClusterNotExists(
                    getCluster -> getCluster.identifier(clusterId),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            );
            System.out.println("Deleted " + cluster.arn());
        } catch (ResourceNotFoundException e) {
            System.out.printf("Cluster %s not found in %s%n", clusterId, region);
        }
    }
}
```

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
package org.example;

import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.retries.api.BackoffStrategy;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.DsqlClientBuilder;
import software.amazon.awssdk.services.dsql.model.DeleteClusterRequest;
```

```
import java.time.Duration;

public class DeleteMultiRegionClusters {

    public static void main(String[] args) {
        Region region1 = Region.US_EAST_1;
        String clusterId1 = "<your cluster id 1>";
        Region region2 = Region.US_EAST_2;
        String clusterId2 = "<your cluster id 2>";

        DsqlClientBuilder clientBuilder = DsqlClient.builder()
                .credentialsProvider(DefaultCredentialsProvider.create());

        try (
            DsqlClient client1 = clientBuilder.region(region1).build();
            DsqlClient client2 = clientBuilder.region(region2).build()
        ) {
            System.out.printf("Deleting cluster %s in %s%n", clusterId1, region1);
            DeleteClusterRequest request1 = DeleteClusterRequest.builder()
                    .identifier(clusterId1)
                    .build();
            client1.deleteCluster(request1);

            // cluster1 will stay in PENDING_DELETE until cluster2 is deleted
            System.out.printf("Deleting cluster %s in %s%n", clusterId2, region2);
            DeleteClusterRequest request2 = DeleteClusterRequest.builder()
                    .identifier(clusterId2)
                    .build();
            client2.deleteCluster(request2);

            // Now that both clusters have been marked for deletion they will
 transition
            // to DELETING state and finalize deletion.
            System.out.printf("Waiting for cluster %s to finish deletion%n",
 clusterId1);
            client1.waiter().waitUntilClusterNotExists(
                    getCluster -> getCluster.identifier(clusterId1),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            );
```

```
            System.out.printf("Waiting for cluster %s to finish deletion%n",
 clusterId2);
            client2.waiter().waitUntilClusterNotExists(
                    getCluster -> getCluster.identifier(clusterId2),
                    config -> config.backoffStrategyV2(

 BackoffStrategy.fixedDelayWithoutJitter(Duration.ofSeconds(10))
                    ).waitTimeout(Duration.ofMinutes(5))
            );

            System.out.printf("Deleted %s in %s and %s in %s%n", clusterId1,
 region1, clusterId2, region2);
        }
    }
}
```

Rust

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
use aws_config::load_defaults;
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::{
    Client, Config,
    config::{BehaviorVersion, Region},
};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
```

```
}

/// Delete a DSQL cluster
pub async fn delete_cluster(region: &'static str, identifier: &'static str) {
    let client = dsql_client(region).await;
    let delete_response = client
        .delete_cluster()
        .identifier(identifier)
        .send()
        .await
        .unwrap();
    println!("Initiated delete of {}", delete_response.arn);

    println!("Waiting for cluster to finish deletion");
    client
        .wait_until_cluster_not_exists()
        .identifier(identifier)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap();
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region = "us-east-1";
    let cluster_id = "<cluster to be deleted>";

    delete_cluster(region, cluster_id).await;
    println!("Deleted {cluster_id}");

    Ok(())
}
```

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
use aws_config::{BehaviorVersion, Region, load_defaults};
use aws_sdk_dsql::client::Waiters;
use aws_sdk_dsql::{Client, Config};

/// Create a client. We will use this later for performing operations on the
 cluster.
async fn dsql_client(region: &'static str) -> Client {
```

```
    // Load default SDK configuration
    let sdk_defaults = load_defaults(BehaviorVersion::latest()).await;

    // You can set your own credentials by following this guide
    // https://docs.aws.amazon.com/sdk-for-rust/latest/dg/credproviders.html
    let credentials = sdk_defaults.credentials_provider().unwrap();

    let config = Config::builder()
        .behavior_version(BehaviorVersion::latest())
        .credentials_provider(credentials)
        .region(Region::new(region))
        .build();

    Client::from_conf(config)
}

/// Create a cluster without delete protection and a name
pub async fn delete_multi_region_clusters(
    region_1: &'static str,
    cluster_id_1: &'static str,
    region_2: &'static str,
    cluster_id_2: &'static str,
) {
    let client_1 = dsql_client(region_1).await;
    let client_2 = dsql_client(region_2).await;

    println!("Deleting cluster {cluster_id_1} in {region_1}");
    client_1
        .delete_cluster()
        .identifier(cluster_id_1)
        .send()
        .await
        .unwrap();

    // cluster_1 will stay in PENDING_DELETE state until cluster_2 is deleted
    println!("Deleting cluster {cluster_id_2} in {region_2}");
    client_2
        .delete_cluster()
        .identifier(cluster_id_2)
        .send()
        .await
        .unwrap();

    // Now that both clusters have been marked for deletion they will transition
```

```
    // to DELETING state and finalize deletion
    println!("Waiting for {cluster_id_1} to finish deletion");
    client_1
        .wait_until_cluster_not_exists()
        .identifier(cluster_id_1)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap();

    println!("Waiting for {cluster_id_2} to finish deletion");
    client_2
        .wait_until_cluster_not_exists()
        .identifier(cluster_id_2)
        .wait(std::time::Duration::from_secs(300)) // Wait up to 5 minutes
        .await
        .unwrap();
}

#[tokio::main(flavor = "current_thread")]
pub async fn main() -> anyhow::Result<()> {
    let region_1 = "us-east-1";
    let cluster_id_1 = "<cluster 1 to be deleted>";
    let region_2 = "us-east-2";
    let cluster_id_2 = "<cluster 2 to be deleted>";

    delete_multi_region_clusters(region_1, cluster_id_1, region_2,
 cluster_id_2).await;
    println!("Deleted {cluster_id_1} in {region_1} and {cluster_id_2} in
 {region_2}");

    Ok(())
}
```

Ruby

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
equire "aws-sdk-dsql"

def delete_cluster(region, identifier)
  client = Aws::DSQL::Client.new(region: region)
  cluster = client.delete_cluster(identifier: identifier)
  puts "Initiated delete of #{cluster.arn}"
```

```
  # The DSQL SDK offers built-in waiters to poll for deletion.
  puts "Waiting for cluster to finish deletion"
  client.wait_until(:cluster_not_exists, identifier: cluster.identifier) do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
rescue Aws::Errors::ServiceError => e
  abort "Unable to delete cluster #{identifier} in #{region}: #{e.message}"
end

def main
  region = "us-east-1"
  cluster_id = "<your cluster id>"
  delete_cluster(region, cluster_id)
  puts "Deleted #{cluster_id}"
end

main if $PROGRAM_NAME == __FILE__
```

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
require "aws-sdk-dsql"

def delete_multi_region_clusters(region_1, cluster_id_1, region_2, cluster_id_2)
  client_1 = Aws::DSQL::Client.new(region: region_1)
  client_2 = Aws::DSQL::Client.new(region: region_2)

  puts "Deleting cluster #{cluster_id_1} in #{region_1}"
  client_1.delete_cluster(identifier: cluster_id_1)

  # cluster_1 will stay in PENDING_DELETE state until cluster_2 is deleted
  puts "Deleting #{cluster_id_2} in #{region_2}"
  client_2.delete_cluster(identifier: cluster_id_2)

  # Now that both clusters have been marked for deletion they will transition
  # to DELETING state and finalize deletion
  puts "Waiting for #{cluster_id_1} to finish deletion"
  client_1.wait_until(:cluster_not_exists, identifier: cluster_id_1) do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
```

```
  end

  puts "Waiting for #{cluster_id_2} to finish deletion"
  client_2.wait_until(:cluster_not_exists, identifier: cluster_id_2) do |w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
rescue Aws::Errors::ServiceError => e
  abort "Failed to delete multi-region clusters: #{e.message}"
end

def main
  region_1 = "us-east-1"
  cluster_id_1 = "<your cluster id 1>"
  region_2 = "us-east-2"
  cluster_id_2 = "<your cluster id 2>"

  delete_multi_region_clusters(region_1, cluster_id_1, region_2, cluster_id_2)
  puts "Deleted #{cluster_id_1} in #{region_1} and #{cluster_id_2} in #{region_2}"
end

main if $PROGRAM_NAME == __FILE__
```

.NET

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;

namespace DSQLExamples.examples
{
    public class DeleteSingleRegionCluster
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
```

```
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Delete a DSQL cluster.
        /// </summary>
        public static async Task Delete(RegionEndpoint region, string identifier)
        {
            using (var client = await CreateDSQLClient(region))
            {
                var deleteRequest = new DeleteClusterRequest
                {
                    Identifier = identifier
                };

                var deleteResponse = await client.DeleteClusterAsync(deleteRequest);
                Console.WriteLine($"Initiated deletion of {deleteResponse.Arn}");
            }
        }

        private static async Task Main()
        {
            var region = RegionEndpoint.USEast1;
            var clusterId = "<cluster to be deleted>";

            await Delete(region, clusterId);
        }
    }
}
```

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
using System;
```

```
using System.Threading.Tasks;
using Amazon;
using Amazon.DSQL;
using Amazon.DSQL.Model;
using Amazon.Runtime.Credentials;
using Amazon.Runtime.Endpoints;

namespace DSQLExamples.examples
{
    public class DeleteMultiRegionClusters
    {
        /// <summary>
        /// Create a client. We will use this later for performing operations on the
 cluster.
        /// </summary>
        private static async Task<AmazonDSQLClient> CreateDSQLClient(RegionEndpoint
 region)
        {
            var awsCredentials = await
 DefaultAWSCredentialsIdentityResolver.GetCredentialsAsync();
            var clientConfig = new AmazonDSQLConfig
            {
                RegionEndpoint = region,
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Delete multi-region clusters.
        /// </summary>
        public static async Task Delete(
            RegionEndpoint region1,
            string clusterId1,
            RegionEndpoint region2,
            string clusterId2)
        {
            using (var client1 = await CreateDSQLClient(region1))
            using (var client2 = await CreateDSQLClient(region2))
            {
                var deleteRequest1 = new DeleteClusterRequest
                {
                    Identifier = clusterId1
                };
```

```
                var deleteResponse1 = await
 client1.DeleteClusterAsync(deleteRequest1);
                Console.WriteLine($"Initiated deletion of {deleteResponse1.Arn}");

                // cluster 1 will stay in PENDING_DELETE state until cluster 2 is
 deleted
                var deleteRequest2 = new DeleteClusterRequest
                {
                    Identifier = clusterId2
                };

                var deleteResponse2 = await
 client2.DeleteClusterAsync(deleteRequest2);
                Console.WriteLine($"Initiated deletion of {deleteResponse2.Arn}");
            }
        }

        private static async Task Main()
        {
            var region1 = RegionEndpoint.USEast1;
            var cluster1 = "<cluster 1 to be deleted>";
            var region2 = RegionEndpoint.USEast2;
            var cluster2 = "<cluster 2 to be deleted>";

            await Delete(region1, cluster1, region2, cluster2);
        }
    }
}
```

Golang

Untuk menghapus cluster dalam satu Wilayah AWS, gunakan contoh berikut.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/dsql"
)
```

```
func DeleteSingleRegion(ctx context.Context, identifier, region string) error {

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  log.Fatalf("Failed to load AWS configuration: %v", err)
 }

 // Initialize the DSQL client
 client := dsql.NewFromConfig(cfg)

 // Create delete cluster input
 deleteInput := &dsql.DeleteClusterInput{
  Identifier: &identifier,
 }

 // Delete the cluster
 result, err := client.DeleteCluster(ctx, deleteInput)
 if err != nil {
  return fmt.Errorf("failed to delete cluster: %w", err)
 }

 fmt.Printf("Initiated deletion of cluster: %s\n", *result.Arn)

 // Create waiter to check cluster deletion
 waiter := dsql.NewClusterNotExistsWaiter(client, func(options
 *dsql.ClusterNotExistsWaiterOptions) {
  options.MinDelay = 10 * time.Second
  options.MaxDelay = 30 * time.Second
  options.LogWaitAttempts = true
 })

 // Create the input for checking cluster status
 getInput := &dsql.GetClusterInput{
  Identifier: &identifier,
 }

 // Wait for the cluster to be deleted
 fmt.Printf("Waiting for cluster %s to be deleted...\n", identifier)
 err = waiter.Wait(ctx, getInput, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for cluster to be deleted: %w", err)
 }
```

```
 fmt.Printf("Cluster %s has been successfully deleted\n", identifier)
 return nil
}

func DeleteCluster(ctx context.Context) {
}

// Example usage in main function
func main() {
 // Your existing setup code for client configuration...

 ctx, cancel := context.WithTimeout(context.Background(), 6*time.Minute)
 defer cancel()

 // Example cluster identifier
 // Need to make sure that cluster does not have delete protection enabled
 identifier := "<CLUSTER_ID>"
 region := "us-east-1"

 err := DeleteSingleRegion(ctx, identifier, region)
 if err != nil {
  log.Fatalf("Failed to delete cluster: %v", err)
 }

}
```

Untuk menghapus cluster Multi-region, gunakan contoh berikut. Menghapus klaster Multi-wilayah mungkin membutuhkan waktu.

```
package main

import (
 "context"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/dsql"
)
```

```
func DeleteMultiRegionClusters(ctx context.Context, region1, clusterId1, region2,
 clusterId2 string) error {
 // Load the AWS configuration for region 1
 cfg1, err := config.LoadDefaultConfig(ctx, config.WithRegion(region1))
 if err != nil {
  return fmt.Errorf("unable to load SDK config for region %s: %w", region1, err)
 }

 // Load the AWS configuration for region 2
 cfg2, err := config.LoadDefaultConfig(ctx, config.WithRegion(region2))
 if err != nil {
  return fmt.Errorf("unable to load SDK config for region %s: %w", region2, err)
 }

 // Create DSQL clients for both regions
 client1 := dsql.NewFromConfig(cfg1)
 client2 := dsql.NewFromConfig(cfg2)

 // Delete cluster in region 1
 fmt.Printf("Deleting cluster %s in %s\n", clusterId1, region1)
 _, err = client1.DeleteCluster(ctx, &dsql.DeleteClusterInput{
  Identifier: aws.String(clusterId1),
 })
 if err != nil {
  return fmt.Errorf("failed to delete cluster in region %s: %w", region1, err)
 }

 // Delete cluster in region 2
 fmt.Printf("Deleting cluster %s in %s\n", clusterId2, region2)
 _, err = client2.DeleteCluster(ctx, &dsql.DeleteClusterInput{
  Identifier: aws.String(clusterId2),
 })
 if err != nil {
  return fmt.Errorf("failed to delete cluster in region %s: %w", region2, err)
 }

 // Create waiters for both regions
 waiter1 := dsql.NewClusterNotExistsWaiter(client1, func(options
 *dsql.ClusterNotExistsWaiterOptions) {
  options.MinDelay = 10 * time.Second
  options.MaxDelay = 30 * time.Second
  options.LogWaitAttempts = true
 })
```

```
 waiter2 := dsql.NewClusterNotExistsWaiter(client2, func(options
 *dsql.ClusterNotExistsWaiterOptions) {
  options.MinDelay = 10 * time.Second
  options.MaxDelay = 30 * time.Second
  options.LogWaitAttempts = true
 })

 // Wait for cluster in region 1 to be deleted
 fmt.Printf("Waiting for cluster %s to finish deletion\n", clusterId1)
 err = waiter1.Wait(ctx, &dsql.GetClusterInput{
  Identifier: aws.String(clusterId1),
 }, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for cluster deletion in region %s: %w", region1,
 err)
 }

 // Wait for cluster in region 2 to be deleted
 fmt.Printf("Waiting for cluster %s to finish deletion\n", clusterId2)
 err = waiter2.Wait(ctx, &dsql.GetClusterInput{
  Identifier: aws.String(clusterId2),
 }, 5*time.Minute)
 if err != nil {
  return fmt.Errorf("error waiting for cluster deletion in region %s: %w", region2,
 err)
 }

 fmt.Printf("Successfully deleted clusters %s in %s and %s in %s\n",
  clusterId1, region1, clusterId2, region2)
 return nil
}

// Example usage in main function
func main() {
 ctx, cancel := context.WithTimeout(context.Background(), 10*time.Minute)
 defer cancel()

 err := DeleteMultiRegionClusters(
  ctx,
  "us-east-1",      // region1
  "<CLUSTER_ID_1>", // clusterId1
  "us-east-2",      // region2
  "<CLUSTER_ID_2>", // clusterId2
 )
```

```
 if err != nil {
  log.Fatalf("Failed to delete multi-region clusters: %v", err)
 }
}
```

Tutorial

Tutorial dan kode contoh berikut GitHub membantu Anda melakukan tugas-tugas umum di Aurora DSQL.

- Menggunakan Benchbase dengan Aurora DSQL — cabang dari utilitas benchmarking open-source Benchbase yang diverifikasi untuk bekerja dengan Aurora DSQL.
- Aurora DSQL loader — skrip Python open-source ini memudahkan Anda memuat data ke Aurora DSQL untuk kasus penggunaan Anda, seperti mengisi tabel untuk menguji atau mentransfer data ke Aurora DSQL.
- Sampel Aurora DSQL — `aws-samples/aurora-dsql-samples` repositori GitHub berisi contoh kode tentang cara menghubungkan dan menggunakan Aurora DSQL dalam berbagai bahasa pemrograman menggunakan, pemetaan relasional objek (), dan kerangka kerja web. AWS SDKs ORMs Contoh menunjukkan bagaimana melakukan tugas-tugas umum, seperti menginstal klien, menangani otentikasi, dan melakukan operasi CRUD.

Menggunakan AWS Lambda dengan Amazon Aurora DSQL

Tutorial berikut menjelaskan cara menggunakan Lambda dengan Aurora DSQL

Prasyarat

- Otorisasi untuk membuat fungsi Lambda. Untuk informasi selengkapnya, lihat Memulai dengan Lambda.
- Otorisasi untuk membuat atau memodifikasi kebijakan IAM yang dibuat oleh Lambda. Anda perlu izin `iam:CreatePolicy` dan`iam:AttachRolePolicy`. Untuk informasi selengkapnya, lihat Tindakan, sumber daya, dan kunci kondisi untuk IAM.
- Anda harus menginstal npm v8.5.3 atau lebih tinggi.
- Anda harus menginstal zip v3.0 atau lebih tinggi.

Buat fungsi baru di AWS Lambda.

1. Masuk ke AWS Management Console dan buka AWS Lambda konsol di https://console.aws.amazon.com/lambda/.
2. Pilih Buat fungsi.
3. Berikan nama, seperti`dsql-sample`.
4. Jangan mengedit pengaturan default untuk memastikan bahwa Lambda membuat peran baru dengan izin Lambda dasar.
5. Pilih Buat fungsi.

Otorisasi peran eksekusi Lambda Anda untuk terhubung ke klaster Anda

1. Di fungsi Lambda Anda, pilih Konfigurasi > Izin.
2. Pilih nama peran untuk membuka peran eksekusi di konsol IAM.
3. Pilih Tambahkan Izin > Buat kebijakan sebaris, dan gunakan editor JSON.
4. Di Action paste dalam tindakan berikut untuk mengotorisasi identitas IAM Anda untuk terhubung menggunakan peran database admin.

```
"Action": ["dsql:DbConnectAdmin"],
```

> **Note**
>
> Kami menggunakan peran admin untuk meminimalkan langkah-langkah prasyarat untuk memulai. Anda tidak boleh menggunakan peran database admin untuk aplikasi produksi Anda. Lihat Menggunakan peran database dan otentikasi IAM untuk mempelajari cara membuat peran basis data kustom dengan otorisasi yang memiliki izin paling sedikit ke database Anda.

5. Di Resource, tambahkan Amazon Resource Name (ARN) klaster Anda. Anda juga dapat menggunakan wildcard.

```
"Resource": ["*"]
```

6. Pilih Berikutnya.
7. Masukkan nama untuk kebijakan tersebut, seperti`dsql-sample-dbconnect`.
8. Pilih Buat kebijakan.

Buat paket untuk diunggah ke Lambda.

1. Buat folder bernamamy`function`.
2. Di folder, buat file baru bernama `package.json` dengan konten berikut.

```
{
  "dependencies": {
    "@aws-sdk/dsql-signer": "^3.705.0",
    "assert": "2.1.0",
    "pg": "^8.13.1"
  }
}
```

3. Di folder, buat file bernama `index.mjs` di direktori dengan konten berikut.

```
import { DsqlSigner } from "@aws-sdk/dsql-signer";
import pg from "pg";
import assert from "node:assert";
const { Client } = pg;


async function dsql_sample(clusterEndpoint, region) {
  let client;
  try {
    // The token expiration time is optional, and the default value 900 seconds
    const signer = new DsqlSigner({
      hostname: clusterEndpoint,
      region,
    });
    const token = await signer.getDbConnectAdminAuthToken();
    // <https://node-postgres.com/apis/client>
    // By default `rejectUnauthorized` is true in TLS options
    // <https://nodejs.org/api/tls.html#tls_tls_connect_options_callback>
    // The config does not offer any specific parameter to set sslmode to verify-
full
    // Settings are controlled either via connection string or by setting
    // rejectUnauthorized to false in ssl options
    client = new Client({
      host: clusterEndpoint,
      user: "admin",
      password: token,
      database: "postgres",
      port: 5432,
```

```
      // <https://node-postgres.com/announcements> for version 8.0
      ssl: true,
      rejectUnauthorized: false
    });

    // Connect
    await client.connect();

    // Create a new table
    await client.query(`CREATE TABLE IF NOT EXISTS owner (
      id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
      name VARCHAR(30) NOT NULL,
      city VARCHAR(80) NOT NULL,
      telephone VARCHAR(20)
    )`);

    // Insert some data
    await client.query("INSERT INTO owner(name, city, telephone) VALUES($1, $2,
 $3)",
      ["John Doe", "Anytown", "555-555-1900"]
    );

    // Check that data is inserted by reading it back
    const result = await client.query("SELECT id, city FROM owner where name='John
 Doe'");
    assert.deepEqual(result.rows[0].city, "Anytown")
    assert.notEqual(result.rows[0].id, null)

    await client.query("DELETE FROM owner where name='John Doe'");

  } catch (error) {
    console.error(error);
    throw new Error("Failed to connect to the database");
  } finally {
    client?.end();
  }
  Promise.resolve();
}


// https://docs.aws.amazon.com/lambda/latest/dg/nodejs-handler.html
export const handler = async (event) => {
  const endpoint = event.endpoint;
  const region = event.region;
```

```
  const responseCode = await dsql_sample(endpoint, region);

  const response = {
    statusCode: responseCode,
    endpoint: endpoint,
  };
  return response;
};
```

4. Gunakan perintah berikut untuk membuat paket.

```
npm install
zip -r pkg.zip .
```

Unggah paket kode dan uji fungsi Lambda Anda

1. Di tab Kode fungsi Lambda Anda, pilih Unggah dari> file.zip
2. Unggah yang `pkg.zip` Anda buat. Untuk informasi selengkapnya, lihat Menerapkan fungsi Lambda Node.js dengan arsip file.zip.
3. Di tab Test fungsi Lambda Anda, tempelkan payload JSON berikut, dan modifikasi untuk menggunakan ID cluster Anda.
4. Di tab Uji fungsi Lambda Anda, gunakan Event JSON berikut yang dimodifikasi untuk menentukan titik akhir klaster Anda.

```
{"endpoint": "replace_with_your_cluster_endpoint"}
```

5. Masukkan nama Acara, seperti`sql-sample-test`. Pilih Simpan.
6. Pilih Uji.
7. Pilih Detail untuk memperluas respons eksekusi dan output log.
8. Jika berhasil, respons eksekusi fungsi Lambda harus mengembalikan kode status 200.

```
{"statusCode": 200, "endpoint": "your_cluster_endpoint"}
```

Jika database mengembalikan kesalahan atau jika koneksi ke database gagal, respons eksekusi fungsi Lambda mengembalikan kode status 500.

