aws

Guía del usuario

Amazon Aurora DSQL

Amazon Aurora DSQL: Guía del usuario

Table of Contents

¿Qué es Amazon Aurora?

Amazon Aurora DSQL es un servicio de base de datos relacional distribuido y sin servidor optimizado para cargas de trabajo transaccionales. Aurora DSQL ofrece una escala prácticamente ilimitada y no requiere administrar la infraestructura. La arquitectura de alta disponibilidad activa-activa proporciona una disponibilidad del 99,99 % para una región y del 99,999 % para varias regiones.

Cuándo utilizar Aurora DSQL

Aurora DSQL se ha optimizado para cargas de trabajo transaccionales que se benefician de las transacciones ACID y de un modelo de datos relacional. Al no ser sin servidor, Aurora DSQL es ideal para patrones de aplicación de arquitecturas de microservicios, sin servidor y basadas en eventos. Aurora DSQL es compatible con PostgreSQL, por lo que puede utilizar controladores conocidos, asignaciones relacionales de objetos (ORM), marcos de trabajo y características SQL.

Aurora DSQL administra automáticamente la infraestructura del sistema y escala la computación, la E/S y el almacenamiento en función de la carga de trabajo. Como no tiene servidores que aprovisionar ni administrar, no tiene que preocuparse por el tiempo de inactividad por mantenimiento relacionado con el aprovisionamiento, la aplicación de parches o las actualizaciones de la infraestructura.

Aurora DSQL lo ayuda a crear y mantener aplicaciones empresariales siempre disponibles a cualquier escala. El diseño sin servidor activo-activo automatiza la recuperación de errores, por lo que no tendrá que preocuparse de la conmutación por error tradicional de las bases de datos. Las aplicaciones se benefician de la disponibilidad Multi-AZ y multirregional, y no tiene que preocuparse por la coherencia final o la falta de datos relacionada con las conmutaciones por error.

Características principales de Aurora DSQL

Las siguientes características clave lo ayudan a crear una base de datos distribuida sin servidor para admitir las aplicaciones de alta disponibilidad:

Arquitectura distribuida

Aurora DSQL se compone de los siguientes componentes de varios inquilinos:

- Retransmisión y conectividad
- Computación y bases de datos

- Registro de transacciones, control de simultaneidad y aislamiento
- Almacenamiento

Un plano de control coordina los componentes anteriores. Cada componente proporciona redundancia en tres zonas de disponibilidad (AZ), con escalado automático de clústeres y reparación automática en caso de error de los componentes. Para obtener más información sobre cómo esta arquitectura admite la alta disponibilidad, consulte Resiliencia en Amazon Aurora DSQL.

Clústeres de una región y multirregionales

Los clústeres de Aurora DSQL proporcionan los siguientes beneficios:

- Replicación de datos síncrona
- Operaciones de lectura coherentes
- Recuperación automática de errores
- Coherencia de datos en varias zonas de disponibilidad o regiones

Si se produce un error un componente de la infraestructura, Aurora DSQL enruta automáticamente las solicitudes a la infraestructura con el estado correcto sin intervención manual. Aurora DSQL proporciona transacciones de atomicidad, coherencia, aislamiento y durabilidad (ACID) con gran coherencia, aislamiento de instantáneas, atomicidad y durabilidad entre las AZ y las regiones.

Los clústeres emparejados de varias regiones proporcionan la misma resiliencia y conectividad que los clústeres de una sola región. Pero mejoran la disponibilidad al ofrecer dos puntos de conexión regionales, uno en cada región del clúster emparejado. Ambos puntos de conexión de un clúster emparejado presentan una única base de datos lógica. Están disponibles para operaciones de lectura y escritura simultáneas y ofrecen una gran coherencia de datos. Puede crear aplicaciones que se ejecuten en varias regiones al mismo tiempo para mejorar el rendimiento y la resiliencia, y saber que los espectadores siempre ven los mismos datos.

Compatibilidad con bases de datos PostgreSQL

La capa de base de datos distribuida (computación) en Aurora DSQL se basa en una versión principal actual de PostgreSQL. Puede conectarse a Aurora DSQL con controladores y herramientas de PostgreSQL conocidos, como `psql`. Aurora DSQL es actualmente compatible con la versión 16 de PostgreSQL y admite un subconjunto de características, expresiones y tipos de datos de PostgreSQL. Para obtener más información sobre las características de SQL compatibles, consulte Compatibilidad con características SQL en Aurora DSQL.

Disponibilidad de regiones para Aurora DSQL

Con Amazon Aurora DSQL, puede implementar instancias de base de datos en varias Regiones de AWS para admitir aplicaciones globales y cumplir los requisitos de residencia de datos. La disponibilidad de región determina dónde puede crear y administrar clústeres de bases de datos de Aurora DSQL. Los administradores de bases de datos y los arquitectos de aplicaciones que necesitan diseñar sistemas de bases de datos de alta disponibilidad distribuidos por todo el mundo a menudo necesitan conocer la compatibilidad de las regiones con las cargas de trabajo. Entre los casos de uso más comunes se incluyen la configuración de la recuperación ante desastres entre regiones, el servicio a los usuarios desde instancias de base de datos geográficamente más cercanas para reducir la latencia y el mantenimiento de copias de datos en ubicaciones específicas para el cumplimiento.

En la siguiente tabla se muestran las Regiones de AWS dónde está disponible actualmente Aurora DSQL y el punto de conexión para cada Región de AWS.

Regiones de AWS y puntos de enlace admitidos

Nombre de la región	Región	Punto de conexión	Protocolo
Este de EE. UU. (Norte de Virginia)	us-east-1	dsql.us-east-1.api.aws	HTTPS
Este de EE. UU. (Ohio)	us-east-2	dsql.us-east-2.api.aws	HTTPS
Oeste de EE. UU. (Oregón)	us-west-2	dsql.us-west-2.api.aws	HTTPS
Europa (Irlanda)	eu-west-1	dsql.eu-west-1.api.aws	HTTPS
Europa (Londres)	eu-west-2	dsql.eu-west-2.api.aws	HTTPS
Europa (París)	eu-west-3	dsql.eu-west-3.api.aws	HTTPS
Asia-Pacífico (Tokio)	ap-northeast-1	dsql.ap-northeast-1.api.aws	HTTPS
Asia-Pacífico (Seúl)	ap-northeast-2	dsql.ap-northeast-2.api.aws	HTTPS

Nombre de la región	Región	Punto de conexión	Protocolo
Asia-Pacífico (Osaka)	ap-northeast-3	dsql.ap-northeast-3.api.aws	HTTPS

Disponibilidad de clústeres multirregionales para Aurora DSQL

Puede crear clústeres multirregionales de Aurora DSQL en conjuntos de regiones de AWS específicos. Cada conjunto de regiones agrupa regiones relacionadas geográficamente que pueden trabajar juntas en un clúster multirregional.

Regiones de EE. UU.

- Este de EE. UU. (Norte de Virginia)
- Este de EE. UU. (Ohio)
- Oeste de EE. UU. (Oregón)

Regiones de Asia-Pacífico

- Asia-Pacífico (Osaka)
- Asia-Pacífico (Seúl)
- Asia-Pacífico (Tokio)

Regiones europeas

- Europa (Irlanda)
- Europa (Londres)
- Europa (París)

Limitaciones importantes

Los clústeres multirregionales deben crearse en un único conjunto de regiones. Por ejemplo, no puede crear un clúster que incluya las regiones Este de EE. UU. (Norte de Virginia) y Europa (Irlanda).

> **Important**
>
> Aurora DSQL no admite actualmente clústeres multirregionales intercontinentales.

Precios para Aurora DSQL

Para obtener información sobre los costos, consulte precios de Aurora DSQL.

Siguientes pasos

Para obtener información sobre los componentes principales de Aurora DSQL y comenzar a utilizar el servicio, consulte lo siguiente:

- Introducción a Aurora DSQL
- Compatibilidad con características SQL en Aurora DSQL
- Acceso a Aurora DSQL
- Aurora DSQL y PostgreSQL

Introducción a Aurora DSQL

Amazon Aurora DSQL es una base de datos relacional distribuida y sin servidor optimizada para cargas de trabajo transaccionales. En las siguientes secciones, obtendrá información sobre cómo crear clústeres de Aurora DSQL de una región y multirregionales, conectarse a ellos y crear y cargar un esquema de ejemplo. Accederá a los clústeres con la AWS Management Console e interactuará con la base de datos con la utilidad `psql`. Al final, tendrá un clúster de Aurora DSQL en funcionamiento configurado y listo para usarse en cargas de trabajo de prueba o producción.