```
{"statusCode": 500,"endpoint": "your_cluster_endpoint"}
```

Cadangkan dan pulihkan untuk Amazon Aurora DSQL

Amazon Aurora DSQL membantu Anda memenuhi kepatuhan peraturan dan persyaratan kelangsungan bisnis melalui integrasi dengan AWS Backup, layanan perlindungan data yang dikelola sepenuhnya yang memudahkan untuk memusatkan dan mengotomatiskan pencadangan di seluruh AWS layanan, di cloud, dan di tempat. Layanan ini merampingkan pembuatan cadangan, pengelolaan, dan pemulihan untuk klaster DSQL Aurora Wilayah Tunggal dan Multi-wilayah.

Fitur utama meliputi:

- Manajemen cadangan terpusat melalui AWS Management Console, SDK, atau AWS CLI
- Cadangan klaster penuh
- Jadwal pencadangan otomatis dan kebijakan retensi
- Kemampuan lintas wilayah dan lintas akun
- Konfigurasi WORM (tulis-sekali, baca-banyak) untuk semua cadangan yang Anda simpan

Untuk informasi selengkapnya tentang fitur AWS Backup Vault Lock dan daftar lengkap AWS Backup fitur yang tersedia untuk Aurora DSQL, lihat Manfaat kunci Vault AWS Backup dan ketersediaan fitur di Panduan Pengembang.AWS Backup

Memulai dengan AWS Backup

AWS Backup membuat salinan lengkap cluster Aurora DSQL Anda. Anda dapat mulai menggunakan AWS Backup untuk Aurora DSQL dengan mengikuti langkah-langkah dalam Memulai dengan: AWS Backup

1. Buat cadangan sesuai permintaan untuk perlindungan segera.
2. Buat rencana cadangan untuk pencadangan terjadwal otomatis.
3. Konfigurasikan periode retensi dan penyalinan lintas wilayah.
4. Siapkan pemantauan dan pemberitahuan untuk aktivitas pencadangan.

Memulihkan cadangan Anda

Saat Anda memulihkan cluster Aurora DSQL, AWS Backup selalu buat cluster baru untuk melestarikan data sumber Anda.

Memulihkan kluster wilayah tunggal

Untuk memulihkan cluster Aurora DSQL Single-region, gunakan console: /backup https://console.aws.amazon.comatau CLI untuk memilih titik pemulihan (backup) yang ingin Anda pulihkan. Konfigurasikan pengaturan untuk cluster baru yang akan dibuat dari cadangan Anda. Untuk petunjuk mendetail, lihat Mengembalikan klaster DSQL Aurora Wilayah Tunggal.

Memulihkan klaster Multi-wilayah

Memulihkan cluster Multi-region Aurora DSQL didukung melalui konsol: /backup dan file. https://console.aws.amazon.com AWS CLI Untuk petunjuk mendetail, lihat Mengembalikan klaster DSQL Aurora Multi-wilayah.

Untuk mengembalikan ke cluster Aurora DSQL Multi-wilayah, Anda dapat menggunakan cadangan yang diambil dalam satu. Wilayah AWS Namun, sebelum Anda memulai proses pemulihan, Anda harus memastikan ada salinan identik dari cadangan Anda di semua Wilayah AWS untuk klaster Multi-wilayah Anda. Jika Anda belum memiliki salinan tersebut, Anda harus terlebih dahulu menyalin cadangan ke yang lain Wilayah AWS yang mendukung klaster Multi-wilayah.

Kami merekomendasikan untuk membuat salinan cadangan sebagai kunci Wilayah AWS untuk mengaktifkan opsi pemulihan bencana yang kuat dan memenuhi persyaratan kepatuhan. Untuk melihat tersedia Wilayah AWS untuk Aurora DSQL, lihat. the section called “Wilayah AWS ketersediaan”

Untuk petunjuk terperinci tentang langkah-langkah ini, lihat dokumentasi pemulihan Amazon Aurora DSQL.

Pemantauan dan kepatuhan

AWS Backup memberikan visibilitas komprehensif ke dalam operasi pencadangan dan pemulihan dengan sumber daya berikut.

- Dasbor terpusat untuk melacak pencadangan dan pemulihan pekerjaan
- Integrasi dengan CloudWatch dan CloudTrail.
- AWS Backup Audit Manager untuk pelaporan dan audit kepatuhan.

Lihat Pencatatan operasi Aurora DSQL menggunakan AWS CloudTrail untuk mempelajari lebih lanjut tentang mencatat catatan tindakan yang diambil oleh pengguna, peran, atau Layanan AWS saat menggunakan Aurora DSQL.

Sumber daya tambahan

Untuk mempelajari lebih lanjut tentang AWS Backup fitur dan dan menggunakannya bersama-sama dengan Aurora DSQL, lihat sumber daya berikut:

- Kebijakan terkelola untuk AWS Backup
- Pemulihan DSQL Amazon Aurora
- Layanan yang didukung oleh Wilayah AWS
- Enkripsi untuk backup di AWS Backup

Dengan menggunakan AWS Backup Aurora DSQL, Anda menerapkan strategi pencadangan yang kuat, sesuai, dan otomatis yang melindungi sumber daya basis data penting Anda sambil meminimalkan overhead administratif. Baik Anda mengelola satu cluster atau penyebaran Multi-wilayah yang kompleks, AWS Backup menyediakan alat yang Anda butuhkan untuk memastikan data Anda tetap aman dan dapat dipulihkan.

Pemantauan dan pencatatan untuk Aurora DSQL

Pemantauan dan pencatatan adalah bagian penting untuk menjaga keandalan, ketersediaan, dan kinerja sumber daya Amazon Aurora DSQL Anda. Anda harus memantau dan mengumpulkan data logging dari semua bagian sumber daya Aurora DSQL Anda sehingga Anda dapat dengan mudah men-debug kegagalan multi-titik.

- Amazon CloudWatch memantau AWS sumber daya Anda dan aplikasi yang Anda jalankan AWS secara real time. Anda dapat mengumpulkan dan melacak metrik, membuat dasbor yang disesuaikan, dan mengatur alarm yang memberi tahu Anda atau mengambil tindakan saat metrik tertentu mencapai ambang batas yang ditentukan. Misalnya, Anda dapat CloudWatch melacak penggunaan CPU atau metrik lain dari EC2 instans Amazon Anda dan secara otomatis meluncurkan instans baru bila diperlukan. Untuk informasi selengkapnya, lihat Panduan CloudWatch Pengguna Amazon.
- AWS CloudTrailmenangkap panggilan API dan peristiwa terkait yang dibuat oleh atau atas nama Anda Akun AWS dan 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, lihat Panduan Pengguna AWS CloudTrail.

Melihat status cluster Aurora DSQL

Status cluster Aurora DSQL memberikan informasi penting tentang kesehatan dan konektivitas cluster. Anda dapat melihat status cluster dan instance cluster dengan menggunakan AWS Management Console, AWS CLI, atau Aurora DSQL API.

Status dan definisi cluster Aurora DSQL

Tabel berikut menjelaskan setiap status yang mungkin untuk cluster Aurora DSQL dan apa arti setiap status.

Status	Deskripsi
Creating	Aurora DSQL sedang mencoba untuk membuat atau mengkonfigurasi sumber daya untuk cluster. Setiap upaya koneksi akan gagal saat cluster dalam keadaan ini.

Status	Deskripsi
Aktif	Cluster ini operasional dan siap digunakan.
Menganggur	Sebuah cluster menjadi idle ketika menganggur cukup lama bagi Aurora DSQL untuk merebut kembali sumber daya yang dikonfigurasi untuknya. Saat Anda terhubung ke cluster idle, Aurora DSQL mentransisikan cluster kembali ke status Aktif.
Tidak aktif	Sebuah cluster menjadi tidak aktif ketika tidak ada aktivitas di cluster untuk waktu yang lama. Saat Anda mencoba terhubung ke cluster yang tidak aktif, Aurora DSQL secara otomatis mentransisikan cluster kembali ke status Aktif.
Memperbarui	Transisi klaster ke status Memperbarui saat Anda membuat perubahan pada konfigurasi klaster.
Deleting	Transisi klaster ke status Menghapus saat Anda mengirimkan permintaan untuk menghapusnya.
Dihapus	Cluster telah berhasil dihapus.
Failed	Aurora DSQL tidak dapat membuat cluster karena mengalami kesalahan.
Pengaturan Tertunda	Hanya untuk klaster Multi-wilayah. Kluster Multi-wilayah memasuki status Pengaturan Tertunda saat Anda membuat klaster Multi-wilayah di Wilayah pertama Anda dengan Wilayah saksi. Pembuatan cluster berhenti sampai Anda membuat cluster lain di Region sekunder dan mengintip kedua cluster bersama-sama.
Hapus Tertunda	Hanya untuk klaster Multi-wilayah. Kluster Multi-wilayah memasuki status Penghapusan Tertunda saat Anda menghapus klaster darinya. Cluster bergerak ke status Menghapus setelah Anda menghapus klaster rekan terakhir.

Melihat status cluster Aurora DSQL Anda

Untuk melihat status klaster Anda, gunakan AWS Management Console AWS CLI, atau Aurora DSQL API.

Konsol

Ikuti langkah-langkah berikut untuk melihat status klaster di AWS Management Console:

Untuk melihat status klaster di konsol

1. Buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql
2. Pilih Cluster di panel navigasi.
3. Lihat status untuk setiap cluster di dasbor.

AWS CLI

Gunakan AWS CLI perintah berikut untuk memeriksa status cluster tunggal.

```
aws dsql get-cluster --identifier cluster-id --query status --output text
```

Jalankan perintah berikut untuk daftar status semua cluster.

```
for id in $(aws dsql list-clusters --query 'clusters[*].identifier' --output text); do

  cluster_status=$(aws dsql get-cluster --identifier "$id" --query 'status' --output
 text)
  echo "$id    $cluster_status"
done
```

Output sampel ini menunjukkan dua cluster aktif dan satu cluster dalam proses dihapus.

```
aaabbb2bkx555xa7p42qd5cdef    ACTIVE
abcde123efghi77t35abcdefgh    ACTIVE
12abc6lqasc5bbbbbbbbbbbbbb    DELETING
```

Memantau Aurora DSQL dengan Amazon CloudWatch

Pantau penggunaan Aurora DSQL CloudWatch, yang mengumpulkan data mentah dan memprosesnya menjadi metrik yang dapat dibaca, mendekati waktu nyata. CloudWatch menyimpan statistik ini selama 15 bulan, membantu Anda mendapatkan perspektif yang lebih baik tentang aplikasi web atau kinerja layanan Anda. Atur alarm untuk mengawasi ambang batas tertentu dan

mengirim pemberitahuan atau mengambil tindakan saat terpenuhi. Tinjau metrik Penggunaan dan Pengamatan berikut yang tersedia untuk Aurora DSQL.

Untuk informasi selengkapnya, lihat Panduan CloudWatch Pengguna Amazon.

Observabilitas dan kinerja

Tabel ini menguraikan metrik observabilitas untuk Aurora DSQL. Ini mencakup metrik untuk melacak transaksi hanya-baca dan total untuk memberikan karakterisasi beban kerja secara keseluruhan. Metrik yang dapat ditindaklanjuti seperti batas waktu kueri dan tingkat konflik OCC disertakan untuk membantu mengidentifikasi masalah kinerja dan konflik konkurensi. Metrik terkait sesi, baik aktif maupun total, menawarkan wawasan tentang beban saat ini pada sistem.

CloudWatch Nama Metrik	Metrik	Unit	Deskripsi
ReadOnlyTransactions	Transaksi hanya-baca	none	Jumlah transaksi read-only
TotalTransactions	Total transaksi	none	Jumlah total transaksi yang dijalankan pada sistem, termasuk transaksi read-only.
QueryTimeouts	Waktu kueri	none	Jumlah kueri yang telah habis waktunya karena mencapai waktu transaksi maksimum
OccConflicts	Konflik OCC	none	Jumlah transaksi dibatalkan karena tingkat kunci OCC
CommitLatency	Komit Latensi	milidetik	Waktu yang dihabiskan oleh fase komit eksekusi kueri (P50)

CloudWatch Nama Metrik	Metrik	Unit	Deskripsi
BytesWritten	Bytes Ditulis	byte	Byte ditulis ke penyimpanan
BytesRead	Bytes Baca	byte	Byte dibaca dari penyimpanan
ComputeTime	Waktu komputasi QP	milidetik	Waktu jam dinding QP
ClusterStorageSize	Ukuran Penyimpanan Cluster	byte	Ukuran cluster

Metrik penggunaan

Aurora DSQL mengukur semua aktivitas berbasis permintaan, seperti pemrosesan kueri, pembacaan, dan penulisan, menggunakan satu unit penagihan dinormalisasi yang disebut Distributed Processing Unit (DPU).

CloudWatch Nama Metrik	Metrik	Dimensi: ResourceId	Unit	Deskripsi
DitulisPU	Tulis Unit	<cluster-id>	DPU	Perkiraan komponen penggunaan aktif tulis dari penggunaan DPU cluster Aurora DSQL Anda.
MultiRegionWriteDPU	Unit Tulis Multi-Wilayah	<cluster-id>	DPU	Berlaku untuk klaster Multi-Wilayah: Memperkirakan komponen penggunaan

CloudWatch Nama Metrik	Metrik	Dimensi: ResourceId	Unit	Deskripsi
				aktif penulisan Multi-wilayah dari penggunaan DPU cluster Aurora DSQL Anda.
bacaDPU	Baca Unit	<cluster-id>	DPU	Perkiraan komponen penggunaan aktif baca dari penggunaan DPU cluster Aurora DSQL Anda.
ComputedPU	Unit Komputasi	<cluster-id>	DPU	Perkiraan komponen penggunaan aktif komputasi dari penggunaan DPU cluster Aurora DSQL Anda.
TotalDPU	Jumlah Unit	<cluster-id>	DPU	Perkiraan total komponen penggunaan aktif dari penggunaan DPU cluster Aurora DSQL Anda.

Pencatatan operasi Aurora DSQL menggunakan AWS CloudTrail

Amazon Aurora DSQL terintegrasi dengan AWS CloudTrail, layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau. Layanan AWS Ada dua jenis peristiwa di CloudTrail: peristiwa manajemen dan peristiwa data. Peristiwa manajemen dipancarkan untuk mengaudit perubahan konfigurasi AWS sumber daya. Peristiwa data menangkap penggunaan AWS sumber daya biasanya di bidang data layanan.

CloudTrail menangkap semua panggilan API untuk Aurora DSQL sebagai peristiwa. Aurora DSQL merekam aktivitas konsol sebagai peristiwa manajemen. Ini juga menangkap upaya koneksi yang diautentikasi ke cluster sebagai peristiwa data.

Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat untuk Aurora DSQL, alamat IP dari mana permintaan dibuat, kapan dibuat, identitas pengguna yang membuat permintaan, dan detail tambahan.

CloudTrail diaktifkan secara default di Akun AWS saat Anda membuat akun dan Anda memiliki akses ke riwayat CloudTrail Acara. Riwayat CloudTrail Acara menyediakan catatan yang dapat dilihat, dapat dicari, dapat diunduh, dan tidak dapat diubah dari 90 hari terakhir dari peristiwa manajemen yang direkam dalam file. Wilayah AWS Untuk informasi selengkapnya, lihat Bekerja dengan riwayat CloudTrail Acara di Panduan AWS CloudTrail Pengguna. Tidak ada CloudTrail biaya untuk merekam riwayat Acara.

Untuk membuat catatan peristiwa yang sedang berlangsung di AWS akun Anda, termasuk peristiwa untuk Aurora DSQL, buat jejak atau penyimpanan data acara AWS CloudTrail Lake (solusi penyimpanan dan analisis terpusat untuk acara). AWS CloudTrail Untuk informasi selengkapnya tentang membuat jejak, lihat Bekerja dengan CloudTrail jalur. Untuk mempelajari cara menyiapkan dan mengelola penyimpanan data acara, lihat Penyimpanan data acara CloudTrail Danau.

Acara manajemen Aurora DSQL di CloudTrail

CloudTrail Acara manajemen memberikan informasi tentang operasi manajemen yang dilakukan pada sumber daya di AWS akun Anda. Ini juga dikenal sebagai operasi pesawat kontrol. Secara default, CloudTrail menangkap peristiwa manajemen dalam riwayat Acara.

Amazon Aurora DSQL mencatat semua operasi pesawat kontrol Aurora DSQL sebagai peristiwa manajemen. Untuk daftar operasi bidang kontrol Amazon Aurora DSQL yang dicatat oleh Aurora DSQL CloudTrail, lihat referensi Aurora DSQL API.

Kontrol log pesawat

Amazon Aurora DSQL mencatat operasi bidang kontrol Aurora DSQL berikut sebagai peristiwa manajemen. CloudTrail

- CreateCluster
- DeleteCluster
- GetCluster
- GetVpcEndpointServiceName
- ListClusters
- ListTagsForResource
- TagResource
- UntagResource
- UpdateCluster

Cadangkan dan pulihkan log

Amazon Aurora DSQL mencatat operasi pencadangan dan pemulihan Aurora DSQL berikut sebagai peristiwa manajemen. CloudTrail

- `StartBackupJob`
- `StopBackupJob`
- `GetBackupJob`
- `StartRestoreJob`
- `StopRestoreJob`
- `GetRestoreJob`

Untuk informasi lebih lanjut tentang melindungi cluster Aurora DSQL Anda menggunakan, lihat. AWS BackupCadangkan dan pulihkan untuk Amazon Aurora DSQL

Log AWS KMS

Amazon Aurora DSQL mencatat AWS KMS operasi berikut sebagai peristiwa manajemen. CloudTrail

- `GenerateDataKey`
- `Decrypt`

Untuk mempelajari lebih lanjut tentang cara CloudTrail log melacak permintaan yang dikirim Aurora DSQL AWS KMS atas nama Anda, lihat. Memantau interaksi Aurora DSQL dengan AWS KMS

Peristiwa data Aurora DSQL di CloudTrail

CloudTrail Peristiwa data biasanya memberikan informasi tentang operasi sumber daya yang dilakukan pada atau di sumber daya. Ini juga digunakan untuk menangkap operasi pesawat data layanan. Peristiwa data seringkali merupakan aktivitas volume tinggi. Secara default, CloudTrail tidak mencatat peristiwa data. Riwayat CloudTrail peristiwa tidak merekam peristiwa data.

Untuk informasi selengkapnya tentang cara mencatat peristiwa data, lihat Mencatat peristiwa data dengan AWS Management Console dan Logging peristiwa data dengan AWS Command Line Interface di Panduan AWS CloudTrail Pengguna.

Biaya tambahan berlaku untuk peristiwa data. Untuk informasi selengkapnya tentang CloudTrail harga, lihat AWS CloudTrail Harga.

Untuk Aurora DSQL, CloudTrail menangkap setiap upaya koneksi yang dilakukan ke cluster Aurora DSQL sebagai peristiwa data. Tabel berikut mencantumkan jenis sumber daya Aurora DSQL yang dapat Anda log peristiwa data. Kolom Jenis sumber daya (konsol) menunjukkan nilai yang akan dipilih dari daftar Jenis sumber daya di CloudTrail konsol. Kolom nilai resources.type menunjukkan **`resources.type`** nilai, yang akan Anda tentukan saat mengonfigurasi penyeleksi acara lanjutan menggunakan or. AWS CLI CloudTrail APIs CloudTrailKolom Data yang APIs dicatat ke menampilkan panggilan API yang dicatat CloudTrail untuk jenis sumber daya.

Jenis sumber daya (konsol)	nilai resources.type	Data APIs masuk ke CloudTrail
Amazon Aurora DSQL	`AWS::DSQL::Cluster`	• `DbConnect` • `DbConnectAdmin`

Anda dapat mengonfigurasi pemilih acara lanjutan untuk memfilter pada `eventName` dan `resources.`ARN bidang untuk mencatat peristiwa yang hanya difilter. Untuk informasi selengkapnya tentang bidang ini, lihat AdvancedFieldSelectordi Referensi AWS CloudTrail API.

Contoh berikut menunjukkan bagaimana menggunakan untuk mengkonfigurasi AWS CLI untuk menerima peristiwa data `dsql-data-events-trail` untuk Aurora DSQL.

```
aws cloudtrail put-event-selectors \
--region us-east-1 \
--trail-name dsql-data-events-trail \
--advanced-event-selectors '[{
"Name": "Log DSQL Data Events",
    "FieldSelectors": [
       { "Field": "eventCategory", "Equals": ["Data"] },
       { "Field": "resources.type", "Equals": ["AWS::DSQL::Cluster"] } ]}]'
```

Keamanan di Amazon Aurora DSQL

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai AWS pelanggan, Anda mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Model tanggung jawab bersama menjelaskan hal ini sebagai keamanan dari cloud dan keamanan dalam cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan AWS layanan di AWS Cloud. AWS juga memberi Anda layanan yang dapat Anda gunakan dengan aman. Auditor pihak ketiga secara teratur menguji dan memverifikasi efektivitas keamanan kami sebagai bagian dari Program AWS Kepatuhan Program AWS Kepatuhan . Untuk mempelajari tentang program kepatuhan yang berlaku untuk Amazon Aurora DSQL, lihat AWS Layanan dalam Lingkup oleh Program Kepatuhan dalam Lingkup oleh Program Kepatuhan.
- Keamanan di cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan. Anda juga bertanggung jawab atas faktor lain, yang mencakup sensitivitas data Anda, persyaratan perusahaan Anda, serta undang-undang dan peraturan yang berlaku.

Dokumentasi ini membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan Aurora DSQL. Topik berikut menunjukkan cara mengkonfigurasi Aurora DSQL untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga mempelajari cara menggunakan AWS layanan lain yang membantu Anda memantau dan mengamankan sumber daya Aurora DSQL Anda.

Topik

- Keamanan Infrastruktur di Amazon Aurora DSQL
- Analisis konfigurasi dan kerentanan di Amazon Aurora DSQL
- Pencegahan "confused deputy" lintas layanan
- Praktik terbaik keamanan untuk Aurora DSQL

AWS kebijakan terkelola untuk Amazon Aurora DSQL

Kebijakan AWS terkelola adalah kebijakan mandiri yang dibuat dan dikelola oleh AWS. AWS Kebijakan terkelola dirancang untuk memberikan izin bagi banyak kasus penggunaan umum sehingga Anda dapat mulai menetapkan izin kepada pengguna, grup, dan peran.

Perlu diingat bahwa kebijakan AWS terkelola mungkin tidak memberikan izin hak istimewa paling sedikit untuk kasus penggunaan spesifik Anda karena tersedia untuk digunakan semua pelanggan. AWS Kami menyarankan Anda untuk mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola pelanggan yang khusus untuk kasus penggunaan Anda.

Anda tidak dapat mengubah izin yang ditentukan dalam kebijakan AWS terkelola. Jika AWS memperbarui izin yang ditentukan dalam kebijakan AWS terkelola, pemutakhiran akan memengaruhi semua identitas utama (pengguna, grup, dan peran) yang dilampirkan kebijakan tersebut. AWS kemungkinan besar akan memperbarui kebijakan AWS terkelola saat baru Layanan AWS diluncurkan atau operasi API baru tersedia untuk layanan yang ada.

Untuk informasi selengkapnya, lihat Kebijakan terkelola AWS dalam Panduan Pengguna IAM.

AWS kebijakan terkelola: AmazonAurora DSQLFull Akses

Anda dapat melampirkan `AmazonAuroraDSQLFullAccess` ke pengguna, grup, dan peran Anda.

Kebijakan ini memberikan izin yang memungkinkan akses administratif penuh ke Aurora DSQL. Prinsipal dengan izin ini dapat membuat, menghapus, dan memperbarui cluster Aurora DSQL, termasuk klaster Multi-wilayah. Mereka dapat menambah dan menghapus tag dari cluster. Mereka dapat membuat daftar cluster dan melihat informasi tentang cluster individu. Mereka dapat melihat tag yang dilampirkan ke cluster Aurora DSQL. Mereka dapat terhubung ke database sebagai

pengguna mana pun, termasuk admin. Mereka dapat melakukan operasi pencadangan dan pemulihan untuk cluster Aurora DSQL, termasuk memulai, menghentikan, dan memantau pekerjaan pencadangan dan pemulihan. Kebijakan ini mencakup AWS KMS izin yang memungkinkan operasi pada kunci yang dikelola pelanggan yang digunakan untuk enkripsi klaster. Mereka dapat melihat metrik apa pun dari CloudWatch akun Anda. Mereka juga memiliki izin untuk membuat peran terkait layanan untuk `dsql.amazonaws.com` layanan, yang diperlukan untuk membuat cluster.

Detail izin

Kebijakan ini mencakup izin berikut.

- `dsql`—memberikan kepala sekolah akses penuh ke Aurora DSQL.
- `cloudwatch`—memberikan izin untuk mempublikasikan titik data metrik ke Amazon. CloudWatch
- `iam`—memberikan izin untuk membuat peran terkait layanan.
- `backup and restore`—memberikan izin untuk memulai, menghentikan, dan memantau pencadangan dan pemulihan pekerjaan untuk klaster Aurora DSQL.
- `kms`—memberikan izin yang diperlukan untuk memvalidasi akses ke kunci yang dikelola pelanggan yang digunakan untuk enkripsi klaster Aurora DSQL saat membuat, memperbarui, atau menghubungkan ke cluster.

Anda dapat menemukan `AmazonAuroraDSQLFullAccess` kebijakan di konsol IAM dan AmazonAuroraDSQLFullAccess di Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AmazonAurora DSQLRead OnlyAccess

Anda dapat melampirkan `AmazonAuroraDSQLReadOnlyAccess` ke pengguna, grup, dan peran Anda.

Memungkinkan akses baca ke Aurora DSQL. Prinsipal dengan izin ini dapat mencantumkan kluster dan melihat informasi tentang kluster individu. Mereka dapat melihat tag yang dilampirkan ke cluster Aurora DSQL. Mereka dapat mengambil dan melihat metrik apa pun dari akun CloudWatch Anda.

Detail izin

Kebijakan ini mencakup izin berikut.

- `dsql`— memberikan izin baca saja untuk semua sumber daya di Aurora DSQL.
- `cloudwatch`— memberikan izin untuk mengambil jumlah batch data CloudWatch metrik dan melakukan matematika metrik pada data yang diambil

Anda dapat menemukan `AmazonAuroraDSQLReadOnlyAccess` kebijakan di konsol IAM dan AmazonAuroraDSQLReadOnlyAccessdi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: AmazonAurora DSQLConsole FullAccess

Anda dapat melampirkan `AmazonAuroraDSQLConsoleFullAccess` ke pengguna, grup, dan peran Anda.

Memungkinkan akses administratif penuh ke Amazon Aurora DSQL melalui file. AWS Management Console Prinsipal dengan izin ini dapat membuat, menghapus, dan memperbarui cluster Aurora DSQL, termasuk klaster Multi-wilayah, dengan konsol. Mereka dapat membuat daftar cluster dan melihat informasi tentang cluster individu. Mereka dapat melihat tag pada sumber daya apa pun di akun Anda. Mereka dapat terhubung ke database sebagai pengguna mana pun, termasuk admin. Mereka dapat melakukan operasi pencadangan dan pemulihan untuk cluster Aurora DSQL, termasuk memulai, menghentikan, dan memantau pekerjaan pencadangan dan pemulihan. Kebijakan ini mencakup AWS KMS izin yang memungkinkan operasi pada kunci yang dikelola pelanggan yang digunakan untuk enkripsi klaster. Mereka dapat meluncurkan AWS CloudShell dari AWS Management Console. Mereka dapat melihat metrik apa pun dari CloudWatch akun Anda. Mereka juga memiliki izin untuk membuat peran terkait layanan untuk `dsql.amazonaws.com` layanan, yang diperlukan untuk membuat cluster.

Anda dapat menemukan `AmazonAuroraDSQLConsoleFullAccess` kebijakan di konsol IAM dan AmazonAuroraDSQLConsoleFullAccessdi Panduan Referensi Kebijakan AWS Terkelola.

Detail izin

Kebijakan ini mencakup izin berikut.

- `dsql`—memberikan izin administratif penuh untuk semua sumber daya di Aurora DSQL melalui file. AWS Management Console
- `cloudwatch`—memberikan izin untuk mengambil jumlah batch data metrik dan melakukan matematika CloudWatch metrik pada data yang diambil.
- `tag`—memberikan izin untuk mengembalikan kunci tag dan nilai yang saat ini digunakan dalam yang ditentukan Wilayah AWS untuk akun panggilan.
- `backup and restore`—memberikan izin untuk memulai, menghentikan, dan memantau pencadangan dan pemulihan pekerjaan untuk klaster Aurora DSQL.
- `kms`—memberikan izin yang diperlukan untuk memvalidasi akses ke kunci yang dikelola pelanggan yang digunakan untuk enkripsi klaster Aurora DSQL saat membuat, memperbarui, atau menghubungkan ke cluster.
- `cloudshell`—memberikan izin untuk meluncurkan untuk berinteraksi dengan Aurora AWS CloudShell DSQL.
- `ec2`—memberikan izin untuk melihat informasi titik akhir VPC Amazon yang diperlukan untuk koneksi Aurora DSQL.

Anda dapat menemukan `AmazonAuroraDSQLReadOnlyAccess` kebijakan di konsol IAM dan AmazonAuroraDSQLReadOnlyAccessdi Panduan Referensi Kebijakan AWS Terkelola.

AWS kebijakan terkelola: Aurora DSQLService RolePolicy

Anda tidak dapat melampirkan Aurora DSQLService RolePolicy ke entitas IAM Anda. Kebijakan ini dilampirkan ke peran terkait layanan yang memungkinkan Aurora DSQL mengakses sumber daya akun.

Anda dapat menemukan `AuroraDSQLServiceRolePolicy` kebijakan di konsol IAM dan DSQLServiceRolePolicyAurora di Panduan Referensi Kebijakan AWS Terkelola.

Aurora DSQL memperbarui kebijakan terkelola AWS

Lihat detail tentang pembaruan kebijakan AWS terkelola untuk Aurora DSQL sejak layanan ini mulai melacak perubahan ini. Untuk peringatan otomatis tentang perubahan pada halaman ini, berlangganan umpan RSS di halaman riwayat Dokumen Aurora DSQL.

Perubahan	Deskripsi	Tanggal
AmazonAuroraDSQLFullAkses pembaruan	Menambahkan kemampuan untuk melakukan operasi pencadangan dan pemulihan untuk cluster Aurora DSQL, termasuk memulai, menghentikan, dan memantau pekerjaan. Ini juga menambahkan kemampuan untuk menggunakan kunci KMS yang dikelola pelanggan untuk enkripsi cluster. Untuk informasi selengkapnya, lihat AmazonAuroraDSQLFullMengakses dan Menggunakan peran terkait layanan di Aurora DSQL.	21 Mei 2025
AmazonAuroraDSQLConsoleFullAccess perbarui	Menambahkan kemampuan untuk melakukan operasi pencadangan dan pemulihan untuk cluster Aurora DSQL melalui. AWS Console Home Ini termasuk memulai, menghentikan, dan memantau pekerjaan. Ini juga mendukung penggunaan kunci KMS yang dikelola pelanggan untuk enkripsi dan peluncuran cluster. AWS CloudShell Untuk informasi selengkapnya, lihat AmazonAuroraDSQLConsoleFullAccessdan Menggunakan	21 Mei 2025

Perubahan	Deskripsi	Tanggal
	peran terkait layanan di Aurora DSQL.	
AmazonAuroraDSQLFullAkses pembaruan	Kebijakan ini menambahkan empat izin baru untuk membuat dan mengelola kluster database di beberapa Wilayah AWS:`PutMultiRegionProperties` ,, `PutWitnessRegion` `AddPeerCluster` , dan. `RemovePeerCluster` Izin ini mencakup kontrol tingkat sumber daya dan kunci kondisi sehingga Anda dapat mengontrol pengguna kluster mana yang dapat Anda modifikasi. Kebijakan ini juga menambahkan `GetVpcEndpointServiceName` izin untuk membantu Anda terhubung ke klaster Aurora DSQL Anda. AWS PrivateLink Untuk informasi selengkapnya, lihat Untuk informasi selengkapnya, lihat AmazonAuroraDSQLFullMengakses dan Menggunakan peran terkait layanan di Aurora DSQL.	13 Mei 2025

Perubahan	Deskripsi	Tanggal
AmazonAuroraDSQLReadOnlyAccess perbarui	Termasuk kemampuan untuk menentukan nama layanan titik akhir VPC yang benar saat menghubungkan ke cluster Aurora DSQL Anda melalui Aurora DSQL menciptakan titik akhir unik per sel, sehingga API ini membantu memastikan Anda dapat mengidentifikasi titik akhir yang benar untuk cluster Anda dan menghindari kesalahan koneksi. AWS PrivateLink Untuk informasi selengkapnya, lihat AmazonAuroraDSQLReadOnlyAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	13 Mei 2025

Perubahan	Deskripsi	Tanggal
AmazonAuroraDSQLConsoleFullAccess perbarui	Menambahkan izin baru ke Aurora DSQL untuk mendukung manajemen klaster Multi-wilayah dan koneksi titik akhir VPC. Izin baru meliputi: `PutMultiRegionProperties` `PutWitnessRegion` `AddPeerCluster` `RemovePeerCluster` `GetVpcEndpointServiceName` Untuk informasi selengkapnya, lihat AmazonAuroraDSQLConsoleFullAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	13 Mei 2025

Perubahan	Deskripsi	Tanggal
AuroraDsqlServiceLinkedRolePolicy perbarui	Menambahkan kemampuan untuk memublikasikan metrik ke `AWS/Usage` CloudWatch ruang nama `AWS/AuroraDSQL` dan ruang nama ke kebijakan. Hal ini memungkinkan layanan atau peran terkait untuk memancarkan data penggunaan dan kinerja yang lebih komprehensif ke CloudWatch lingkungan Anda. Untuk informasi selengkapnya, lihat AuroraDsqlServiceLinkedRolePolicydan Menggunakan peran terkait layanan di Aurora DSQL.	8 Mei 2025
Halaman dibuat	Mulai melacak kebijakan AWS terkelola yang terkait dengan Amazon Aurora DSQL	Desember 3, 2024

Perlindungan data di Amazon Aurora DSQL

Model tanggung jawab bersama model berlaku untuk perlindungan data di. Seperti yang dijelaskan dalam model ini, bertanggung jawab untuk melindungi infrastruktur global yang menjalankan semua AWS Cloud. Anda bertanggung jawab untuk mempertahankan kendali atas konten yang di-host pada infrastruktur ini. Anda juga bertanggung jawab atas tugas-tugas konfigurasi dan manajemen keamanan untuk 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 di Blog Keamanan .

Untuk tujuan perlindungan data, kami menyarankan Anda melindungi kredensil dan mengatur pengguna individu dengan AWS IAM Identity Center atau. AWS Identity and Access Management

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. Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Siapkan API dan pencatatan aktivitas pengguna dengan AWS CloudTrail. Untuk informasi tentang penggunaan jejak untuk menangkap aktivitas, lihat Bekerja dengan jejak di Panduan Pengguna.
- Gunakan solusi enkripsi, bersama dengan semua kontrol keamanan default di dalamnya Layanan AWS.
- Gunakan layanan keamanan terkelola tingkat lanjut seperti Amazon Macie, yang membantu menemukan dan mengamankan data sensitif yang disimpan di Amazon S3.

Kami sangat menyarankan agar Anda tidak pernah memasukkan informasi rahasia atau sensitif, seperti alamat email pelanggan Anda, ke dalam tag atau bidang teks bentuk bebas seperti bidang Nama. Ini termasuk saat Anda bekerja dengan atau lainnya menggunakan konsol, API AWS CLI, atau AWS SDKs. Data apa pun yang Anda masukkan ke dalam tanda atau bidang isian bebas yang digunakan untuk nama dapat digunakan untuk log penagihan atau log 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.

Enkripsi data

Amazon Aurora DSQL menyediakan infrastruktur penyimpanan yang sangat tahan lama yang dirancang untuk penyimpanan data utama dan kritis misi. Data disimpan secara berlebihan di beberapa perangkat di beberapa fasilitas di Wilayah Aurora DSQL.

Enkripsi bergerak

Secara default, enkripsi dalam perjalanan dikonfigurasi untuk Anda. Aurora DSQL menggunakan TLS untuk mengenkripsi semua lalu lintas antara klien SQL Anda dan Aurora DSQL.

Enkripsi dan penandatanganan data dalam transit antara AWS CLI, SDK, atau klien API dan titik akhir Aurora DSQL:

- Aurora DSQL menyediakan titik akhir HTTPS untuk mengenkripsi data dalam perjalanan.

- Untuk melindungi integritas permintaan API ke Aurora DSQL, panggilan API harus ditandatangani oleh pemanggil. Panggilan ditandatangani oleh sertifikat X.509 atau kunci akses AWS rahasia pelanggan sesuai dengan Proses Penandatanganan Versi Tanda Tangan 4 (Sigv4). Untuk informasi selengkapnya, lihat Proses Penandatanganan Versi Tanda Tangan 4 di Referensi Umum AWS.
- Gunakan AWS CLI atau salah satu AWS SDKs untuk membuat permintaan AWS. Alat-alat ini secara otomatis menandatangani permintaan untuk Anda dengan kunci akses yang Anda tentukan saat Anda mengonfigurasi alat.

Untuk enkripsi saat istirahat, lihatEnkripsi saat istirahat di Aurora DSQL.

Privasi lalu lintas antar jaringan

Koneksi dilindungi baik antara Aurora DSQL dan aplikasi lokal dan antara Aurora DSQL dan sumber daya lain dalam hal yang sama. AWS Wilayah AWS

Anda memiliki dua opsi konektivitas antara jaringan pribadi Anda dan AWS:

- Koneksi AWS Site-to-Site VPN. Untuk informasi selengkapnya, lihat Apa itu AWS Site-to-Site VPN?
- AWS Direct Connect Koneksi. Untuk informasi lebih lanjut, lihat Apa itu AWS Direct Connect?

Anda mendapatkan akses ke Aurora DSQL melalui jaringan dengan menggunakan AWS operasi API yang diterbitkan. Klien harus mendukung hal-hal berikut:

- Keamanan Lapisan Pengangkutan (TLS). Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Sandi cocok dengan sistem kerahasiaan maju sempurna (perfect forward secrecy, PFS) seperti DHE (Ephemeral Diffie-Hellman) atau ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Sebagian besar sistem modern seperti Java 7 dan versi lebih baru mendukung mode-mode ini.

Mengkonfigurasi SSL/TLS sertifikat untuk koneksi Aurora DSQL

Aurora DSQL membutuhkan semua koneksi untuk menggunakan enkripsi Transport Layer Security (TLS). Untuk membuat koneksi aman, sistem klien Anda harus mempercayai Amazon Root Certificate Authority (Amazon Root CA 1). Sertifikat ini sudah diinstal sebelumnya pada banyak sistem operasi. Bagian ini memberikan instruksi untuk memverifikasi sertifikat Amazon Root CA 1

yang sudah diinstal sebelumnya pada berbagai sistem operasi, dan memandu Anda melalui proses menginstal sertifikat secara manual jika belum ada.

Kami merekomendasikan menggunakan PostgreSQL versi 17.

> **Important**
>
> Untuk lingkungan produksi, sebaiknya gunakan mode `verify-full` SSL untuk memastikan tingkat keamanan koneksi tertinggi. Mode ini memverifikasi bahwa sertifikat server ditandatangani oleh otoritas sertifikat tepercaya dan bahwa nama host server cocok dengan sertifikat.

Memverifikasi sertifikat pra-instal

Di sebagian besar sistem operasi, Amazon Root CA 1 sudah diinstal sebelumnya. Untuk memvalidasi ini, Anda dapat mengikuti langkah-langkah di bawah ini.

Linux (RedHat/CentOS/Fedora)

Jalankan perintah berikut di terminal Anda:

```
trust list | grep "Amazon Root CA 1"
```

Jika sertifikat diinstal, Anda melihat output berikut:

```
label: Amazon Root CA 1
```

macOS

1. Buka Pencarian Sorotan (Perintah+Spasi)
2. Cari Akses Gantungan Kunci
3. Pilih Akar Sistem di bawah Gantungan Kunci Sistem
4. Cari Amazon Root CA 1 dalam daftar sertifikat

Windows

> **Note**
>
> Karena masalah yang diketahui dengan klien Windows psql, menggunakan sertifikat root sistem (`sslrootcert=system`) dapat mengembalikan kesalahan berikut:. `SSL error: unregistered scheme` Anda dapat mengikuti Menghubungkan dari Windows sebagai cara alternatif untuk terhubung ke cluster Anda menggunakan SSL.

Jika Amazon Root CA 1 tidak diinstal di sistem operasi Anda, ikuti langkah-langkah di bawah ini.

Instalasi sertifikat

Jika `Amazon Root CA 1` sertifikat tidak diinstal sebelumnya pada sistem operasi Anda, Anda harus menginstalnya secara manual untuk membuat koneksi aman ke cluster Aurora DSQL Anda.

Instalasi sertifikat Linux

Ikuti langkah-langkah ini untuk menginstal sertifikat Amazon Root CA pada sistem Linux.

1. Unduh Sertifikat Root:

```
wget https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

2. Salin sertifikat ke toko kepercayaan:

```
sudo cp ./AmazonRootCA1.pem /etc/pki/ca-trust/source/anchors/
```

3. Perbarui toko kepercayaan CA:

```
sudo update-ca-trust
```

4. Verifikasi instalasi:

```
trust list | grep "Amazon Root CA 1"
```

Instalasi sertifikat macOS

Langkah-langkah pemasangan sertifikat ini bersifat opsional. Ini Instalasi sertifikat Linux juga berfungsi untuk macOS.

1. Unduh Sertifikat Root:

```
wget https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

2. Tambahkan sertifikat ke gantungan kunci Sistem:

```
sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain
 AmazonRootCA1.pem
```

3. Verifikasi instalasi:

```
security find-certificate -a -c "Amazon Root CA 1" -p /Library/Keychains/
System.keychain
```

Menghubungkan dengan SSL/TLS verifikasi

Sebelum mengonfigurasi SSL/TLS sertifikat untuk koneksi aman ke cluster Aurora DSQL Anda, pastikan Anda memiliki prasyarat berikut.

- PostgreSQL versi 17 diinstal
- AWS CLI dikonfigurasi dengan kredensil yang sesuai
- Informasi titik akhir cluster Aurora DSQL

Menghubungkan dari Linux

1. Hasilkan dan atur token otentikasi:

```
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token --region=your-
cluster-region --hostname your-cluster-endpoint)
```

2. Connect menggunakan sertifikat sistem (jika sudah diinstal sebelumnya):

```
PGSSLROOTCERT=system \
PGSSLMODE=verify-full \
```

```
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

3. Atau, sambungkan menggunakan sertifikat yang diunduh:

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

> **Note**
>
> Untuk selengkapnya tentang pengaturan PGSSLMODE, lihat sslmode di dokumentasi PostgreSQL 17 Database Connection Control Functions.

Menghubungkan dari macOS

1. Hasilkan dan atur token otentikasi:

```
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token --region=your-cluster-region --hostname your-cluster-endpoint)
```

2. Connect menggunakan sertifikat sistem (jika sudah diinstal sebelumnya):

```
PGSSLROOTCERT=system \
PGSSLMODE=verify-full \
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

3. Atau, unduh sertifikat root dan simpan sebagai `root.pem` (jika sertifikat tidak diinstal sebelumnya)

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql –dbname postgres \
--username admin \
--host your_cluster_endpoint
```

4. Connect menggunakan psql:

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql –dbname postgres \
--username admin \
--host your_cluster_endpoint
```

Menghubungkan dari Windows

Menggunakan Command Prompt

1. Hasilkan token otentikasi:

```
aws dsql generate-db-connect-admin-auth-token ^
--region=your-cluster-region ^
--expires-in=3600 ^
--hostname=your-cluster-endpoint
```

2. Mengatur variabel lingkungan kata sandi:

```
set "PGPASSWORD=token-from-above"
```

3. Atur konfigurasi SSL:

```
set PGSSLROOTCERT=C:\full\path\to\root.pem
set PGSSLMODE=verify-full
```

4. Connect ke database:

```
"C:\Program Files\PostgreSQL\17\bin\psql.exe" --dbname postgres ^
--username admin ^
--host your-cluster-endpoint
```

Menggunakan PowerShell

1. Hasilkan dan atur token otentikasi:

```
$env:PGPASSWORD = (aws dsql generate-db-connect-admin-auth-token --region=your-cluster-region --expires-in=3600 --hostname=your-cluster-endpoint)
```

2. Atur konfigurasi SSL:

```
$env:PGSSLROOTCERT='C:\full\path\to\root.pem'
$env:PGSSLMODE='verify-full'
```

3. Connect ke database:

```
 "C:\Program Files\PostgreSQL\17\bin\psql.exe" --dbname postgres `
--username admin `
--host your-cluster-endpoint
```

Sumber daya tambahan

- Dokumentasi PostgreSQL SSL
- Layanan Amazon Trust

Enkripsi data untuk Amazon Aurora DSQL

Amazon Aurora DSQL mengenkripsi semua data pengguna saat istirahat. Untuk keamanan yang ditingkatkan, enkripsi ini menggunakan AWS Key Management Service (AWS KMS). Fungsi ini membantu mengurangi beban operasional dan kompleksitas yang terlibat dalam melindungi data sensitif. Enkripsi saat istirahat membantu Anda:

- Mengurangi beban operasional untuk melindungi data sensitif
- Membangun aplikasi yang sensitif terhadap keamanan yang memenuhi kepatuhan enkripsi yang ketat dan persyaratan peraturan
- Tambahkan lapisan perlindungan data tambahan dengan selalu mengamankan data Anda dalam klaster terenkripsi
- Mematuhi kebijakan organisasi, peraturan industri atau pemerintah, dan persyaratan kepatuhan

Dengan Aurora DSQL, Anda dapat membangun aplikasi yang sensitif terhadap keamanan yang memenuhi kepatuhan enkripsi dan persyaratan peraturan yang ketat. Bagian berikut menjelaskan

cara mengkonfigurasi enkripsi untuk database Aurora DSQL baru dan yang sudah ada dan mengelola kunci enkripsi Anda.

Topik

Jenis kunci KMS untuk Aurora DSQL

Aurora DSQL terintegrasi dengan AWS KMS untuk mengelola kunci enkripsi untuk cluster Anda. Untuk mempelajari lebih lanjut tentang jenis dan status kunci, lihat AWS Key Management Service konsep di Panduan AWS Key Management Service Pengembang. Saat membuat klaster baru, Anda dapat memilih dari jenis kunci KMS berikut untuk mengenkripsi klaster Anda:

Kunci milik AWS

Jenis enkripsi default. Aurora DSQL memiliki kunci tanpa biaya tambahan kepada Anda. Amazon Aurora DSQL secara transparan mendekripsi data klaster saat Anda mengakses kluster terenkripsi. Anda tidak perlu mengubah kode atau aplikasi untuk menggunakan atau mengelola cluster terenkripsi, dan semua kueri Aurora DSQL berfungsi dengan data terenkripsi Anda.

Kunci yang dikelola pelanggan

Anda membuat, memiliki, dan mengelola kunci di Anda Akun AWS. Anda memiliki kontrol penuh atas tombol KMS. AWS KMS dikenakan biaya.

Enkripsi saat istirahat menggunakan Kunci milik AWS tersedia tanpa biaya tambahan. Namun, AWS KMS biaya berlaku untuk kunci yang dikelola pelanggan. Untuk informasi lebih lanjut, lihat halaman AWS KMS Harga.

Anda dapat beralih di antara jenis-jenis kunci ini kapan saja. Untuk informasi selengkapnya tentang jenis kunci, lihat Kunci yang dikelola pelanggan dan Kunci milik AWSdi Panduan AWS Key Management Service Pengembang.

> **Note**
>
> Enkripsi Aurora DSQL saat istirahat tersedia di semua Wilayah di AWS mana Aurora DSQL tersedia.

Enkripsi saat istirahat di Aurora DSQL

Amazon Aurora DSQL menggunakan Standar Enkripsi Lanjutan 256-bit (AES-256) untuk mengenkripsi data Anda saat istirahat. Enkripsi ini membantu melindungi data Anda dari akses tidak sah ke penyimpanan yang mendasarinya. AWS KMS mengelola kunci enkripsi untuk cluster Anda. Anda dapat menggunakan defaultKunci milik AWS, atau memilih untuk menggunakan sendiri AWS KMS Kunci yang dikelola pelanggan. Untuk mempelajari lebih lanjut tentang menentukan dan mengelola kunci untuk klaster Aurora DSQL Anda, lihat dan. Membuat cluster Aurora DSQL terenkripsi Menghapus atau memperbarui kunci untuk cluster Aurora DSQL Anda

Topik

- Kunci milik AWS
- Kunci yang dikelola pelanggan

Kunci milik AWS

Aurora DSQL mengenkripsi semua cluster secara default dengan. Kunci milik AWS Kunci ini gratis untuk digunakan dan diputar setiap tahun untuk melindungi sumber daya akun Anda. Anda tidak perlu melihat, mengelola, menggunakan, atau mengaudit kunci ini, sehingga tidak ada tindakan yang diperlukan untuk perlindungan data. Untuk informasi selengkapnya Kunci milik AWS, lihat Kunci milik AWSdi Panduan AWS Key Management Service Pengembang.

Kunci yang dikelola pelanggan

Anda membuat, memiliki, dan mengelola kunci yang dikelola pelanggan di Anda Akun AWS. Anda memiliki kendali penuh atas kunci KMS ini, termasuk kebijakan, materi enkripsi, tag, dan aliasnya. Untuk informasi selengkapnya tentang mengelola izin, lihat Kunci yang dikelola pelanggan di Panduan AWS Key Management Service Pengembang.

Saat Anda menentukan kunci terkelola pelanggan untuk enkripsi tingkat kluster, Aurora DSQL mengenkripsi cluster dan semua data regionalnya dengan kunci itu. Untuk mencegah kehilangan data dan mempertahankan akses cluster, Aurora DSQL memerlukan akses ke kunci enkripsi Anda. Jika Anda menonaktifkan kunci terkelola pelanggan, menjadwalkan kunci Anda untuk dihapus, atau memiliki kebijakan yang membatasi akses layanan Anda, status enkripsi untuk klaster Anda berubah menjadi. `KMS_KEY_INACCESSIBLE` Ketika Aurora DSQL tidak dapat mengakses kunci, pengguna tidak dapat terhubung ke cluster, status enkripsi untuk klaster berubah`KMS_KEY_INACCESSIBLE`, dan layanan kehilangan akses ke data cluster.

Untuk klaster Multi-region, pelanggan dapat mengonfigurasi kunci AWS KMS enkripsi setiap wilayah secara terpisah, dan setiap cluster regional menggunakan kunci enkripsi tingkat klaster sendiri. Jika Aurora DSQL tidak dapat mengakses kunci enkripsi untuk peer di klaster Multi-wilayah, status untuk rekan tersebut menjadi `KMS_KEY_INACCESSIBLE` dan menjadi tidak tersedia untuk operasi baca dan tulis. Rekan-rekan lainnya melanjutkan operasi normal.

> **Note**
>
> Jika Aurora DSQL tidak dapat mengakses kunci terkelola pelanggan Anda, status enkripsi klaster Anda akan berubah menjadi. `KMS_KEY_INACCESSIBLE` Setelah Anda mengembalikan akses kunci, layanan akan secara otomatis mendeteksi pemulihan dalam waktu 15 menit. Untuk informasi selengkapnya, lihat Cluster idling.
> Untuk klaster Multi-wilayah, jika akses kunci hilang untuk waktu yang lama, waktu pemulihan klaster tergantung pada berapa banyak data yang ditulis saat kunci tidak dapat diakses.

Menggunakan AWS KMS dan kunci data dengan Aurora DSQL

Fitur enkripsi Aurora DSQL saat istirahat menggunakan AWS KMS key dan hierarki kunci data untuk melindungi data cluster Anda.

Kami menyarankan Anda merencanakan strategi enkripsi Anda sebelum menerapkan cluster Anda di Aurora DSQL. Jika Anda menyimpan data sensitif atau rahasia di Aurora DSQL, pertimbangkan untuk menyertakan enkripsi sisi klien dalam paket Anda. Dengan cara ini, Anda dapat mengenkripsi data sedekat mungkin dengan asalnya, dan memastikan perlindungannya sepanjang siklus hidupnya.

Topik

- Menggunakan AWS KMS key s dengan Aurora DSQL
- Menggunakan tombol cluster dengan Aurora DSQL

- Caching kunci cluster

Menggunakan AWS KMS key s dengan Aurora DSQL

Enkripsi saat istirahat melindungi cluster Aurora DSQL Anda di bawah file. AWS KMS key Secara default, Aurora DSQL menggunakan Kunci milik AWS, kunci enkripsi multi-tenant yang dibuat dan dikelola di akun layanan Aurora DSQL. Tetapi Anda dapat mengenkripsi cluster Aurora DSQL Anda di bawah kunci yang dikelola pelanggan di Anda. Akun AWS Anda dapat memilih kunci KMS yang berbeda untuk setiap cluster, bahkan jika itu berpartisipasi dalam pengaturan Multi-wilayah.

Anda memilih kunci KMS untuk klaster saat Anda membuat atau memperbarui cluster. Anda dapat mengubah kunci KMS untuk cluster kapan saja, baik di konsol Aurora DSQL atau dengan menggunakan operasi. `UpdateCluster` Proses peralihan kunci tidak memerlukan downtime atau menurunkan layanan.

> **Important**
>
> Aurora DSQL hanya mendukung tombol KMS simetris. Anda tidak dapat menggunakan kunci KMS asimetris untuk mengenkripsi cluster Aurora DSQL Anda.

Kunci yang dikelola pelanggan memberikan manfaat berikut.

- Anda membuat dan mengelola kunci KMS, termasuk menyetel kebijakan utama dan kebijakan IAM untuk mengontrol akses ke kunci KMS. Anda dapat mengaktifkan dan menonaktifkan kunci KMS, mengaktifkan dan menonaktifkan rotasi kunci otomatis, dan menghapus kunci KMS ketika sudah tidak digunakan.
- Anda dapat menggunakan kunci yang dikelola pelanggan dengan material kunci yang diimpor atau kunci yang dikelola pelanggan di penyimpanan kunci kustom yang Anda miliki dan kelola.
- Anda dapat mengaudit enkripsi dan dekripsi cluster Aurora DSQL Anda dengan memeriksa panggilan Aurora DSQL API ke dalam log. AWS KMS AWS CloudTrail

Namun, Kunci milik AWS ini gratis dan penggunaannya tidak dihitung terhadap AWS KMS sumber daya atau kuota permintaan. Kunci yang dikelola pelanggan dikenakan biaya untuk setiap panggilan API dan AWS KMS kuota berlaku untuk kunci ini.

Menggunakan tombol cluster dengan Aurora DSQL

Aurora DSQL menggunakan AWS KMS key for the cluster untuk menghasilkan dan mengenkripsi kunci data unik untuk cluster, yang dikenal sebagai kunci cluster.

Kunci cluster digunakan sebagai kunci enkripsi kunci. Aurora DSQL menggunakan kunci cluster ini untuk melindungi kunci enkripsi data yang digunakan untuk mengenkripsi data cluster. Aurora DSQL menghasilkan kunci enkripsi data unik untuk setiap struktur yang mendasarinya dalam sebuah cluster, tetapi beberapa item cluster mungkin dilindungi oleh kunci enkripsi data yang sama.

Untuk mendekripsi kunci cluster, Aurora DSQL mengirimkan permintaan AWS KMS ketika Anda pertama kali mengakses cluster terenkripsi. Agar klaster tetap tersedia, Aurora DSQL secara berkala memverifikasi akses dekripsi ke kunci KMS, bahkan ketika Anda tidak aktif mengakses cluster.

Aurora DSQL menyimpan dan menggunakan kunci cluster dan kunci enkripsi data di luar. AWS KMS Layanan ini melindungi semua kunci dengan enkripsi Advanced Encryption Standard (AES) dan kunci enkripsi 256-bit. Kemudian, ia menyimpan kunci terenkripsi dengan data terenkripsi sehingga tersedia untuk mendekripsi data cluster sesuai permintaan.

Jika Anda mengubah kunci KMS untuk cluster Anda, Aurora DSQL mengenkripsi ulang kunci cluster yang ada dengan kunci KMS baru.

Caching kunci cluster

Untuk menghindari panggilan AWS KMS untuk setiap operasi Aurora DSQL, Aurora DSQL menyimpan kunci cluster plaintext untuk setiap pemanggil dalam memori. Jika Aurora DSQL mendapat permintaan untuk kunci cluster cache setelah 15 menit tidak aktif, ia mengirimkan permintaan baru untuk AWS KMS mendekripsi kunci cluster. Panggilan ini akan menangkap setiap perubahan yang dibuat pada kebijakan akses AWS KMS key dalam AWS KMS atau AWS Identity and Access Management (IAM) setelah permintaan terakhir untuk mendekripsi kunci klaster.

Otorisasi penggunaan Anda AWS KMS key untuk Aurora DSQL

Jika Anda menggunakan kunci yang dikelola pelanggan di akun Anda untuk melindungi klaster Aurora DSQL Anda, kebijakan pada kunci tersebut harus memberikan izin kepada Aurora DSQL untuk menggunakannya atas nama Anda.

Anda memiliki kendali penuh atas kebijakan pada kunci yang dikelola pelanggan. Aurora DSQL tidak memerlukan otorisasi tambahan untuk menggunakan default untuk Kunci milik AWS melindungi cluster Aurora DSQL di Anda. Akun AWS

Kebijakan kunci untuk kunci yang dikelola pelanggan

Ketika Anda memilih kunci yang dikelola pelanggan untuk melindungi cluster Aurora DSQL, Aurora DSQL memerlukan izin untuk menggunakan AWS KMS key atas nama kepala sekolah yang membuat pilihan. Prinsipal itu, pengguna atau peran, harus memiliki izin pada Aurora DSQL AWS KMS key yang dibutuhkan. Anda dapat memberikan izin ini dalam kebijakan utama, atau kebijakan IAM.

Minimal, Aurora DSQL memerlukan izin berikut pada kunci yang dikelola pelanggan:

- `kms:Encrypt`
- `kms:Decrypt`
- `kms:ReEncrypt*`(untuk kms: ReEncryptFrom dan kms:ReEncryptTo)
- `kms:GenerateDataKey`
- `kms:DescribeKey`

Sebagai contoh, kebijakan kunci berikut hanya menyediakan izin yang diperlukan. Kebijakan ini memiliki efek sebagai berikut:

- Memungkinkan Aurora DSQL untuk menggunakan AWS KMS key dalam operasi kriptografi, tetapi hanya ketika bertindak atas nama kepala sekolah di akun yang memiliki izin untuk menggunakan Aurora DSQL. Jika prinsipal yang ditentukan dalam pernyataan kebijakan tidak memiliki izin untuk menggunakan Aurora DSQL, panggilan gagal, bahkan ketika itu berasal dari layanan Aurora DSQL.
- Kunci `kms:ViaService` kondisi mengizinkan izin hanya jika permintaan berasal dari Aurora DSQL atas nama prinsipal yang tercantum dalam pernyataan kebijakan. Pengguna utama ini tidak dapat memanggil operasi ini secara langsung.
- Memberikan AWS KMS key administrator (pengguna yang dapat mengambil `db-team` peran) akses hanya-baca ke AWS KMS key

Sebelum menggunakan kebijakan kunci contoh, ganti prinsip contoh dengan prinsip aktual dari Anda. Akun AWS

```
{
  "Sid": "Enable dsql IAM User Permissions",
  "Effect": "Allow",
```

```
  "Principal": {
    "Service": "dsql.amazonaws.com"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey",
    "kms:Encrypt",
    "kms:ReEncryptFrom",
    "kms:ReEncryptTo"
  ],
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "kms:EncryptionContext:aws:dsql:ClusterId": "w4abucpbwuxx",
      "aws:SourceArn": "arn:aws:dsql:us-east-2:111122223333:cluster/w4abucpbwuxx"
    }
  }
},
{
  "Sid": "Enable dsql IAM User Describe Permissions",
  "Effect": "Allow",
  "Principal": {
    "Service": "dsql.amazonaws.com"
  },
  "Action": "kms:DescribeKey",
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "aws:SourceArn": "arn:aws:dsql:us-east-2:111122223333:cluster/w4abucpbwuxx"
    }
  }
}
```

Konteks enkripsi Aurora DSQL

Konteks enkripsi adalah seperangkat pasangan kunci-nilai yang berisi data non-rahasia yang arbitrer. Ketika Anda menyertakan konteks enkripsi dalam permintaan untuk mengenkripsi data, secara AWS KMS kriptografis mengikat konteks enkripsi ke data terenkripsi. Untuk mendekripsi data, Anda harus meneruskan konteks enkripsi yang sama.

Aurora DSQL menggunakan konteks enkripsi yang sama di semua operasi kriptografi. AWS KMS Jika Anda menggunakan kunci yang dikelola pelanggan untuk melindungi klaster Aurora DSQL Anda,

Anda dapat menggunakan konteks enkripsi untuk mengidentifikasi penggunaan catatan dan log audit AWS KMS key dalam. Itu juga muncul dalam teks biasa di log seperti yang ada di. AWS CloudTrail

Konteks enkripsi juga dapat digunakan sebagai syarat untuk otorisasi dalam kebijakan.

Dalam permintaannya AWS KMS, Aurora DSQL menggunakan konteks enkripsi dengan pasangan kunci-nilai:

```
"encryptionContext": {
  "aws:dsql:ClusterId": "w4abucpbwuxx"
},
```

Pasangan kunci-nilai mengidentifikasi cluster yang dienkripsi Aurora DSQL. Kuncinya adalah `aws:dsql:ClusterId`. Nilainya adalah pengidentifikasi cluster.

Memantau interaksi Aurora DSQL dengan AWS KMS

Jika Anda menggunakan kunci yang dikelola pelanggan untuk melindungi klaster Aurora DSQL Anda, Anda dapat menggunakan AWS CloudTrail log untuk melacak permintaan yang dikirim Aurora DSQL atas nama Anda. AWS KMS

Perluas bagian berikut untuk mempelajari bagaimana Aurora DSQL menggunakan operasi dan. AWS KMS `GenerateDataKey` `Decrypt`

`GenerateDataKey`

Saat Anda mengaktifkan enkripsi saat istirahat di cluster, Aurora DSQL membuat kunci cluster yang unik. Ini mengirimkan `GenerateDataKey` permintaan untuk AWS KMS yang menentukan AWS KMS key untuk cluster.

Peristiwa yang mencatat operasi `GenerateDataKey` serupa dengan peristiwa contoh berikut. Pengguna adalah akun layanan Aurora DSQL. Parameter termasuk Amazon Resource Name (ARN) dari AWS KMS key, penentu kunci yang memerlukan kunci 256-bit, dan konteks enkripsi yang mengidentifikasi cluster.

```
{
    "eventVersion": "1.11",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "dsql.amazonaws.com"
```

```
    },
    "eventTime": "2025-05-16T18:41:24Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "dsql.amazonaws.com",
    "userAgent": "dsql.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:dsql:ClusterId": "w4abucpbwuxx"
        },
        "keySpec": "AES_256",
        "keyId": "arn:aws:kms:us-east-1:982127530226:key/8b60dd9f-2ff8-4b1f-8a9c-
bf570cbfdb5e"
    },
    "responseElements": null,
    "requestID": "2da2dc32-d3f4-4d6c-8a41-aff27cd9a733",
    "eventID": "426df0a6-ba56-3244-9337-438411f826f4",
    "readOnly": true,
    "resources": [
        {
            "accountId": "AWS Internal",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:us-east-1:982127530226:key/8b60dd9f-2ff8-4b1f-8a9c-
bf570cbfdb5e"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "111122223333",
    "sharedEventID": "f88e0dd8-6057-4ce0-b77d-800448426d4e",
    "vpcEndpointId": "AWS Internal",
    "vpcEndpointAccountId": "vpce-1a2b3c4d5e6f1a2b3",
    "eventCategory": "Management"
}
```


Dekripsi

Saat Anda mengakses cluster Aurora DSQL terenkripsi, Aurora DSQL perlu mendekripsi kunci cluster sehingga dapat mendekripsi kunci di bawahnya dalam hierarki. Kemudian mendekripsi data di cluster. Untuk mendekripsi kunci cluster, Aurora DSQL mengirimkan `Decrypt` permintaan yang menentukan untuk AWS KMS cluster. AWS KMS key

Peristiwa yang mencatat operasi `Decrypt` serupa dengan peristiwa contoh berikut. Pengguna adalah kepala sekolah Anda Akun AWS yang mengakses cluster. Parameter termasuk kunci cluster terenkripsi (sebagai gumpalan ciphertext) dan konteks enkripsi yang mengidentifikasi cluster. AWS KMS berasal dari ID dari AWS KMS key ciphertext.


```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AWSService",
    "invokedBy": "dsql.amazonaws.com"
  },
  "eventTime": "2018-02-14T16:42:39Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Decrypt",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "dsql.amazonaws.com",
  "userAgent": "dsql.amazonaws.com",
  "requestParameters": {
    "keyId": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "encryptionContext": {
      "aws:dsql:ClusterId": "w4abucpbwuxx"
    },
    "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
  },
  "responseElements": null,
  "requestID": "11cab293-11a6-11e8-8386-13160d3e5db5",
  "eventID": "b7d16574-e887-4b5b-a064-bf92f8ec9ad3",
  "readOnly": true,
  "resources": [
    {
      "ARN": "arn:aws:kms:us-
east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "accountId": "AWS Internal",
      "type": "AWS::KMS::Key"
    }
  ],
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "111122223333",
  "sharedEventID": "d99f2dc5-b576-45b6-aa1d-3a3822edbeeb",
  "vpcEndpointId": "AWS Internal",
  "vpcEndpointAccountId": "vpce-1a2b3c4d5e6f1a2b3",
```

```
  "eventCategory": "Management"
}
```

Membuat cluster Aurora DSQL terenkripsi

Semua cluster Aurora DSQL dienkripsi saat istirahat. Secara default, cluster menggunakan tanpa Kunci milik AWS biaya, atau Anda dapat menentukan AWS KMS kunci kustom. Ikuti langkah-langkah ini untuk membuat cluster terenkripsi Anda baik dari AWS Management Console atau. AWS CLI

Console

Untuk membuat cluster terenkripsi di AWS Management Console

1. Masuk ke Konsol AWS Manajemen dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql/
2. Di panel navigasi di sisi kiri konsol, pilih Klaster.
3. Pilih Create Cluster di kanan atas dan pilih Single-Region.
4. Dalam pengaturan enkripsi Cluster, pilih salah satu opsi berikut.
   - Terima pengaturan default untuk mengenkripsi dengan tanpa Kunci milik AWS biaya tambahan.
   - Pilih Sesuaikan pengaturan enkripsi (lanjutan) untuk menentukan kunci KMS kustom. Kemudian, cari atau masukkan ID atau alias kunci KMS Anda. Atau, pilih Buat AWS KMS kunci untuk membuat kunci baru di AWS KMS Konsol.
5. Pilih Buat klaster.

Untuk mengonfirmasi jenis enkripsi klaster Anda, navigasikan ke halaman Cluster dan pilih ID klaster untuk melihat detail klaster. Tinjau tab pengaturan Cluster Pengaturan kunci KMS Cluster menunjukkan kunci default Aurora DSQL untuk cluster yang AWS menggunakan kunci yang dimiliki atau ID kunci untuk jenis enkripsi lainnya.

> **Note**
>
> Jika Anda memilih untuk memiliki dan mengelola kunci Anda sendiri, pastikan Anda menetapkan kebijakan kunci KMS dengan tepat. Untuk contoh dan informasi lebih lanjut, lihatthe section called “Kebijakan kunci untuk kunci yang dikelola pelanggan”.

CLI

Untuk membuat cluster yang dienkripsi dengan default Kunci milik AWS

- Gunakan perintah berikut untuk membuat cluster Aurora DSQL.

```
aws dsql create-cluster
```

Seperti yang ditunjukkan dalam detail enkripsi berikut, status enkripsi untuk cluster diaktifkan secara default, dan jenis enkripsi default adalah kunci yang dimiliki AWS. Cluster sekarang dienkripsi dengan kunci AWS default yang dimiliki di akun layanan Aurora DSQL.

```
"encryptionDetails": {
  "encryptionType" : "AWS_OWNED_KMS_KEY",
  "encryptionStatus" : "ENABLED"
}
```

Untuk membuat klaster yang dienkripsi dengan kunci terkelola pelanggan Anda

- Gunakan perintah berikut untuk membuat cluster Aurora DSQL, mengganti ID kunci dalam teks merah dengan ID kunci yang dikelola pelanggan Anda.

```
aws dsql create-cluster \
--kms-encryption-key d41d8cd98f00b204e9800998ecf8427e
```

Seperti yang ditunjukkan dalam detail enkripsi berikut, status enkripsi untuk cluster diaktifkan secara default, dan jenis enkripsi adalah kunci KMS yang dikelola pelanggan. Cluster sekarang dienkripsi dengan kunci Anda.

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:kms:us-east-1:111122223333:key/
d41d8cd98f00b204e9800998ecf8427e",
  "encryptionStatus" : "ENABLED"
}
```

Menghapus atau memperbarui kunci untuk cluster Aurora DSQL Anda

Anda dapat menggunakan atau AWS CLI untuk memperbarui AWS Management Console atau menghapus kunci enkripsi pada cluster yang ada di Amazon Aurora DSQL. Jika Anda menghapus kunci tanpa menggantinya, Aurora DSQL menggunakan default. Kunci milik AWS Ikuti langkah-langkah ini untuk memperbarui kunci enkripsi cluster yang ada dari konsol Aurora DSQL atau. AWS CLI

Console

Untuk memperbarui atau menghapus kunci enkripsi di AWS Management Console

1. Masuk ke Konsol AWS Manajemen dan buka konsol Aurora DSQL di. https://console.aws.amazon.com/dsql/
2. Di panel navigasi di sisi kiri konsol, pilih Klaster.
3. Dari tampilan daftar, temukan dan pilih baris cluster yang ingin Anda perbarui.
4. Pilih menu Tindakan dan kemudian pilih Ubah.
5. Dalam pengaturan enkripsi Cluster, pilih salah satu opsi berikut untuk mengubah pengaturan enkripsi Anda.
   - Jika Anda ingin beralih dari kunci khusus ke kunci Kunci milik AWS, hapus pilihan Sesuaikan pengaturan enkripsi (lanjutan). Pengaturan default akan menerapkan dan mengenkripsi cluster Anda dengan tanpa Kunci milik AWS biaya.
   - Jika Anda ingin beralih dari satu kunci KMS kustom ke yang lain atau dari tombol Kunci milik AWS ke KMS, pilih opsi Sesuaikan pengaturan enkripsi (lanjutan) jika belum dipilih. Kemudian, cari dan pilih ID atau alias kunci yang ingin Anda gunakan. Atau, pilih Buat AWS KMS kunci untuk membuat kunci baru di AWS KMS Konsol.
6. Pilih Simpan.

CLI

Contoh berikut menunjukkan cara menggunakan AWS CLI untuk memperbarui cluster terenkripsi.

Untuk memperbarui cluster terenkripsi dengan default Kunci milik AWS

```
aws dsql update-cluster \
```

```
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key "AWS_OWNED_KMS_KEY"
```

Deskripsi cluster diatur ke ENABLED dan EncryptionType adalahAWS_OWNED_KMS_KEY. EncryptionStatus

```
"encryptionDetails": {
  "encryptionType" : "AWS_OWNED_KMS_KEY",
  "encryptionStatus" : "ENABLED"
}
```

Cluster ini sekarang dienkripsi menggunakan default Kunci milik AWS di akun layanan Aurora DSQL.

Untuk memperbarui cluster terenkripsi dengan kunci terkelola pelanggan untuk Aurora DSQL

Perbarui cluster terenkripsi, seperti pada contoh berikut:

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-a123-
ab1234a1b234
```

Transisi deskripsi cluster ke UPDATING dan EncryptionType isCUSTOMER_MANAGED_KMS_KEY. EncryptionStatus Setelah Aurora DSQL selesai menyebarkan kunci baru melalui platform, status enkripsi akan dialihkan ke ENABLED

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:us-east-1:kms:key/abcd1234-abcd-1234-a123-ab1234a1b234",
  "encryptionStatus" : "ENABLED"
}
```

> **Note**
>
> Jika Anda memilih untuk memiliki dan mengelola kunci Anda sendiri, pastikan Anda menetapkan kebijakan kunci KMS dengan tepat. Untuk contoh dan informasi lebih lanjut, lihatthe section called “Kebijakan kunci untuk kunci yang dikelola pelanggan”.

Pertimbangan untuk enkripsi dengan Aurora DSQL

- Aurora DSQL mengenkripsi semua data cluster saat istirahat. Anda tidak dapat menonaktifkan enkripsi ini atau mengenkripsi hanya beberapa item dalam sebuah cluster.
- AWS Backup mengenkripsi cadangan Anda dan cluster apa pun yang dipulihkan dari cadangan ini. Anda dapat mengenkripsi data cadangan Anda dengan AWS Backup menggunakan kunci yang AWS dimiliki atau kunci yang dikelola pelanggan.
- Status perlindungan data berikut diaktifkan untuk Aurora DSQL:
  - Data saat istirahat - Aurora DSQL mengenkripsi semua data statis pada media penyimpanan persisten
  - Data dalam perjalanan - Aurora DSQL mengenkripsi semua komunikasi menggunakan Transport Layer Security (TLS) secara default
- Saat Anda beralih ke kunci yang berbeda, kami sarankan agar Anda tetap mengaktifkan kunci asli hingga transisi selesai. AWS membutuhkan kunci asli untuk mendekripsi data sebelum mengenkripsi data Anda dengan kunci baru. Prosesnya selesai ketika cluster `encryptionStatus` berada `ENABLED` dan Anda melihat kunci `kmsKeyArn` yang dikelola pelanggan baru.
- Saat Anda menonaktifkan Kunci Terkelola Pelanggan atau mencabut akses Aurora DSQL untuk menggunakan kunci Anda, klaster Anda akan masuk ke status. `IDLE`
- API DSQL Amazon Aurora AWS Management Console dan Amazon menggunakan istilah yang berbeda untuk jenis enkripsi:
  - AWS Konsol — Di konsol, Anda akan melihat `KMS` saat menggunakan kunci yang dikelola Pelanggan dan `DEFAULT` saat menggunakan kunci Kunci milik AWS.
  - API — Amazon Aurora DSQL API digunakan `CUSTOMER_MANAGED_KMS_KEY` untuk kunci yang dikelola pelanggan, dan untuk. `AWS_OWNED_KMS_KEY` Kunci milik AWS
- Jika Anda tidak menentukan kunci enkripsi selama pembuatan klaster, Aurora DSQL secara otomatis mengenkripsi data Anda menggunakan file. Kunci milik AWS

- Anda dapat beralih antara kunci yang dikelola Pelanggan Kunci milik AWS dan kapan saja. Buat perubahan ini menggunakan AWS Management Console, AWS CLI, atau Amazon Aurora DSQL API.

Manajemen identitas dan akses untuk Aurora DSQL

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 diberi wewenang (memiliki izin) untuk menggunakan sumber daya Aurora DSQL. IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik

Audiens

Bagaimana Anda menggunakan AWS Identity and Access Management (IAM) berbeda, tergantung pada pekerjaan yang Anda lakukan di Aurora DSQL.

Pengguna layanan — Jika Anda menggunakan layanan Aurora DSQL untuk melakukan pekerjaan Anda, maka administrator Anda memberi Anda kredensi dan izin yang Anda butuhkan. Saat Anda menggunakan lebih banyak fitur Aurora DSQL untuk melakukan pekerjaan Anda, Anda mungkin memerlukan izin tambahan. Memahami cara akses dikelola dapat membantu Anda meminta izin yang tepat dari administrator Anda. Jika Anda tidak dapat mengakses fitur di Aurora DSQL, lihat. Memecahkan masalah identitas dan akses Amazon Aurora DSQL

Administrator layanan — Jika Anda bertanggung jawab atas sumber daya Aurora DSQL di perusahaan Anda, Anda mungkin memiliki akses penuh ke Aurora DSQL. Tugas Anda adalah menentukan fitur dan sumber daya Aurora DSQL mana yang harus diakses pengguna layanan Anda. Kemudian, Anda harus mengirimkan permintaan kepada administrator IAM untuk mengubah izin

pengguna layanan Anda. Tinjau informasi di halaman ini untuk memahami konsep dasar IAM. Untuk mempelajari lebih lanjut tentang bagaimana perusahaan Anda dapat menggunakan IAM dengan Aurora DSQL, lihat. Bagaimana Amazon Aurora DSQL bekerja dengan IAM

Administrator IAM — Jika Anda seorang administrator IAM, Anda mungkin ingin mempelajari detail tentang bagaimana Anda dapat menulis kebijakan untuk mengelola akses ke Aurora DSQL. Untuk melihat contoh kebijakan berbasis identitas Aurora DSQL yang dapat Anda gunakan di IAM, lihat. Contoh kebijakan berbasis identitas untuk Amazon Aurora DSQL

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 kredensyal 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 terfederasi, 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. Guna mengetahui informasi selengkapnya tentang penggunaan metode yang disarankan untuk menandatangani permintaan sendiri, lihat AWS Signature Version 4 untuk permintaan API dalam 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 Autentikasi multi-faktor AWS di IAM dalam Panduan Pengguna IAM.

Akun AWS pengguna root

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. 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 gabungan

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 Apakah 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, kami merekomendasikan untuk mengandalkan kredensial sementara, bukan membuat pengguna IAM yang memiliki kredensial jangka panjang seperti kata sandi dan kunci akses. Namun, jika Anda memiliki kasus penggunaan tertentu yang memerlukan kredensial jangka panjang dengan pengguna IAM, kami merekomendasikan Anda merotasi kunci akses. Untuk informasi selengkapnya, lihat Merotasi kunci akses secara teratur untuk kasus penggunaan yang memerlukan kredensial jangka panjang dalam Panduan Pengguna IAM.

Grup IAM adalah identitas yang menentukan sekumpulan pengguna IAM. Anda tidak dapat masuk sebagai grup. Anda dapat menggunakan grup untuk menentukan izin bagi beberapa pengguna sekaligus. Grup mempermudah manajemen izin untuk sejumlah besar pengguna sekaligus. Misalnya, Anda dapat meminta kelompok untuk menyebutkan 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 Kasus penggunaan untuk pengguna IAM dalam Panduan Pengguna IAM.

Peran IAM

Peran IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus. Peran ini mirip dengan pengguna IAM, tetapi tidak terkait dengan orang tertentu. Untuk mengambil peran IAM sementara AWS Management Console, Anda dapat beralih dari pengguna ke peran IAM (konsol). 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 Metode untuk mengambil peran dalam Panduan Pengguna IAM.

Peran IAM dengan kredensial sementara berguna dalam situasi berikut:

- Akses pengguna terfederasi – Untuk menetapkan izin ke identitas terfederasi, Anda membuat peran dan menentukan izin untuk peran tersebut. Ketika identitas terfederasi mengautentikasi, identitas tersebut terhubung dengan peran dan diberi izin yang ditentukan oleh peran. Untuk informasi tentang peran untuk federasi, lihat Buat peran untuk penyedia identitas pihak ketiga dalam Panduan Pengguna IAM. Jika menggunakan Pusat Identitas IAM, Anda harus mengonfigurasi set izin. Untuk mengontrol apa yang dapat diakses identitas Anda setelah identitas tersebut diautentikasi, Pusat Identitas IAM akan mengorelasikan set izin ke peran dalam IAM. Untuk informasi tentang set izin, lihat Set izin dalam Panduan Pengguna AWS IAM Identity Center .
- Izin pengguna IAM sementara – Pengguna atau peran IAM dapat mengambil peran IAM guna mendapatkan berbagai izin secara sementara untuk tugas tertentu.
- 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 peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.

- Akses lintas layanan — Beberapa Layanan AWS menggunakan fitur lain Layanan AWS. Misalnya, saat 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 (FAS) — Saat Anda menggunakan pengguna atau peran IAM untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Ketika Anda menggunakan beberapa layanan, Anda mungkin melakukan sebuah tindakan yang kemudian menginisiasi 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 ketika mengajukan permintaan FAS, lihat Sesi akses maju.
  - Peran layanan – Peran layanan adalah peran IAM yang dijalankan oleh layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Buat sebuah peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.
  - Peran terkait layanan — Peran terkait layanan adalah jenis peran layanan yang ditautkan ke peran layanan. Layanan AWS Layanan tersebut dapat menjalankan 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 EC2 instance dan membuat AWS CLI atau AWS permintaan API. Ini lebih baik untuk menyimpan kunci akses dalam EC2 instance. Untuk menetapkan AWS peran ke EC2 instance dan membuatnya tersedia untuk semua aplikasinya, Anda membuat profil instans yang dilampirkan ke instance. Profil instance berisi peran dan memungkinkan program yang berjalan pada EC2 instance untuk mendapatkan kredensi sementara. Untuk informasi selengkapnya, lihat Menggunakan peran IAM untuk memberikan izin ke aplikasi yang berjalan di EC2 instans Amazon di Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan adalah objek AWS yang, ketika dikaitkan dengan identitas atau sumber daya, menentukan izinnya. AWS mengevaluasi kebijakan ini ketika prinsipal (pengguna, pengguna root, atau sesi peran) membuat permintaan. Izin dalam kebijakan menentukan apakah permintaan diizinkan atau ditolak. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang struktur dan isi dokumen kebijakan JSON, lihat Gambaran umum kebijakan JSON dalam Panduan Pengguna IAM.

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Artinya, prinsipal manakah yang dapat melakukan tindakan pada sumber daya apa, dan dengan kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Untuk memberikan izin kepada pengguna untuk melakukan tindakan di sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat mengambil peran.

Kebijakan IAM mendefinisikan izin untuk suatu tindakan terlepas dari metode yang Anda gunakan untuk melakukan operasinya. Misalnya, anggaplah 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 sebuah identitas, seperti pengguna IAM, grup pengguna IAM, atau peran IAM. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan terkelola pelanggan dalam Panduan Pengguna IAM.

Kebijakan berbasis identitas dapat dikategorikan lebih lanjut sebagai kebijakan inline atau kebijakan yang dikelola. Kebijakan inline disematkan langsung ke satu pengguna, grup, atau peran. Kebijakan terkelola adalah kebijakan mandiri yang dapat Anda lampirkan ke beberapa pengguna, grup, dan peran dalam. Akun AWS Kebijakan AWS terkelola mencakup kebijakan terkelola dan kebijakan yang dikelola pelanggan. Untuk mempelajari cara memilih antara kebijakan yang dikelola atau kebijakan inline, lihat Pilih antara kebijakan yang dikelola dan kebijakan inline dalam Panduan Pengguna IAM.

Kebijakan berbasis sumber daya

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 tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat dilakukan oleh prinsipal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus menentukan prinsipal dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Kebijakan berbasis sumber daya merupakan kebijakan inline yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan AWS terkelola dari IAM dalam kebijakan berbasis sumber daya.

Daftar kontrol akses (ACLs)

Access control lists (ACLs) mengontrol prinsipal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACLs mirip dengan kebijakan berbasis sumber daya, meskipun mereka tidak menggunakan format dokumen kebijakan JSON.

Amazon S3, AWS WAF, dan Amazon VPC adalah contoh layanan yang mendukung. ACLs Untuk mempelajari selengkapnya ACLs, lihat Ringkasan daftar kontrol akses (ACL) di Panduan Pengembang Layanan Penyimpanan Sederhana Amazon.

Jenis-jenis kebijakan lain

AWS mendukung jenis kebijakan tambahan yang kurang umum. Jenis-jenis kebijakan ini dapat mengatur izin maksimum yang diberikan kepada Anda oleh jenis kebijakan yang lebih umum.

- Batasan izin – Batasan izin adalah fitur lanjutan tempat Anda mengatur izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas ke entitas IAM (pengguna IAM atau peran IAM). Anda dapat menetapkan batasan izin untuk suatu entitas. Izin yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas milik entitas dan batasan izinnya. Kebijakan berbasis sumber daya yang menentukan pengguna atau peran dalam bidang `Principal` tidak dibatasi oleh batasan izin. Penolakan eksplisit dalam salah satu kebijakan ini akan menggantikan pemberian izin. Untuk informasi selengkapnya tentang batasan izin, lihat Batasan izin untuk entitas IAM dalam Panduan Pengguna IAM.
- Kebijakan kontrol layanan (SCPs) — SCPs adalah kebijakan JSON yang menentukan izin maksimum untuk organisasi atau unit organisasi (OU) di. AWS Organizations AWS Organizations

adalah layanan untuk mengelompokkan dan mengelola secara terpusat beberapa Akun AWS yang dimiliki bisnis Anda. Jika Anda mengaktifkan semua fitur dalam suatu organisasi, maka Anda dapat menerapkan kebijakan kontrol layanan (SCPs) 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 Organizations dan SCPs, lihat Kebijakan kontrol layanan di Panduan AWS Organizations Pengguna.

- Kebijakan kontrol sumber daya (RCPs) — RCPs adalah kebijakan JSON yang dapat Anda gunakan untuk menetapkan izin maksimum yang tersedia untuk sumber daya di akun Anda tanpa memperbarui kebijakan IAM yang dilampirkan ke setiap sumber daya yang Anda miliki. RCP membatasi izin untuk sumber daya di akun anggota dan dapat memengaruhi izin efektif untuk identitas, termasuk Pengguna root akun AWS, terlepas dari apakah itu milik organisasi Anda. Untuk informasi selengkapnya tentang Organizations dan RCPs, termasuk daftar dukungan Layanan AWS tersebut RCPs, lihat Kebijakan kontrol sumber daya (RCPs) di Panduan AWS Organizations Pengguna.
- Kebijakan sesi – Kebijakan sesi adalah kebijakan lanjutan yang Anda berikan sebagai parameter ketika Anda membuat sesi sementara secara programatis untuk peran atau pengguna terfederasi. Izin sesi yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas pengguna atau peran dan kebijakan sesi. Izin juga bisa datang dari kebijakan berbasis sumber daya. Penolakan eksplisit dalam salah satu kebijakan ini akan menggantikan pemberian izin. Untuk informasi selengkapnya, lihat Kebijakan sesi dalam Panduan Pengguna IAM.

Berbagai jenis kebijakan

Ketika beberapa jenis kebijakan berlaku pada suatu 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.

Bagaimana Amazon Aurora DSQL bekerja dengan IAM

Sebelum Anda menggunakan IAM untuk mengelola akses ke Aurora DSQL, pelajari fitur IAM apa yang tersedia untuk digunakan dengan Aurora DSQL.

Fitur IAM yang dapat Anda gunakan dengan Amazon Aurora DSQL

Fitur IAM	Dukungan Aurora DSQL
Kebijakan berbasis identitas	Ya
Kebijakan berbasis sumber daya	Tidak
Tindakan kebijakan	Ya
Sumber daya kebijakan	Ya
Kunci kondisi kebijakan	Ya
ACLs	Tidak
ABAC (tanda dalam kebijakan)	Ya
Kredensial sementara	Ya
Izin principal	Ya
Peran layanan	Ya
Peran terkait layanan	Ya

Untuk mendapatkan tampilan tingkat tinggi tentang bagaimana Aurora DSQL dan layanan AWS lainnya bekerja dengan sebagian besar fitur IAM, AWS lihat layanan yang bekerja dengan IAM di Panduan Pengguna IAM.

Kebijakan berbasis identitas untuk Aurora DSQL

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 oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan terkelola pelanggan dalam Panduan Pengguna IAM.

Dengan kebijakan berbasis identitas IAM, Anda dapat menentukan secara spesifik apakah tindakan dan sumber daya diizinkan atau ditolak, serta kondisi yang menjadi dasar dikabulkan atau ditolaknya tindakan tersebut. Anda tidak dapat menentukan secara spesifik prinsipal dalam sebuah kebijakan berbasis identitas karena prinsipal berlaku bagi pengguna atau peran yang melekat kepadanya. Untuk mempelajari semua elemen yang dapat Anda gunakan dalam kebijakan JSON, lihat Referensi elemen kebijakan JSON IAM dalam Panduan Pengguna IAM.

Contoh kebijakan berbasis identitas untuk Aurora DSQL

Untuk melihat contoh kebijakan berbasis identitas Aurora DSQL, lihat. Contoh kebijakan berbasis identitas untuk Amazon Aurora DSQL

Kebijakan berbasis sumber daya dalam Aurora DSQL

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 tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat dilakukan oleh prinsipal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus menentukan prinsipal dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan secara spesifik seluruh akun atau entitas IAM di akun lain sebagai prinsipal dalam kebijakan berbasis sumber daya. Menambahkan prinsipal akun silang ke kebijakan berbasis sumber daya hanya setengah 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. Mereka memberikan izin dengan melampirkan kebijakan berbasis identitas kepada entitas. Namun, jika kebijakan berbasis sumber daya memberikan akses ke principal dalam akun yang sama, tidak diperlukan kebijakan berbasis identitas tambahan. Untuk informasi selengkapnya, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.

Tindakan kebijakan untuk Aurora DSQL

Mendukung tindakan kebijakan: Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Artinya, prinsipal manakah yang dapat melakukan tindakan pada sumber daya apa, dan dengan 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.

Sertakan tindakan dalam kebijakan untuk memberikan izin untuk melakukan operasi terkait.

Untuk melihat daftar tindakan Aurora DSQL, lihat Tindakan yang Ditentukan oleh Amazon Aurora DSQL di Referensi Otorisasi Layanan.

Tindakan kebijakan di Aurora DSQL menggunakan awalan berikut sebelum tindakan:

```
dsql
```

Untuk menetapkan secara spesifik beberapa tindakan dalam satu pernyataan, pisahkan tindakan tersebut dengan koma.

```
"Action": [
      "dsql:action1",
      "dsql:action2"
]
```

Untuk melihat contoh kebijakan berbasis identitas Aurora DSQL, lihat. Contoh kebijakan berbasis identitas untuk Amazon Aurora DSQL

Sumber daya kebijakan untuk Aurora DSQL

Mendukung sumber daya kebijakan: Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Artinya, prinsipal manakah yang dapat melakukan tindakan pada sumber daya apa, dan dengan kondisi apa.

Elemen kebijakan JSON `Resource` menentukan 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 menunjukkan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

```
"Resource": "*"
```

Untuk melihat daftar jenis sumber daya Aurora DSQL dan jenisnya ARNs, lihat Sumber Daya yang Ditentukan oleh Amazon Aurora DSQL di Referensi Otorisasi Layanan. Untuk mempelajari tindakan mana yang dapat Anda tentukan ARN dari setiap sumber daya, lihat Tindakan yang Ditentukan oleh Amazon Aurora DSQL.

Untuk melihat contoh kebijakan berbasis identitas Aurora DSQL, lihat. Contoh kebijakan berbasis identitas untuk Amazon Aurora DSQL

Kunci kondisi kebijakan untuk Aurora DSQL

Mendukung kunci kondisi kebijakan khusus layanan: Yes

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Artinya, prinsipal manakah yang dapat melakukan tindakan pada sumber daya apa, dan dengan kondisi apa.

Elemen `Condition` (atau blok `Condition`) akan memungkinkan Anda menentukan kondisi yang menjadi dasar suatu pernyataan berlaku. Elemen `Condition` bersifat opsional. Anda dapat membuat ekspresi bersyarat 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 sebuah pernyataan, atau beberapa kunci dalam elemen `Condition` tunggal, maka AWS akan mengevaluasinya 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. Sebagai contoh, Anda dapat memberikan izin kepada pengguna IAM untuk mengakses sumber daya hanya jika izin tersebut mempunyai tanda yang sesuai dengan nama pengguna IAM mereka. Untuk informasi selengkapnya, lihat Elemen kebijakan IAM: variabel dan tanda dalam 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.

Untuk melihat daftar kunci kondisi Aurora DSQL, lihat Kunci kondisi untuk Amazon Aurora DSQL di Referensi Otorisasi Layanan. Untuk mempelajari tindakan dan sumber daya yang dapat Anda gunakan kunci kondisi, lihat Tindakan yang ditentukan oleh Amazon Aurora DSQL.

Untuk melihat contoh kebijakan berbasis identitas Aurora DSQL, lihat. Contoh kebijakan berbasis identitas untuk Amazon Aurora DSQL

ACLs di Aurora DSQL

Mendukung ACLs: Tidak

Access control lists (ACLs) mengontrol prinsipal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACLs mirip dengan kebijakan berbasis sumber daya, meskipun mereka tidak menggunakan format dokumen kebijakan JSON.

ABAC dengan Aurora DSQL

Mendukung ABAC (tanda 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. Penandaan ke entitas dan sumber daya adalah langkah pertama dari ABAC. Kemudian rancanglah kebijakan ABAC untuk mengizinkan operasi ketika tanda milik prinsipal cocok dengan tanda yang ada di sumber daya yang ingin diakses.

ABAC sangat berguna di lingkungan yang berkembang dengan cepat dan berguna di situasi saat manajemen kebijakan menjadi rumit.

Untuk mengendalikan akses berdasarkan tanda, berikan informasi tentang tanda di elemen kondisi dari kebijakan 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 untuk hanya beberapa jenis sumber daya, nilainya adalah Parsial.

Untuk informasi selengkapnya tentang ABAC, lihat Tentukan izin dengan otorisasi ABAC dalam Panduan Pengguna IAM. Untuk melihat tutorial yang menguraikan langkah-langkah pengaturan ABAC, lihat Menggunakan kontrol akses berbasis atribut (ABAC) dalam Panduan Pengguna IAM.

Menggunakan kredensil sementara dengan Aurora DSQL

Mendukung kredensial sementara: Ya

Beberapa Layanan AWS tidak berfungsi saat Anda masuk menggunakan kredensil 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 kredensyal 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 kredensil sementara. Anda juga akan secara otomatis membuat kredensial sementara ketika Anda masuk ke konsol sebagai seorang pengguna lalu beralih peran. Untuk informasi selengkapnya tentang peralihan peran, lihat Beralih dari pengguna ke peran IAM (konsol) dalam Panduan Pengguna IAM.

Anda dapat membuat kredensyal sementara secara manual menggunakan API AWS CLI atau AWS . Anda kemudian dapat menggunakan kredensil sementara tersebut untuk mengakses. AWS AWS merekomendasikan agar Anda secara dinamis menghasilkan kredensyal sementara alih-alih menggunakan kunci akses jangka panjang. Untuk informasi selengkapnya, lihat Kredensial keamanan sementara di IAM.

Izin utama lintas layanan untuk Aurora DSQL

Mendukung sesi akses maju (FAS): Ya

Saat Anda menggunakan pengguna atau peran IAM untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Ketika Anda menggunakan beberapa layanan, Anda mungkin melakukan sebuah tindakan yang kemudian menginisiasi 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 ketika mengajukan permintaan FAS, lihat Sesi akses maju.

Peran layanan untuk Aurora DSQL

Mendukung peran layanan: Ya

Peran layanan adalah peran IAM yang diambil oleh sebuah layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Buat sebuah peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.

> **Warning**
>
> Mengubah izin untuk peran layanan dapat merusak fungsionalitas Aurora DSQL. Edit peran layanan hanya ketika Aurora DSQL memberikan panduan untuk melakukannya.

Peran terkait layanan untuk Aurora DSQL

Mendukung peran terkait layanan: Ya

Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan tersebut dapat menjalankan 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 membuat atau mengelola peran terkait layanan untuk Aurora DSQL, lihat. Menggunakan peran terkait layanan di Aurora DSQL

Contoh kebijakan berbasis identitas untuk Amazon Aurora DSQL

Secara default, pengguna dan peran tidak memiliki izin untuk membuat atau memodifikasi sumber daya Aurora DSQL. Mereka juga tidak dapat melakukan tugas dengan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau AWS API. Untuk memberikan izin kepada pengguna untuk melakukan tindakan di sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat mengambil peran.

Untuk mempelajari cara membuat kebijakan berbasis identitas IAM dengan menggunakan contoh dokumen kebijakan JSON ini, lihat Membuat kebijakan IAM (konsol) di Panduan Pengguna IAM.

Untuk detail tentang tindakan dan jenis sumber daya yang ditentukan oleh Aurora DSQL, termasuk format ARNs untuk setiap jenis sumber daya, lihat Tindakan, Sumber Daya, dan Kunci Kondisi untuk Amazon Aurora DSQL di Referensi Otorisasi Layanan.

Topik

- Praktik terbaik kebijakan
- Menggunakan konsol Aurora DSQL
- Mengizinkan pengguna melihat izin mereka sendiri

Praktik terbaik kebijakan

Kebijakan berbasis identitas menentukan apakah seseorang dapat membuat, mengakses, atau menghapus sumber daya Aurora DSQL di akun Anda. Tindakan ini membuat Akun AWS Anda dikenai biaya. 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 tugas dalam 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 melakukannya dengan mendefinisikan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, yang juga dikenal sebagai izin dengan hak akses paling rendah. Untuk informasi selengkapnya tentang cara menggunakan IAM untuk mengajukan izin, lihat Kebijakan dan izin dalam IAM dalam Panduan Pengguna IAM.
- Gunakan kondisi dalam kebijakan IAM untuk membatasi akses lebih lanjut – Anda dapat menambahkan suatu 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: Kondisi dalam Panduan Pengguna IAM.

- Gunakan IAM Access Analyzer untuk memvalidasi kebijakan IAM Anda untuk memastikan izin yang aman dan fungsional – 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 dengan IAM Access Analyzer dalam 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 meminta MFA ketika operasi API dipanggil, tambahkan kondisi MFA pada kebijakan Anda. Untuk informasi selengkapnya, lihat Amankan akses API dengan MFA dalam Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang praktik terbaik dalam IAM, lihat Praktik terbaik keamanan di IAM dalam Panduan Pengguna IAM.

Menggunakan konsol Aurora DSQL

Untuk mengakses konsol Amazon Aurora DSQL, Anda harus memiliki set izin minimum. Izin ini harus memungkinkan Anda untuk daftar dan melihat rincian tentang sumber daya Aurora DSQL 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 gantinya, izinkan akses hanya ke tindakan yang sesuai dengan operasi API yang coba mereka lakukan.

Untuk memastikan bahwa pengguna dan peran masih dapat menggunakan konsol Aurora DSQL, lampirkan juga Aurora `AmazonAuroraDSQLConsoleFullAccess` DSQL atau kebijakan terkelola ke entitas. `AmazonAuroraDSQLReadOnlyAccess` AWS Untuk informasi selengkapnya, lihat Menambah izin untuk pengguna dalam Panduan Pengguna IAM.

Mengizinkan pengguna melihat izin mereka sendiri

Contoh ini menunjukkan cara membuat kebijakan yang mengizinkan 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": "*"
        }
    ]
}
```

Memecahkan masalah identitas dan akses Amazon Aurora DSQL

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda temui saat bekerja dengan Aurora DSQL dan IAM.

Topik

Saya tidak berwenang untuk melakukan tindakan di Aurora DSQL

Jika Anda menerima pesan kesalahan bahwa Anda tidak memiliki otorisasi untuk melakukan tindakan, kebijakan Anda harus diperbarui agar Anda dapat melakukan tindakan tersebut.

Contoh kesalahan berikut terjadi ketika `mateojackson` mencoba menggunakan konsol untuk melihat detail tentang *`my-dsql-cluster`* sumber daya tetapi tidak memiliki *`GetCluster`* izin.

```
User: iam:::user/mateojackson is not authorized to perform: GetCluster on resource: my-dsql-cluster
```

Dalam hal ini, kebijakan untuk pengguna `mateojackson` harus diperbarui untuk mengizinkan akses ke sumber daya *`my-dsql-cluster`* dengan menggunakan tindakan *`GetCluster`*.

Jika Anda membutuhkan bantuan, hubungi administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan bahwa Anda tidak berwenang untuk melakukan `iam:PassRole` tindakan, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran ke Aurora DSQL.

Beberapa Layanan AWS memungkinkan Anda untuk meneruskan peran yang ada ke layanan tersebut alih-alih membuat peran layanan baru atau peran terkait layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh kesalahan berikut terjadi ketika pengguna IAM bernama `marymajor` mencoba menggunakan konsol untuk melakukan tindakan di Aurora DSQL. Namun, tindakan tersebut memerlukan layanan untuk mendapatkan izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut pada layanan.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dalam kasus ini, kebijakan Mary harus diperbarui agar dia mendapatkan izin untuk melakukan tindakan `iam:PassRole` tersebut.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya ingin mengizinkan orang di luar saya Akun AWS untuk mengakses sumber daya Aurora DSQL saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau orang-orang 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 (ACLs), Anda dapat menggunakan kebijakan tersebut untuk memberi orang akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa referensi berikut:

- Untuk mempelajari apakah Aurora DSQL mendukung fitur-fitur ini, lihat. Bagaimana Amazon Aurora DSQL bekerja dengan IAM
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda di seluruh sumber daya Akun AWS yang Anda miliki, lihat Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki di Panduan Pengguna IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda kepada pihak ketiga Akun AWS, lihat Menyediakan akses yang Akun AWS dimiliki oleh pihak ketiga dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat Menyediakan akses ke pengguna terautentikasi eksternal (federasi identitas) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara menggunakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM di Panduan Pengguna IAM.

Menggunakan peran terkait layanan di Aurora DSQL

Aurora DSQL menggunakan peran terkait layanan AWS Identity and Access Management (IAM). Peran terkait layanan adalah jenis peran IAM unik yang ditautkan langsung ke Aurora DSQL. Peran terkait layanan telah ditentukan sebelumnya oleh Aurora DSQL dan mencakup semua izin yang diperlukan layanan untuk memanggil atas Layanan AWS nama cluster Aurora DSQL Anda.

Peran terkait layanan membuat proses penyiapan lebih mudah karena Anda tidak perlu menambahkan izin yang diperlukan secara manual untuk menggunakan Aurora DSQL. Saat Anda membuat klaster, Aurora DSQL secara otomatis membuat peran terkait layanan untuk Anda. Anda dapat menghapus peran terkait layanan hanya setelah Anda menghapus semua cluster Anda. Ini melindungi sumber daya Aurora DSQL Anda karena Anda tidak dapat secara tidak sengaja menghapus izin yang diperlukan untuk mengakses sumber daya.

Untuk informasi tentang layanan lain yang mendukung peran terkait layanan, lihat layanan Layanan AWS yang berfungsi dengan IAM dan cari layanan yang memiliki Ya di kolom Peran Tertaut Layanan. Pilih Ya dengan sebuah tautan untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Peran terkait layanan tersedia di semua Wilayah Aurora DSQL yang didukung.

Izin peran terkait layanan untuk Aurora DSQL

Aurora DSQL menggunakan peran terkait layanan bernama — Memungkinkan `AWSServiceRoleForAuroraDsql` Amazon Aurora DSQL membuat dan mengelola sumber daya atas nama Anda. AWS Peran terkait layanan ini dilampirkan ke kebijakan terkelola berikut ini: AuroraDsqlServiceLinkedRolePolicy.

> **Note**
>
> Anda harus mengonfigurasi izin agar entitas IAM (seperti pengguna, grup, atau peran) dapat membuat, mengedit, atau menghapus peran terkait layanan. Anda mungkin menemukan pesan kesalahan berikut:`You don't have the permissions to create an Amazon Aurora DSQL service-linked role`. Jika Anda melihat pesan ini, pastikan bahwa Anda mengaktifkan izin berikut:
>
> JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": ["dsql:CreateCluster"],
            "Resource": [
                "arn:aws:dsql:us-east-1:*:cluster/*",
                "arn:aws:dsql:us-east-2:*:cluster/*"
            ],
```

```
            "Effect": "Allow"
        }
    ]
}
```

Untuk informasi selengkapnya, lihat Izin peran terkait layanan.

Buat peran tertaut layanan

Anda tidak perlu membuat peran terkait DSQLService LinkedRolePolicy layanan Aurora secara manual. Aurora DSQL menciptakan peran terkait layanan untuk Anda. Jika peran DSQLService LinkedRolePolicy terkait layanan Aurora telah dihapus dari akun Anda, Aurora DSQL akan membuat peran tersebut saat Anda membuat cluster Aurora DSQL baru.

Edit peran tertaut layanan

Aurora DSQL tidak memungkinkan Anda untuk mengedit peran terkait layanan Aurora. DSQLService LinkedRolePolicy Setelah membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin mereferensikan peran tersebut. Namun, Anda dapat mengedit deskripsi peran menggunakan konsol IAM, AWS Command Line Interface (AWS CLI), atau IAM API.

Hapus peran tertaut layanan

Jika Anda tidak perlu lagi menggunakan fitur atau layanan yang memerlukan peran terkait layanan, sebaiknya hapus peran tersebut. Dengan begitu, Anda tidak memiliki entitas yang tidak terpakai yang tidak dipantau atau dipelihara secara aktif.

Sebelum Anda dapat menghapus peran terkait layanan untuk akun, Anda harus menghapus klaster apa pun di akun tersebut.

Anda dapat menggunakan konsol IAM, API IAM AWS CLI, atau IAM untuk menghapus peran terkait layanan. Untuk informasi selengkapnya, lihat Membuat peran terkait layanan di Panduan Pengguna IAM.

Wilayah yang Didukung untuk peran terkait layanan Aurora DSQL

Aurora DSQL mendukung penggunaan peran terkait layanan di semua Wilayah tempat layanan tersedia. Untuk informasi selengkapnya, lihat AWS Wilayah dan titik akhir.

Menggunakan kunci kondisi IAM dengan Amazon Aurora DSQL

Saat Anda memberikan izin di Aurora DSQL, Anda dapat menentukan kondisi yang menentukan bagaimana kebijakan izin diterapkan. Berikut ini adalah contoh bagaimana Anda dapat menggunakan kunci kondisi dalam kebijakan izin Aurora DSQL.

Contoh 1: Berikan izin untuk membuat klaster di tempat tertentu Wilayah AWS

Kebijakan berikut memberikan izin untuk membuat cluster di Wilayah AS Timur (Virginia N.) dan Timur AS (Ohio). Kebijakan ini menggunakan ARN sumber daya untuk membatasi Wilayah yang diizinkan, sehingga Aurora DSQL hanya dapat membuat klaster hanya jika ARN tersebut ditentukan di bagian kebijakan. `Resource`

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": ["dsql:CreateCluster"],
            "Resource": [
                "arn:aws:dsql:us-east-1:*:cluster/*",
                "arn:aws:dsql:us-east-2:*:cluster/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Contoh 2: Berikan izin untuk membuat cluster Multi-wilayah di s tertentu Wilayah AWS

Kebijakan berikut memberikan izin untuk membuat klaster Multi-wilayah di Wilayah AS Timur (Virginia N.) dan Timur AS (Ohio). Kebijakan ini menggunakan ARN sumber daya untuk membatasi Wilayah yang diizinkan, sehingga Aurora DSQL dapat membuat klaster Multi-wilayah hanya jika ARN ini ditentukan di bagian kebijakan. `Resource` Perhatikan bahwa membuat klaster Multi-wilayah juga

memerlukan `AddPeerCluster` izin`PutMultiRegionProperties`,`PutWitnessRegion`, dan di setiap Wilayah yang ditentukan.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
          "dsql:CreateCluster",
          "dsql:PutMultiRegionProperties",
          "dsql:PutWitnessRegion",
          "dsql:AddPeerCluster"
        ],
        "Resource": [
           "arn:aws:dsql:us-east-1:123456789012:cluster/*",
           "arn:aws:dsql:us-east-2:123456789012:cluster/*"
        ]
      }
    ]
}
```

Contoh 3: Berikan izin untuk membuat klaster Multi-wilayah dengan Wilayah saksi tertentu

Kebijakan berikut menggunakan kunci `dsql:WitnessRegion` kondisi Aurora DSQL dan memungkinkan pengguna membuat klaster Multi-wilayah dengan Wilayah saksi di AS Barat (Oregon). Jika Anda tidak menentukan `dsql:WitnessRegion` kondisinya, Anda dapat menggunakan Wilayah mana pun sebagai Wilayah saksi.

JSON

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "dsql:CreateCluster",
                "dsql:PutMultiRegionProperties",
                "dsql:AddPeerCluster"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dsql:PutWitnessRegion"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
            "Condition": {
                "StringEquals": {
                    "dsql:WitnessRegion": [
                        "us-west-2"
                    ]
                }
            }
        }
    ]
}
```

Respon insiden di Amazon Aurora DSQL

Keamanan adalah prioritas tertinggi di AWS. Sebagai bagian dari model tanggung jawab bersama AWS Cloud, AWS mengelola pusat data, jaringan, dan arsitektur perangkat lunak yang memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan. AWS bertanggung jawab atas setiap respons insiden sehubungan dengan layanan Amazon Aurora DSQL itu sendiri. Selain itu, sebagai AWS pelanggan, Anda berbagi tanggung jawab untuk menjaga keamanan di cloud. Ini berarti Anda mengontrol keamanan yang Anda pilih untuk diterapkan dari AWS alat dan fitur yang dapat Anda akses. Selain itu, Anda bertanggung jawab atas respons insiden di pihak Anda dari model tanggung jawab bersama.

Dengan menetapkan garis dasar keamanan yang memenuhi tujuan aplikasi Anda yang berjalan di cloud, Anda dapat mendeteksi penyimpangan yang dapat Anda tanggapi. Untuk membantu Anda memahami dampak respons insiden dan pilihan Anda terhadap tujuan perusahaan Anda, kami mendorong Anda untuk meninjau sumber daya berikut:

- AWS Panduan Respons Insiden Keamanan
- AWS Praktik Terbaik untuk Keamanan, Identitas, dan Kepatuhan
- Perspektif Keamanan dari AWS whitepaper Cloud Adoption Framework (CAF)

Amazon GuardDuty adalah layanan deteksi ancaman terkelola yang terus memantau perilaku berbahaya atau tidak sah untuk membantu pelanggan melindungi Akun AWS dan beban kerja serta mengidentifikasi aktivitas mencurigakan yang berpotensi sebelum meningkat menjadi insiden. Ini memantau aktivitas seperti panggilan API yang tidak biasa atau penerapan yang berpotensi tidak sah yang menunjukkan kemungkinan kompromi akun atau sumber daya atau pengintaian oleh aktor jahat. Misalnya, Amazon GuardDuty dapat mendeteksi aktivitas mencurigakan di Amazon Aurora APIs DSQL, seperti pengguna yang masuk dari lokasi baru dan membuat cluster baru.

Validasi kepatuhan untuk Amazon Aurora DSQL

Untuk mempelajari apakah an Layanan AWS berada dalam lingkup program kepatuhan tertentu, lihat Layanan AWS di Lingkup oleh Program Kepatuhan Layanan AWS dan pilih program kepatuhan yang Anda minati. Untuk informasi umum, lihat Program AWS Kepatuhan Program 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 Layanan AWS ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, dan hukum dan peraturan yang berlaku. AWS menyediakan sumber daya berikut untuk membantu kepatuhan:

- Kepatuhan dan Tata Kelola Keamanan – Panduan implementasi solusi ini membahas pertimbangan arsitektur serta memberikan langkah-langkah untuk menerapkan fitur keamanan dan kepatuhan.
- Referensi Layanan yang Memenuhi Syarat HIPAA — Daftar layanan yang memenuhi syarat HIPAA. Tidak semua memenuhi Layanan AWS syarat HIPAA.
- AWS Sumber Daya AWS — Kumpulan buku kerja dan panduan ini mungkin berlaku untuk industri dan lokasi Anda.
- AWS Panduan Kepatuhan Pelanggan - Memahami model tanggung jawab bersama melalui lensa kepatuhan. Panduan ini merangkum praktik terbaik untuk mengamankan Layanan AWS dan memetakan panduan untuk kontrol keamanan di berbagai kerangka kerja (termasuk Institut Standar dan Teknologi Nasional (NIST), Dewan Standar Keamanan Industri Kartu Pembayaran (PCI), dan Organisasi Internasional untuk Standardisasi (ISO)).

- Mengevaluasi Sumber Daya dengan Aturan dalam Panduan AWS Config Pengembang — AWS Config Layanan menilai seberapa baik konfigurasi sumber daya Anda mematuhi praktik internal, pedoman industri, dan peraturan.
- AWS Security Hub— Ini Layanan AWS memberikan pandangan komprehensif tentang keadaan keamanan Anda di dalamnya AWS. Security Hub menggunakan kontrol keamanan untuk sumber daya AWS Anda serta untuk memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik. Untuk daftar layanan dan kontrol yang didukung, lihat Referensi kontrol Security Hub.
- Amazon GuardDuty — Ini Layanan AWS mendeteksi potensi ancaman terhadap beban kerja Akun AWS, kontainer, dan data Anda dengan memantau lingkungan Anda untuk aktivitas yang mencurigakan dan berbahaya. GuardDuty dapat membantu Anda mengatasi berbagai persyaratan kepatuhan, seperti PCI DSS, dengan memenuhi persyaratan deteksi intrusi yang diamanatkan oleh kerangka kerja kepatuhan tertentu.
- AWS Audit ManagerIni Layanan AWS membantu Anda terus mengaudit AWS penggunaan Anda untuk menyederhanakan cara Anda mengelola risiko dan kepatuhan terhadap peraturan dan standar industri.

Ketahanan di Amazon Aurora DSQL

Infrastruktur AWS global dibangun di sekitar Wilayah AWS dan Availability Zones (AZ). Wilayah AWS menyediakan beberapa Availability Zone yang terpisah secara fisik dan terisolasi, yang terhubung dengan latensi rendah, throughput tinggi, dan jaringan yang sangat redundan. Dengan Zona Ketersediaan, Anda dapat merancang serta mengoperasikan aplikasi dan basis data yang secara otomatis melakukan fail over di antara zona tanpa gangguan. Zona Ketersediaan memiliki ketersediaan dan toleransi kesalahan yang lebih baik, dan dapat diskalakan dibandingkan infrastruktur pusat data tunggal atau multi tradisional. Aurora DSQL dirancang sedemikian rupa sehingga Anda dapat memanfaatkan infrastruktur AWS Regional sambil menyediakan ketersediaan database tertinggi. Secara default, kluster wilayah tunggal di Aurora DSQL memiliki ketersediaan Multi-AZ, memberikan toleransi terhadap kegagalan komponen utama dan gangguan infrastruktur yang dapat memengaruhi akses ke AZ penuh. Cluster Multi-Region memberikan semua manfaat dari ketahanan Multi-AZ sambil tetap menyediakan ketersediaan database yang sangat konsisten, bahkan dalam kasus di mana Wilayah AWS tidak dapat diakses oleh klien aplikasi.

Untuk informasi selengkapnya tentang Wilayah AWS dan Availability Zone, lihat Infrastruktur AWS Global.

Selain infrastruktur AWS global, Aurora DSQL menawarkan beberapa fitur untuk membantu mendukung ketahanan data dan kebutuhan cadangan Anda.

Pencadangan dan pemulihan

Aurora DSQL mendukung pencadangan dan pemulihan dengan. Konsol AWS Backup Anda dapat melakukan pencadangan dan pemulihan penuh untuk kluster Single-region dan Multi-region Anda. Untuk informasi selengkapnya, lihat Cadangkan dan pulihkan untuk Amazon Aurora DSQL.

Replikasi

Secara desain, Aurora DSQL melakukan semua transaksi tulis ke log transaksi terdistribusi dan secara sinkron mereplikasi semua data log yang berkomitmen ke replika penyimpanan pengguna dalam tiga. AZs Cluster Multi-Region menyediakan kemampuan replikasi Lintas wilayah lengkap antara Wilayah baca dan tulis.

Wilayah saksi yang ditunjuk mendukung penulisan log-only transaksi dan tidak menggunakan penyimpanan. Wilayah Saksi tidak memiliki titik akhir. Ini berarti bahwa Witness Regions hanya menyimpan log transaksi terenkripsi, tidak memerlukan administrasi atau konfigurasi, dan tidak dapat diakses oleh pengguna.

Log transaksi Aurora DSQL dan penyimpanan pengguna didistribusikan dengan semua data yang disajikan ke prosesor kueri Aurora DSQL sebagai volume logis tunggal. Aurora DSQL secara otomatis membagi, menggabungkan, dan mereplikasi data berdasarkan rentang kunci utama basis data dan pola akses. Aurora DSQL secara otomatis menskalakan replika baca, baik naik maupun turun, berdasarkan frekuensi akses baca.

Replika penyimpanan cluster didistribusikan di seluruh armada penyimpanan multi-penyewa. Jika komponen atau AZ menjadi rusak, Aurora DSQL secara otomatis mengalihkan akses ke komponen yang masih ada dan secara asinkron memperbaiki replika yang hilang. Setelah Aurora DSQL memperbaiki replika yang rusak, Aurora DSQL secara otomatis menambahkannya kembali ke kuorum penyimpanan dan membuatnya tersedia untuk cluster Anda.

Ketersediaan tinggi

Secara default, kluster Single-region dan Multi-region di Aurora DSQL adalah aktif-aktif, dan Anda tidak perlu menyediakan, mengonfigurasi, atau mengkonfigurasi ulang cluster apa pun secara manual. Aurora DSQL sepenuhnya mengotomatiskan pemulihan cluster, yang menghilangkan

kebutuhan untuk operasi failover primer-sekunder tradisional. Replikasi selalu sinkron dan dilakukan dalam beberapa AZs, sehingga tidak ada risiko kehilangan data karena replikasi lag atau failover ke database sekunder asinkron selama pemulihan kegagalan.

Cluster Wilayah Tunggal menyediakan titik akhir redundan multi-AZ yang secara otomatis memungkinkan akses bersamaan dengan konsistensi data yang kuat di tiga. AZs Ini berarti bahwa replika penyimpanan pengguna pada salah satu dari ketiganya AZs selalu mengembalikan hasil yang sama ke satu atau lebih pembaca dan selalu tersedia untuk menerima tulisan. Konsistensi yang kuat dan ketahanan Multi-AZ ini tersedia di semua Wilayah untuk klaster Multi-wilayah Aurora DSQL. Ini berarti bahwa klaster Multi-wilayah menyediakan dua titik akhir Regional yang sangat konsisten, sehingga klien dapat membaca atau menulis tanpa pandang bulu ke salah satu Wilayah tanpa jeda replikasi pada komit.

Aurora DSQL menyediakan 99,99% ketersediaan untuk kluster Single-region dan 99,999% untuk cluster Multi-region.

Keamanan Infrastruktur di Amazon Aurora DSQL

Sebagai layanan terkelola, Amazon Aurora DSQL dilindungi oleh prosedur keamanan jaringan AWS global yang dijelaskan dalam Praktik Terbaik untuk Keamanan, Identitas, & Kepatuhan.

Anda menggunakan panggilan API yang AWS dipublikasikan untuk mengakses Aurora DSQL melalui jaringan. Klien harus mendukung Keamanan Lapisan Pengangkutan (TLS) 1.2 atau versi yang lebih baru. Klien juga harus mendukung suite cipher dengan perfect forward secrecy (PFS) seperti Ephemeral Diffie-Hellman (DHE) atau Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Sebagian besar sistem modern seperti Java 7 dan versi lebih baru mendukung mode-mode ini.

Selain itu, permintaan harus ditandatangani menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan prinsipal IAM. Atau Anda dapat menggunakan AWS Security Token Service (AWS STS) untuk menghasilkan kredensial keamanan sementara untuk menandatangani permintaan.

Mengelola dan menghubungkan ke cluster DSQL Amazon Aurora menggunakan AWS PrivateLink

Dengan AWS PrivateLink Amazon Aurora DSQL, Anda dapat menyediakan antarmuka titik akhir Amazon VPC (titik akhir antarmuka) di Amazon Virtual Private Cloud Anda. Titik akhir ini dapat

diakses langsung dari aplikasi yang ada di lokasi melalui Amazon VPC AWS Direct Connect dan, atau berbeda dengan Wilayah AWS peering VPC Amazon. Menggunakan AWS PrivateLink dan antarmuka endpoint, Anda dapat menyederhanakan konektivitas jaringan pribadi dari aplikasi Anda ke Aurora DSQL.

Aplikasi dalam VPC Amazon Anda dapat mengakses Aurora DSQL menggunakan titik akhir antarmuka Amazon VPC tanpa memerlukan alamat IP publik.

Titik akhir antarmuka diwakili oleh satu atau lebih antarmuka jaringan elastis (ENIs) yang diberi alamat IP pribadi dari subnet di VPC Amazon Anda. Permintaan ke Aurora DSQL melalui titik akhir antarmuka tetap ada di jaringan. AWS Untuk informasi selengkapnya tentang cara menghubungkan VPC Amazon dengan jaringan lokal, lihat Panduan AWS Direct Connect Pengguna dan Panduan Pengguna AWS Site-to-Site VPN VPN.

Untuk informasi umum tentang titik akhir antarmuka, lihat Mengakses AWS layanan menggunakan antarmuka titik akhir Amazon VPC di AWS PrivateLinkPanduan Pengguna.

Jenis titik akhir VPC Amazon untuk Aurora DSQL

Aurora DSQL membutuhkan dua jenis endpoint yang berbeda. AWS PrivateLink

1. Endpoint manajemen — Endpoint ini digunakan untuk operasi administratif, seperti,,`get`, `create` `update``delete`, dan pada cluster `list` Aurora DSQL. Lihat Mengelola cluster Aurora DSQL menggunakan AWS PrivateLink.
2. Connection endpoint — Endpoint ini digunakan untuk menghubungkan ke cluster Aurora DSQL melalui klien PostgreSQL. Lihat Menghubungkan ke cluster Aurora DSQL menggunakan AWS PrivateLink.

Pertimbangan saat menggunakan AWS PrivateLink untuk Aurora DSQL

Pertimbangan Amazon VPC berlaku untuk AWS PrivateLink Aurora DSQL. Untuk informasi selengkapnya, lihat Mengakses AWS layanan menggunakan titik akhir VPC antarmuka dan AWS PrivateLink kuota di Panduan. AWS PrivateLink

Mengelola cluster Aurora DSQL menggunakan AWS PrivateLink

Anda dapat menggunakan AWS Command Line Interface atau AWS Software Development Kit (SDKs) untuk mengelola cluster Aurora DSQL melalui titik akhir antarmuka Aurora DSQL.

Membuat titik akhir Amazon VPC

Untuk membuat titik akhir antarmuka VPC Amazon, lihat Membuat titik akhir VPC Amazon di Panduan. AWS PrivateLink

```
aws ec2 create-vpc-endpoint \
--region region \
--service-name com.amazonaws.region.dsql \
--vpc-id your-vpc-id \
--subnet-ids your-subnet-id \
--vpc-endpoint-type Interface \
--security-group-ids client-sg-id \
```

Untuk menggunakan nama DNS Regional default untuk permintaan API Aurora DSQL, jangan nonaktifkan DNS pribadi saat Anda membuat titik akhir antarmuka Aurora DSQL. Ketika DNS pribadi diaktifkan, permintaan ke layanan Aurora DSQL yang dibuat dari dalam VPC Amazon Anda akan secara otomatis menyelesaikan ke alamat IP pribadi titik akhir VPC Amazon, bukan nama DNS publik. Saat DNS pribadi diaktifkan, permintaan Aurora DSQL yang dibuat dalam VPC Amazon Anda akan secara otomatis diselesaikan ke titik akhir VPC Amazon Anda.

Jika DNS pribadi tidak diaktifkan, gunakan `--endpoint-url` parameter `--region` dan dengan AWS CLI perintah untuk mengelola cluster Aurora DSQL melalui titik akhir antarmuka Aurora DSQL.

Daftar cluster menggunakan URL endpoint

Dalam contoh berikut, ganti Wilayah AWS `us-east-1` dan nama DNS `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` ID endpoint Amazon VPC dengan informasi Anda sendiri.

```
aws dsql --region us-east-1 --endpoint-url https://vpce-1a2b3c4d-5e6f.dsql.us-
east-1.vpce.amazonaws.com list-clusters
```

Operasi API

Lihat referensi Aurora DSQL API untuk dokumentasi pengelolaan sumber daya di Aurora DSQL.

Mengelola kebijakan endpoint

Dengan menguji dan mengonfigurasi kebijakan titik akhir VPC Amazon secara menyeluruh, Anda dapat membantu memastikan bahwa klaster Aurora DSQL Anda aman, sesuai, dan selaras dengan kontrol akses dan persyaratan tata kelola khusus organisasi Anda.

Contoh: Kebijakan akses DSQL Aurora Penuh

Kebijakan berikut memberikan akses penuh ke semua tindakan dan sumber daya Aurora DSQL melalui titik akhir VPC Amazon yang ditentukan.

```
aws ec2 modify-vpc-endpoint \
    --vpc-endpoint-id vpce-xxxxxxxxxxxxxxxxx \
    --region region \
    --policy-document '{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": "*",
          "Action": "dsql:*",
          "Resource": "*"
        }
      ]
    }'
```

Contoh: Kebijakan Akses Aurora DSQL Terbatas

Kebijakan berikut hanya mengizinkan tindakan Aurora DSQL ini.

- `CreateCluster`
- `GetCluster`
- `ListClusters`

Semua tindakan Aurora DSQL lainnya ditolak.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "dsql:CreateCluster",
```

```
        "dsql:GetCluster",
        "dsql:ListClusters"
      ],
      "Resource": "*"
    }
  ]
}
```

Menghubungkan ke cluster Aurora DSQL menggunakan AWS PrivateLink

Setelah AWS PrivateLink endpoint Anda diatur dan aktif, Anda dapat terhubung ke cluster Aurora DSQL Anda menggunakan klien PostgreSQL. Instruksi koneksi di bawah ini menguraikan langkah-langkah untuk membangun nama host yang tepat untuk menghubungkan melalui titik akhir. AWS PrivateLink

Menyiapkan titik akhir AWS PrivateLink koneksi

Langkah 1: Dapatkan nama layanan untuk cluster Anda

Saat membuat AWS PrivateLink endpoint untuk menghubungkan ke cluster Anda, Anda harus terlebih dahulu mengambil nama layanan khusus cluster.

AWS CLI

```
aws dsql get-vpc-endpoint-service-name \
--region us-east-1 \
--identifier your-cluster-id
```

Contoh tanggapan

```
{
    "serviceName": "com.amazonaws.us-east-1.dsql-fnh4"
}
```

Nama layanan termasuk pengenal, seperti `dsql-fnh4` dalam contoh. Pengenal ini juga diperlukan saat membuat nama host untuk menghubungkan ke cluster Anda.

AWS SDK for Python (Boto3)

```
import boto3
```

```
dsql_client = boto3.client('dsql', region_name='us-east-1')
response = dsql_client.get_vpc_endpoint_service_name(
    identifier='your-cluster-id'
)
service_name = response['serviceName']
print(f"Service Name: {service_name}")
```

AWS SDK for Java 2.x

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetVpcEndpointServiceNameRequest;
import software.amazon.awssdk.services.dsql.model.GetVpcEndpointServiceNameResponse;

String region = "us-east-1";
String clusterId = "your-cluster-id";

DsqlClient dsqlClient = DsqlClient.builder()
    .region(Region.of(region))
    .credentialsProvider(DefaultCredentialsProvider.create())
    .build();

GetVpcEndpointServiceNameResponse response = dsqlClient.getVpcEndpointServiceName(
    GetVpcEndpointServiceNameRequest.builder()
        .identifier(clusterId)
        .build()
);
String serviceName = response.serviceName();
System.out.println("Service Name: " + serviceName);
```

Langkah 2: Buat titik akhir Amazon VPC

Menggunakan nama layanan yang diperoleh pada langkah sebelumnya, buat titik akhir VPC Amazon.

> **Important**
>
> Petunjuk koneksi di bawah ini hanya berfungsi untuk menghubungkan ke cluster saat privat diaktifkan DNS. Jangan gunakan `--no-private-dns-enabled` bendera saat membuat titik akhir, karena ini akan mencegah instruksi koneksi di bawah ini berfungsi dengan baik.

Jika Anda menonaktifkan DNS pribadi, Anda harus membuat catatan DNS pribadi wildcard Anda sendiri yang menunjuk ke titik akhir yang dibuat.

AWS CLI

```
aws ec2 create-vpc-endpoint \
    --region us-east-1 \
    --service-name service-name-for-your-cluster \
    --vpc-id your-vpc-id \
    --subnet-ids subnet-id-1 subnet-id-2  \
    --vpc-endpoint-type Interface \
    --security-group-ids security-group-id
```

Contoh respon

```
{
    "VpcEndpoint": {
        "VpcEndpointId": "vpce-0123456789abcdef0",
        "VpcEndpointType": "Interface",
        "VpcId": "vpc-0123456789abcdef0",
        "ServiceName": "com.amazonaws.us-east-1.dsql-fnh4",
        "State": "pending",
        "RouteTableIds": [],
        "SubnetIds": [
            "subnet-0123456789abcdef0",
            "subnet-0123456789abcdef1"
        ],
        "Groups": [
            {
                "GroupId": "sg-0123456789abcdef0",
                "GroupName": "default"
            }
        ],
        "PrivateDnsEnabled": true,
        "RequesterManaged": false,
        "NetworkInterfaceIds": [
            "eni-0123456789abcdef0",
            "eni-0123456789abcdef1"
        ],
        "DnsEntries": [
            {
```

```
                "DnsName": "*.dsql-fnh4.us-east-1.vpce.amazonaws.com",
                "HostedZoneId": "Z7HUB22UULQXV"
            }
        ],
        "CreationTimestamp": "2025-01-01T00:00:00.000Z"
    }
}
```

SDK for Python

```
import boto3

ec2_client = boto3.client('ec2', region_name='us-east-1')
response = ec2_client.create_vpc_endpoint(
    VpcEndpointType='Interface',
    VpcId='your-vpc-id',
    ServiceName='com.amazonaws.us-east-1.dsql-fnh4',  # Use the service name from
 previous step
    SubnetIds=[
        'subnet-id-1',
        'subnet-id-2'
    ],
    SecurityGroupIds=[
        'security-group-id'
    ]
)

vpc_endpoint_id = response['VpcEndpoint']['VpcEndpointId']
print(f"VPC Endpoint created with ID: {vpc_endpoint_id}")
```

SDK for Java 2.x

Gunakan URL endpoint untuk Aurora DSQL APIs

```
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.ec2.Ec2Client;
import software.amazon.awssdk.services.ec2.model.CreateVpcEndpointRequest;
import software.amazon.awssdk.services.ec2.model.CreateVpcEndpointResponse;
import software.amazon.awssdk.services.ec2.model.VpcEndpointType;

String region = "us-east-1";
```

```
String serviceName = "com.amazonaws.us-east-1.dsql-fnh4";  // Use the service name
 from previous step
String vpcId = "your-vpc-id";

Ec2Client ec2Client = Ec2Client.builder()
    .region(Region.of(region))
    .credentialsProvider(DefaultCredentialsProvider.create())
    .build();

CreateVpcEndpointRequest request = CreateVpcEndpointRequest.builder()
    .vpcId(vpcId)
    .serviceName(serviceName)
    .vpcEndpointType(VpcEndpointType.INTERFACE)
    .subnetIds("subnet-id-1", "subnet-id-2")
    .securityGroupIds("security-group-id")
    .build();

CreateVpcEndpointResponse response = ec2Client.createVpcEndpoint(request);
String vpcEndpointId = response.vpcEndpoint().vpcEndpointId();
System.out.println("VPC Endpoint created with ID: " + vpcEndpointId);
```

Menghubungkan ke cluster Aurora DSQL menggunakan titik akhir koneksi AWS PrivateLink

Setelah AWS PrivateLink titik akhir Anda diatur dan aktif (periksa apakah ada`available`), Anda dapat terhubung ke cluster Aurora DSQL Anda menggunakan klien PostgreSQL. `State` Untuk petunjuk penggunaan AWS SDKs, Anda dapat mengikuti panduan dalam Pemrograman dengan Aurora DSQL. Anda harus mengubah titik akhir cluster agar sesuai dengan format nama host.

Membangun nama host

Nama host untuk menghubungkan AWS PrivateLink berbeda dari nama host DNS publik. Anda perlu membangunnya menggunakan komponen-komponen berikut.

1. `Your-cluster-id`
2. Pengidentifikasi layanan dari nama layanan. Misalnya: `dsql-fnh4`
3. The Wilayah AWS

Gunakan format berikut: *`cluster-id`*`.`*`service-identifier`*`.`*`region`*`.on.aws`

Contoh: Koneksi Menggunakan PostgreSQL

```
# Set environment variables
export CLUSTERID=your-cluster-id
export REGION=us-east-1
export SERVICE_IDENTIFIER=dsql-fnh4  # This should match the identifier in your service
 name

# Construct the hostname
export HOSTNAME="$CLUSTERID.$SERVICE_IDENTIFIER.$REGION.on.aws"

# Generate authentication token
export PGPASSWORD=$(aws dsql --region $REGION generate-db-connect-admin-auth-token --
hostname $HOSTNAME)

# Connect using psql
psql -d postgres -h $HOSTNAME -U admin
```

Memecahkan masalah dengan AWS PrivateLink

Masalah dan Solusi Umum

Tabel berikut mencantumkan masalah umum dan solusi yang berkaitan AWS PrivateLink dengan Aurora DSQL.

Isu	Kemungkinan penyebab	Solusi
Batas waktu koneksi	Grup keamanan tidak dikonfigurasi dengan benar	Gunakan Amazon VPC Reachability Analyzer untuk memastikan penyiapan jaringan Anda memungkinkan lalu lintas di port 5432.
Kegagalan resolusi DNS	DNS pribadi tidak diaktifkan	Verifikasi bahwa titik akhir VPC Amazon dibuat dengan DNS pribadi diaktifkan.
Kegagalan otentikasi	Kredensi salah atau token kedaluwarsa	Buat token otentikasi baru dan verifikasi nama pengguna.
Nama layanan tidak ditemukan	ID cluster salah	Periksa kembali ID cluster Anda dan Wilayah AWS saat mengambil nama layanan.

Sumber Daya Terkait

Untuk informasi selengkapnya, lihat sumber daya berikut:

- Panduan Pengguna Amazon Aurora DSQL
- Dokumentasi AWS PrivateLink
- Akses AWS layanan melalui AWS PrivateLink

Analisis konfigurasi dan kerentanan di Amazon Aurora DSQL

AWS menangani tugas-tugas keamanan dasar seperti sistem operasi tamu (OS) dan patching database, konfigurasi firewall, dan pemulihan bencana. Prosedur ini telah ditinjau dan disertifikasi oleh pihak ketiga yang sesuai. Untuk detail selengkapnya, lihat sumber daya berikut:

- Model tanggung jawab bersama
- Amazon Web Services: Ikhtisar proses keamanan (whitepaper)

Pencegahan "confused deputy" lintas layanan

Masalah "confused deputy" adalah masalah keamanan saat entitas yang tidak memiliki izin untuk melakukan suatu tindakan dapat memaksa entitas yang memilik hak akses lebih tinggi untuk melakukan tindakan tersebut. 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 principal layanan yang telah diberi akses ke sumber daya di akun Anda.

Sebaiknya gunakan kunci konteks kondisi `aws:SourceAccount`global `aws:SourceArn`dan global dalam kebijakan sumber daya untuk membatasi izin yang diberikan Amazon Aurora DSQL layanan lain ke sumber daya. Gunakan `aws:SourceArn` jika Anda ingin hanya satu sumber daya yang akan dikaitkan dengan akses lintas layanan. Gunakan `aws:SourceAccount` jika Anda ingin mengizinkan sumber daya apa pun di akun tersebut dikaitkan dengan penggunaan lintas layanan.

Cara paling efektif untuk melindungi dari masalah "confused deputy" adalah dengan menggunakan kunci konteks kondisi global `aws:SourceArn` dengan ARN lengkap sumber daya. Jika Anda tidak

mengetahui ARN lengkap sumber daya atau jika Anda menentukan beberapa sumber daya, gunakan kunci kondisi konteks global `aws:SourceArn` dengan karakter wildcard (*) untuk bagian ARN yang tidak diketahui. Misalnya, `arn:aws:dsql:*:123456789012:*`.

Jika nilai `aws:SourceArn` tidak berisi ID akun, seperti ARN bucket Amazon S3, Anda harus menggunakan kedua kunci konteks kondisi global tersebut untuk membatasi izin.

Nilai `aws:SourceArn` harus ResourceDescription.

Contoh berikut menunjukkan bagaimana Anda dapat menggunakan `aws:SourceArn` dan kunci konteks kondisi `aws:SourceAccount` global di Aurora DSQL untuk mencegah masalah wakil bingung.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "backup.amazonaws.com"
    },
    "Action": "dsql:GetCluster",
    "Resource": [
      "arn:aws:dsql:*:123456789012:cluster/*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:backup:*:123456789012:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
      }
    }
  }
}
```

Praktik terbaik keamanan untuk Aurora DSQL

Aurora DSQL menyediakan sejumlah fitur keamanan untuk dipertimbangkan saat Anda mengembangkan dan menerapkan kebijakan keamanan Anda sendiri. Praktik terbaik berikut adalah pedoman umum dan tidak mewakili solusi keamanan yang lengkap. Karena praktik terbaik ini mungkin tidak sesuai atau tidak memadai untuk lingkungan Anda, perlakukan itu sebagai pertimbangan yang bermanfaat, bukan sebagai resep.

Topik

Praktik terbaik keamanan Detektif untuk Aurora DSQL

Selain cara-cara berikut untuk menggunakan Aurora DSQL dengan aman, lihat Keamanan AWS Well-Architected Tool untuk mempelajari tentang bagaimana teknologi cloud meningkatkan keamanan Anda.

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 dipelihara selama jangka waktu tertentu.

Tandai sumber daya Aurora DSQL Anda untuk identifikasi dan otomatisasi

Anda dapat menetapkan metadata ke AWS sumber daya Anda dalam bentuk tag. Setiap tanda adalah label sederhana yang terdiri dari kunci yang ditetapkan pelanggan dan nilai opsional yang memudahkan untuk mengelola, mencari, dan memfilter sumber daya.

Penandaan memungkinkan implementasi kontrol berkelompok. Meskipun tidak ada jenis tanda yang melekat, tanda memungkinkan Anda untuk mengelompokkan sumber daya berdasarkan tujuan, pemilik, lingkungan, atau kriteria lainnya. Berikut ini beberapa contohnya:

- Keamanan – Digunakan untuk menentukan persyaratan seperti enkripsi.
- Kerahasiaan – Sebuah pengidentifikasi untuk dukungan sumber daya tingkat kerahasiaan data tertentu.

- Lingkungan – Digunakan untuk membedakan antara pengembangan, pengujian, dan infrastruktur produksi.

Anda dapat menetapkan metadata ke AWS sumber daya Anda dalam bentuk tag. Setiap tanda adalah label sederhana yang terdiri dari kunci yang ditetapkan pelanggan dan nilai opsional yang memudahkan untuk mengelola, mencari, dan memfilter sumber daya.

Penandaan memungkinkan implementasi kontrol berkelompok. Meskipun tidak ada jenis tag yang melekat, tag memungkinkan Anda untuk mengelompokkan sumber daya berdasarkan tujuan, pemilik, lingkungan, atau kriteria lainnya. Berikut ini adalah beberapa contoh.

- Keamanan — digunakan untuk menentukan persyaratan seperti enkripsi.
- Kerahasiaan — pengenal untuk tingkat kerahasiaan data tertentu yang didukung sumber daya.
- Lingkungan — digunakan untuk membedakan antara pembangunan, pengujian, dan infrastruktur produksi.

Untuk informasi selengkapnya, lihat Praktik Terbaik untuk Menandai AWS Sumber Daya.

Praktik terbaik keamanan preventif untuk Aurora DSQL

Selain cara-cara berikut untuk menggunakan Aurora DSQL dengan aman, lihat Keamanan AWS Well-Architected Tool untuk mempelajari tentang bagaimana teknologi cloud meningkatkan keamanan Anda.

Gunakan peran IAM untuk mengautentikasi akses ke Aurora DSQL.

Pengguna, aplikasi, dan lainnya Layanan AWS yang mengakses Aurora DSQL harus menyertakan AWS kredensi yang valid dalam API dan permintaan. AWS AWS CLI Anda tidak boleh menyimpan AWS kredensil secara langsung di aplikasi atau EC2 instance. Ini adalah kredensi jangka panjang yang tidak diputar secara otomatis. Ada dampak bisnis yang signifikan jika kredensil ini dikompromikan. Peran IAM memungkinkan Anda memperoleh kunci akses sementara yang dapat Anda gunakan untuk mengakses Layanan AWS dan sumber daya.

Untuk informasi selengkapnya, lihat Otentikasi dan otorisasi untuk Aurora DSQL.

Gunakan kebijakan IAM untuk otorisasi dasar Aurora DSQL.

Saat Anda memberikan izin, Anda memutuskan siapa yang mendapatkannya, operasi Aurora DSQL API mana yang mereka dapatkan izin, dan tindakan spesifik yang ingin Anda izinkan pada

sumber daya tersebut. Menerapkan akses hak akses paling rendah adalah hal mendasar dalam mengurangi risiko keamanan dan dampak yang dapat disebabkan oleh kesalahan atau niat jahat.

Lampirkan kebijakan izin ke peran IAM dan berikan izin untuk melakukan operasi pada sumber daya Aurora DSQL. Juga tersedia batas izin untuk entitas IAM, yang memungkinkan Anda menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM.

Mirip dengan praktik terbaik pengguna root untuk Anda Akun AWS, jangan gunakan `admin` peran di Aurora DSQL untuk melakukan operasi sehari-hari. Sebagai gantinya, kami menyarankan Anda membuat peran basis data khusus untuk mengelola dan terhubung ke klaster Anda. Untuk informasi selengkapnya, lihat Mengakses Aurora DSQL dan Memahami otentikasi dan otorisasi untuk Aurora DSQL.

Gunakan **`verify-full`** di lingkungan produksi.

Pengaturan ini memverifikasi bahwa sertifikat server ditandatangani oleh otoritas sertifikat tepercaya dan bahwa nama host server cocok dengan sertifikat.

Perbarui klien PostgreSQL Anda

Perbarui klien PostgreSQL Anda secara teratur ke versi terbaru untuk mendapatkan manfaat dari peningkatan keamanan. Kami merekomendasikan menggunakan PostgreSQL versi 17.

Menandai sumber daya di Aurora DSQL

Di AWS, tag adalah pasangan nilai kunci yang ditentukan pengguna yang Anda tentukan dan kaitkan dengan sumber daya Aurora DSQL seperti cluster. Tanda adalah opsional. Jika Anda memberikan kunci, nilainya opsional.

Anda dapat menggunakan AWS Management Console,, atau AWS SDKs untuk menambahkan AWS CLI, daftar, dan menghapus tag pada cluster Aurora DSQL. Anda dapat menambahkan tag selama dan setelah pembuatan cluster menggunakan AWS konsol. Untuk menandai cluster setelah pembuatan dengan AWS CLI menggunakan `TagResource` operasi.

Menandai cluster dengan Nama

Aurora DSQL membuat cluster dengan pengenal unik global yang ditetapkan sebagai Amazon Resource Name (ARN). Jika Anda ingin menetapkan nama yang ramah pengguna ke klaster Anda, kami sarankan Anda menggunakan Tag.

Jika Anda membuat konsol dengan konsol Aurora DSQL, Aurora DSQL secara otomatis membuat tag. Tag ini memiliki kunci Nama dan nilai yang dihasilkan secara otomatis yang mewakili nama cluster. Nilai ini dapat dikonfigurasi, sehingga Anda dapat menetapkan nama yang lebih ramah ke cluster Anda. Jika sebuah cluster memiliki tag Nama dengan nilai terkait, Anda dapat melihat nilai di seluruh konsol Aurora DSQL.

Persyaratan penandaan

Tag memiliki persyaratan sebagai berikut:

- Kunci tidak dapat diawali dengan`aws:`.
- Kunci harus unik per set tag.
- Kunci harus antara 1 dan 128 karakter yang diizinkan.
- Nilai harus antara 0 dan 256 karakter yang diizinkan.
- Nilai tidak harus unik per set tag.
- Karakter yang diizinkan untuk kunci dan nilai adalah huruf, digit, spasi putih, dan salah satu simbol berikut: _.:/= + - @.
- Kunci dan nilai peka huruf besar dan kecil.

Menandai catatan penggunaan

Saat menggunakan tag di Aurora DSQL, pertimbangkan hal berikut.

- Saat menggunakan operasi Aurora DSQL API AWS CLI atau Aurora, pastikan untuk memberikan Nama Sumber Daya Amazon (ARN) agar sumber daya Aurora DSQL dapat digunakan. Untuk informasi selengkapnya, lihat format Amazon Resource Name (ARNs) untuk sumber daya Aurora DSQL.
- Setiap sumber daya memiliki satu set tag, yang merupakan kumpulan dari satu atau beberapa tag yang ditetapkan ke sumber daya.
- Setiap sumber daya dapat memiliki hingga 50 tag per set tag.
- Jika Anda menghapus sumber daya, tag terkait akan dihapus.
- Anda dapat menambahkan tag saat membuat sumber daya, Anda dapat melihat dan memodifikasi tag menggunakan operasi API berikut:`TagResource`,`UntagResource`, dan`ListTagsForResource`.
- Anda dapat menggunakan tag dengan kebijakan IAM. Anda dapat menggunakannya untuk mengelola akses ke cluster Aurora DSQL dan untuk mengontrol tindakan apa yang dapat diterapkan pada sumber daya tersebut. Untuk mempelajari lebih lanjut, lihat Mengontrol akses ke AWS sumber daya menggunakan tag.
- Anda dapat menggunakan tag untuk berbagai aktivitas lainnya AWS. Untuk mempelajari lebih lanjut, lihat Strategi penandaan umum.

Pertimbangan untuk bekerja dengan Amazon Aurora DSQL

Pertimbangkan perilaku berikut saat Anda bekerja dengan Amazon Aurora DSQL. Untuk informasi selengkapnya tentang kompatibilitas dan dukungan PostgreSQL, lihat. Kompatibilitas fitur SQL di Aurora DSQL Untuk kuota dan batasan, lihatKuota cluster dan batas database di Amazon Aurora DSQL.

- Aurora DSQL tidak menyelesaikan `COUNT(*)` operasi sebelum batas waktu transaksi untuk tabel besar. Untuk mengambil jumlah baris tabel dari katalog sistem, lihat Menggunakan tabel dan perintah sistem di Aurora DSQL.
- Pemanggilan driver `PG_PREPARED_STATEMENTS` mungkin memberikan tampilan yang tidak konsisten dari pernyataan yang disiapkan dalam cache untuk cluster. Anda mungkin melihat lebih dari jumlah yang diharapkan dari pernyataan yang disiapkan per koneksi untuk cluster dan peran IAM yang sama. Aurora DSQL tidak menyimpan nama pernyataan yang Anda siapkan.
- Dalam skenario penurunan klaster tertaut Multi-wilayah yang jarang terjadi, mungkin diperlukan waktu lebih lama dari yang diharapkan untuk melanjutkan ketersediaan komit transaksi. Secara umum, operasi pemulihan klaster otomatis dapat mengakibatkan kontrol konkurensi sementara atau kesalahan koneksi. Dalam kebanyakan kasus, Anda hanya akan melihat efek untuk persentase dari beban kerja Anda. Ketika Anda melihat kesalahan transit ini, coba kembali transaksi Anda atau hubungkan kembali dengan klien Anda.
- Beberapa klien SQL, seperti Datagrip, membuat panggilan ekspansif ke metadata sistem untuk mengisi informasi skema. Aurora DSQL tidak mendukung semua informasi ini dan mengembalikan kesalahan. Masalah ini tidak memengaruhi fungsionalitas kueri SQL, tetapi mungkin memengaruhi tampilan skema.
- Peran admin memiliki seperangkat izin yang terkait dengan tugas manajemen database. Secara default, izin ini tidak meluas ke objek yang dibuat pengguna lain. Peran admin tidak dapat memberikan atau mencabut izin pada objek yang dibuat pengguna ini kepada pengguna lain. Pengguna admin dapat memberikan dirinya sendiri peran lain untuk mendapatkan izin yang diperlukan pada objek ini.

Kuota cluster dan batas database di Amazon Aurora DSQL

Bagian berikut menjelaskan kuota cluster dan batas database untuk Aurora DSQL.

Kuota cluster

Anda Akun AWS memiliki kuota cluster berikut di Aurora DSQL. Untuk meminta peningkatan kuota layanan untuk kluster Single-region dan Multi-region dalam suatu kelompok tertentu Wilayah AWS, gunakan halaman konsol Service Quotas. Untuk kenaikan kuota lainnya, hubungi AWS Dukungan.

Deskripsi	Batas default	Dapat dikonfigurasi?	Kode kesalahan Aurora DSQL
Maksimum kluster wilayah tunggal per Akun AWS	20 kluster	Ya	Kode kesalahan API `ServiceQuotaExceededExcep`
Kluster Multi-wilayah maksimum per Akun AWS	5 cluster	Ya	Kode kesalahan API `ServiceQuotaExceededExcep`
Penyimpanan maksimum per cluster	Batas default 10 TiB, hingga 128 TiB dengan peningkatan batas yang disetujui	Ya	`DISK_FULL(53100)`
Koneksi maksimum per cluster	10.000 koneksi	Ya	`TOO_MANY_CONNECTIONS(53300)`

Deskripsi	Batas default	Dapat dikonfigurasi?	Kode kesalahan Aurora DSQL
Tingkat koneksi maksimum per cluster	100 koneksi per detik	Tidak	`CONFIGURED_LIMIT_EXCEEDED(53400)`
Kapasitas burst koneksi maksimum per cluster	1.000 koneksi	Tidak	Tidak ada kode kesalahan
Pekerjaan pemulihan bersamaan maksimum	4	Tidak	Tidak ada kode kesalahan
Tingkat Isi Ulang Koneksi	100 koneksi per detik	Tidak	Tidak ada kode kesalahan

Batas basis data di Aurora DSQL

Tabel berikut menjelaskan batas database di Aurora DSQL.

Deskripsi	Batas default	Dapat dikonfigurasi?	Kode kesalahan Aurora DSQL	Pesan kesalahan
Ukuran gabungan maksimum kolom yang digunakan dalam kunci primer	1 KiB	Tidak	`54000`	`ERROR: key size too large`

Deskripsi	Batas default	Dapat dikonfigurasi?	Kode kesalahan Aurora DSQL	Pesan kesalahan
Ukuran gabungan maksimum kolom dalam indeks sekunder	1 KiB	Tidak	54000	ERROR: key size too large
Ukuran maksimum baris dalam tabel	2 MiB	Tidak	54000	ERROR: maximum row size excee
Ukuran maksimum kolom yang bukan bagian dari indeks	1 MiB	Tidak	54000	ERROR: maximum column size ex
Jumlah maksimum kolom dalam kunci primer atau indeks sekunder	8	Tidak	54011	ERROR: more than 8 column key are not supported
Jumlah maksimum kolom dalam tabel	255	Tidak	54011	ERROR: tables can have at mos
Jumlah maksimum indeks dalam tabel	24	Tidak	54000	ERROR: more than 24 indexes p allowed

Deskripsi	Batas default	Dapat dikonfigurasi?	Kode kesalahan Aurora DSQL	Pesan kesalahan
Ukuran maksimum semua data yang dimodifikasi dalam transaksi tulis	10 MiB	Tidak	54000	ERROR: transaction size limit DETAIL: Current transaction s 10mb
Jumlah maksimum baris tabel dan indeks yang dapat dimutasi dalam blok transaksi	3.000 baris per transaksi. Lihat Pertimbangan Aurora DSQL untuk kompatibilitas PostgreSQL.	Tidak	54000	ERROR: transaction row limit
Jumlah dasar maksimum memori yang dapat digunakan oleh operasi kueri	128 MiB per transaksi	Tidak	53200	ERROR: query requires too muc out of memory.
Jumlah maksimum skema yang didefinisikan dalam database	10	Tidak	54000	ERROR: more than 10 schemas n
Jumlah maksimum tabel dalam database	1.000 tabel	Tidak	54000	ERROR: creating more than 100 allowed

Deskripsi	Batas default	Dapat dikonfigurasi?	Kode kesalahan Aurora DSQL	Pesan kesalahan
Jumlah maksimum database dalam sebuah cluster	1	Tidak	Tidak ada kode kesalahan	`ERROR: unsupported statement`
Waktu transaksi maksimum	5 menit	Tidak	`54000`	`ERROR: transaction age limit exceeded`
Durasi koneksi maksimum	60 menit	Tidak	Tidak ada kode kesalahan	Tidak ada pesan kesalahan
Jumlah maksimum tampilan dalam database	5.000	Tidak	`54000`	`ERROR: creating more than 500 allowed`
Ukuran definisi tampilan maksimum	2 MiB	Tidak	`54000`	`ERROR: view definition too la`

Untuk batas tipe data khusus untuk Aurora DSQL, lihat. Tipe data yang didukung di Aurora DSQL

Referensi Aurora DSQL API

Selain AWS Management Console dan AWS Command Line Interface (AWS CLI), Aurora DSQL juga menyediakan antarmuka API. Anda dapat menggunakan operasi API untuk mengelola sumber daya Anda di Aurora DSQL.

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 lebih lanjut tentang AWS CLI, lihat AWS Command Line Interface referensi untuk Aurora DSQL.

Memecahkan masalah di Aurora DSQL

> **Note**
>
> Topik berikut memberikan saran pemecahan masalah untuk kesalahan dan masalah yang mungkin Anda temui saat menggunakan Aurora DSQL. Jika Anda menemukan masalah yang tidak tercantum di sini, hubungi AWS dukungan

Topik

Memecahkan masalah kesalahan koneksi

kesalahan: kode kesalahan SSL yang tidak dikenal: 6

Penyebab: Anda mungkin menggunakan versi psql lebih awal dari versi 14, yang tidak mendukung Indikasi Nama Server (SNI). SNI diperlukan saat menghubungkan ke Aurora DSQL.

Anda dapat memeriksa versi klien Anda dengan`psql --version`.

kesalahan: NetworkUnreachable

`NetworkUnreachable`Kesalahan selama upaya koneksi mungkin menunjukkan bahwa klien Anda tidak mendukung IPv6 koneksi, daripada menandakan masalah jaringan yang sebenarnya. Kesalahan ini biasanya terjadi pada instance IPv4 -only karena bagaimana klien PostgreSQL menangani koneksi dual-stack. Ketika server mendukung mode dual-stack, klien ini pertama-tama menyelesaikan nama host ke keduanya IPv4 dan alamat. IPv6 Mereka mencoba IPv4 koneksi terlebih dahulu, lalu coba IPv6 jika koneksi awal gagal. Jika sistem Anda tidak mendukung IPv6, Anda akan melihat `NetworkUnreachable` kesalahan umum alih-alih pesan “IPv6 tidak didukung” yang jelas.

Memecahkan masalah kesalahan otentikasi

Autentikasi IAM gagal untuk pengguna “...”

Saat Anda membuat token otentikasi Aurora DSQL IAM, durasi maksimum yang dapat Anda atur adalah 1 minggu. Setelah satu minggu, Anda tidak dapat mengautentikasi dengan token itu.

Selain itu, Aurora DSQL menolak permintaan koneksi Anda jika peran yang Anda anggap telah kedaluwarsa. Misalnya, jika Anda mencoba terhubung dengan peran IAM sementara meskipun token otentikasi Anda belum kedaluwarsa, Aurora DSQL akan menolak permintaan koneksi.

Untuk mempelajari lebih lanjut tentang bagaimana IAM bekerja dengan Aurora DSQL, lihat Memahami otentikasi dan otorisasi untuk Aurora DSQL dan di Aurora DSQL.AWS Identity and Access Management

Terjadi kesalahan (InvalidAccessKeyId) saat memanggil GetObject operasi: ID Kunci AWS Akses yang Anda berikan tidak ada dalam catatan kami

IAM menolak permintaan Anda. Untuk informasi selengkapnya, lihat Mengapa permintaan ditandatangani.

Peran IAM tidak ada <role>

Aurora DSQL tidak dapat menemukan peran IAM Anda. Untuk informasi selengkapnya, lihat peran IAM.

Peran IAM harus terlihat seperti ARN IAM

Lihat IAM Identifiers - IAM ARNs untuk informasi lebih lanjut.

Memecahkan masalah kesalahan otorisasi

Peran tidak didukung <role>

Aurora DSQL tidak mendukung operasi. `GRANT` Lihat Subset yang didukung dari perintah PostgreSQL di Aurora DSQL.

Tidak dapat membangun kepercayaan dengan peran <role>

Aurora DSQL tidak mendukung operasi. `GRANT` Lihat Subset yang didukung dari perintah PostgreSQL di Aurora DSQL.

Peran tidak ada <role>

Aurora DSQL tidak dapat menemukan pengguna database tertentu. Lihat Mengotorisasi peran basis data kustom untuk terhubung ke klaster.

KESALAHAN: izin ditolak untuk memberikan kepercayaan IAM dengan peran <role>

Untuk memberikan akses ke peran database, Anda harus terhubung ke klaster Anda dengan peran admin. Untuk mempelajari selengkapnya, lihat Mengotorisasi peran database untuk menggunakan SQL dalam database.

ERROR: peran harus memiliki atribut LOGIN <role>

Setiap peran database yang Anda buat harus memiliki `LOGIN` izin.

Untuk mengatasi kesalahan ini, pastikan Anda telah membuat Peran PostgreSQL dengan izin. `LOGIN` Untuk informasi selengkapnya, lihat MEMBUAT PERAN dan MENGUBAH PERAN dalam dokumentasi PostgreSQL.

KESALAHAN: peran tidak dapat dijatuhkan karena beberapa objek bergantung padanya <role>

Aurora DSQL mengembalikan kesalahan jika Anda menjatuhkan peran database dengan hubungan IAM sampai Anda mencabut hubungan menggunakan. `AWS IAM REVOKE` Untuk mempelajari lebih lanjut, lihat Mencabut otorisasi.

Memecahkan masalah kesalahan SQL

Kesalahan: Tidak didukung

Aurora DSQL tidak mendukung semua dialek berbasis PostgreSQL. Untuk mempelajari tentang apa yang didukung, lihat Fitur PostgreSQL yang Didukung di Aurora DSQL.

Kesalahan: PILIH UNTUK PEMBARUAN dalam transaksi hanya-baca adalah no-op

Anda mencoba operasi yang tidak diizinkan dalam transaksi hanya-baca. Untuk mempelajari lebih lanjut, lihat Memahami kontrol konkurensi di Aurora DSQL.

Kesalahan: gunakan **`CREATE INDEX ASYNC`** sebagai gantinya

Untuk membuat indeks pada tabel dengan baris yang ada, Anda harus menggunakan `CREATE INDEX ASYNC` perintah. Untuk mempelajari lebih lanjut, lihat Membuat indeks secara asinkron di Aurora DSQL.

Memecahkan masalah kesalahan OCC

OC000 “ERROR: mutasi bertentangan dengan transaksi lain, coba lagi sesuai kebutuhan”

OC001 “ERROR: skema telah diperbarui oleh transaksi lain, coba lagi sesuai kebutuhan”

Sesi PostgreSQL Anda memiliki salinan katalog skema yang di-cache. Salinan yang di-cache itu valid pada saat dimuat. Mari kita sebut waktu T1 dan versi V1.

Transaksi lain memperbarui katalog pada waktu T2. Mari kita sebut ini V2.

Ketika sesi asli mencoba membaca dari penyimpanan pada waktu T2 itu masih menggunakan katalog versi V1. Lapisan penyimpanan Aurora DSQL menolak permintaan karena versi katalog terbaru di T2 adalah V2.

Ketika Anda mencoba lagi pada waktu T3 dari sesi asli, Aurora DSQL menyegarkan cache katalog. Transaksi di T3 menggunakan katalog V2. Aurora DSQL akan menyelesaikan transaksi selama tidak ada perubahan katalog lain yang terjadi sejak waktu T2.

Memecahkan masalah koneksi SSL/TLS

Kesalahan SSL: verifikasi sertifikat gagal

Kesalahan ini menunjukkan bahwa klien tidak dapat memverifikasi sertifikat server. Pastikan bahwa:

1. Sertifikat Amazon Root CA 1 diinstal dengan benar. Lihat Mengkonfigurasi SSL/TLS sertifikat untuk koneksi Aurora DSQL petunjuk tentang cara memvalidasi dan menginstal sertifikat ini.
2. Variabel `PGSSLROOTCERT` lingkungan menunjuk ke file sertifikat yang benar.
3. File sertifikat memiliki izin yang benar.

Kode kesalahan SSL yang tidak dikenal: 6

Kesalahan ini terjadi dengan klien PostgreSQL di bawah versi 14. Tingkatkan klien PostgreSQL Anda ke versi 17 untuk mengatasi masalah ini.

Kesalahan SSL: skema tidak terdaftar (Windows)

Ini adalah masalah yang diketahui dengan klien psql Windows saat menggunakan sertifikat sistem. Gunakan metode file sertifikat yang diunduh yang dijelaskan dalam Menghubungkan dari Windows instruksi.

Riwayat dokumen untuk Panduan Pengguna Amazon Aurora DSQL

Tabel berikut menjelaskan rilis dokumentasi untuk Aurora DSQL.

Perubahan	Deskripsi	Tanggal
Ketersediaan umum (GA) Amazon Aurora DSQL	Amazon Aurora DSQL sekarang tersedia secara umum dengan dukungan tambahan untuk CloudWatch pemantauan, fitur perlindungan data yang ditingkatkan, dan integrasi. AWS Backup Untuk informasi selengkapnya, lihat Memantau Aurora DSQL dengan, CloudWatchMencadangkan dan memulihkan Amazon Aurora DSQL, dan Enkripsi data untuk Amazon Aurora DSQL.	27 Mei 2025
AmazonAuroraDSQLFullAkses pembaruan	Menambahkan kemampuan untuk melakukan operasi pencadangan dan pemulihan untuk cluster Aurora DSQL, termasuk memulai, menghentikan, dan memantau pekerjaan. Ini juga menambahkan kemampuan untuk menggunakan kunci KMS yang dikelola pelanggan untuk enkripsi cluster. Untuk informasi selengkapnya, lihat AmazonAuroraDSQLFu	21 Mei 2025

	llMengakses dan Menggunakan peran terkait layanan di Aurora DSQL.	
AmazonAuroraDSQLConsoleFullAccess perbarui	Menambahkan kemampuan untuk melakukan operasi pencadangan dan pemulihan untuk cluster Aurora DSQL melalui. AWS Console Home Ini termasuk memulai, menghentikan, dan memantau pekerjaan. Ini juga mendukung penggunaan kunci KMS yang dikelola pelanggan untuk enkripsi dan peluncuran cluster. AWS CloudShell Untuk informasi selengkapnya, lihat AmazonAuroraDSQLConsoleFullAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	21 Mei 2025

AmazonAuroraDSQLReadOnlyAccess perbarui	Termasuk kemampuan untuk menentukan nama layanan titik akhir VPC yang benar saat menghubungkan ke cluster Aurora DSQL Anda melalui Aurora DSQL menciptakan titik akhir unik per sel, sehingga API ini membantu memastikan Anda dapat mengidentifikasi titik akhir yang benar untuk cluster Anda dan menghindari kesalahan koneksi. AWS PrivateLink Untuk informasi selengkapnya, lihat AmazonAuroraDSQLReadOnlyAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	13 Mei 2025

AmazonAuroraDSQLFullAksespembaruan	Kebijakan ini menambahkan empat izin baru untuk membuat dan mengelola kluster database di beberapa Wilayah AWS:`PutMultiRegionProperties` ,, `PutWitnessRegion` `AddPeerCluster` , dan. `RemovePeerCluster` Izin ini mencakup kontrol tingkat sumber daya dan kunci kondisi sehingga Anda dapat mengontrol pengguna kluster mana yang dapat Anda modifikasi. Kebijakan ini juga menambahkan `GetVpcEndpointServiceName` izin untuk membantu Anda terhubung ke klaster Aurora DSQL Anda. AWS PrivateLink Untuk informasi selengkapnya, lihat AmazonAuroraDSQLConsoleFullAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	13 Mei 2025

AmazonAuroraDSQLConsoleFullAccess perbarui	Menambahkan izin baru ke Aurora DSQL untuk mendukung manajemen klaster Multi-wilayah dan koneksi titik akhir VPC. Izin baru meliputi: `PutMultiRegionProperties` `PutWitnessRegion` `AddPeerCluster` `RemovePeerCluster` `GetVpcEndpointServiceName` . Lihat AmazonAuroraDSQLConsoleFullAccessdan Menggunakan peran terkait layanan di Aurora DSQL.	13 Mei 2025
AuroraDsqlServiceLinkedRolePolicy perbarui	Menambahkan kemampuan untuk memublikasikan metrik ke `AWS/Usage` `CloudWatch` ruang nama `AWS/AuroraDSQL` dan ruang nama ke kebijakan. Hal ini memungkinkan layanan atau peran terkait untuk memancarkan data penggunaan dan kinerja yang lebih komprehensif ke CloudWatch lingkungan Anda. Untuk informasi selengkapnya, lihat AuroraDsqlServiceLinkedRolePolicydan Menggunakan peran terkait layanan di Aurora DSQL.	8 Mei 2025

AWS PrivateLink untuk Amazon Aurora DSQL	Aurora DSQL sekarang mendukung. AWS PrivateLink Dengan AWS PrivateLink, Anda dapat menyederhanakan konektivitas jaringan pribadi antara virtual private cloud (VPCs), Aurora DSQL, dan pusat data lokal Anda menggunakan antarmuka titik akhir VPC Amazon dan alamat IP pribadi. Untuk informasi selengkapnya, lihat Mengelola dan menghubungkan ke kluster DSQL Amazon Aurora menggunakan. AWS PrivateLink	8 Mei 2025
Rilis awal	Rilis awal Panduan Pengguna Amazon Aurora DSQL.	Desember 3, 2024

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.