Temas

Requisitos previos

Antes de empezar a utilizar Aurora DSQL, asegúrese de que cumple los siguientes requisitos previos:

- Su identidad de IAM debe tener permiso para iniciar sesión en la AWS Management Console.
- Su identidad de IAM debe cumplir alguno de los siguientes criterios:
  - Acceso para realizar cualquier acción en cualquier recurso de la Cuenta de AWS
  - El permiso de IAM `iam:CreateServiceLinkedRole` y la capacidad de obtener acceso a la acción de política de IAM `dsql:*`
- Si utiliza la AWS CLI en un entorno tipo Unix, asegúrese de que Python versión 3.8 o superiores y `psql` versión 14 o superiores están instalados. Para consultar las versiones de las aplicaciones, ejecute los siguientes comandos.

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

Si utiliza la AWS CLI en un entorno diferente, asegúrese de configurar de forma manual Python versión 3.8 o superiores y `psql` versión 14 o superiores.

- Si tiene la intención de acceder a Aurora DSQL mediante AWS CloudShell, Python versión 3.8 o superiores y `psql` versión 14 o superiores se suministran sin configuración adicional. Para obtener más información acerca de AWS CloudShell, consulte ¿Qué es AWS CloudShell?
- Si tiene la intención de acceder a Aurora DSQL con una GUI, utilice DBeaver o JetBrains DataGrip. Para obtener más información, consulte Acceso a Aurora DSQL con DBeaver y Acceso a Aurora DSQL con JetBrains DataGrip.

Acceso a Aurora DSQL

Puede acceder a Aurora DSQL mediante las siguientes técnicas. Para obtener información sobre cómo utilizar la CLI, las API y los SDK, consulte Acceso a Aurora DSQL.

Temas

- Acceso a Aurora DSQL a través de la AWS Management Console
- Acceso a Aurora DSQL con clientes de SQL
- Uso del protocolo PostgreSQL con Aurora DSQL

Acceso a Aurora DSQL a través de la AWS Management Console

Puede acceder a la AWS Management Console para Aurora DSQL en https://console.aws.amazon.com/dsql.

Acceso a Aurora DSQL con clientes de SQL

Aurora DSQL utiliza el protocolo PostgreSQL. Utilice el cliente interactivo que prefiera; para ello, proporcione un token de autenticación de IAM firmado como contraseña cuando se conecte al clúster. Un token de autenticación es una cadena de caracteres única que Aurora DSQL genera dinámicamente mediante AWS Signature Version 4.

Aurora DSQL utiliza el token solo para la autenticación. El token no afecta la conexión una vez establecida. Si intenta conectarse mediante un token caducado, se denegará la solicitud de conexión. Para obtener más información, consulte Generación de un token de autenticación en Amazon Aurora DSQL.

Temas

Acceso a Aurora DSQL con psql (terminal interactivo de PostgreSQL)

La utilidad `psql` es un frontend de PostgreSQL basado en terminal. Le permite escribir consultas de forma interactiva, emitirlas a PostgreSQL y ver los resultados de la consulta. Para mejorar los tiempos de respuesta de las consultas, utilice el cliente de PostgreSQL versión 17.

Descargue el instalador del sistema operativo desde la página de Descargas de PostgreSQL. Para obtener más información sobre `psql`, consulte https://www.postgresql.org/docs/current/app-psql.htm.

Si ya tiene instalada la AWS CLI, utilice el siguiente ejemplo para conectarse al clúster. Puede utilizar AWS CloudShell, que tiene `psql` preinstalado, o puede instalar `psql` directamente.

```
# 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
```

Acceso a Aurora DSQL con DBeaver

DBeaver es una herramienta de base de datos de código abierto basada en GUI. Puede utilizarla para conectarse a la base de datos y administrarla. Para descargar DBeaver, consulte la página

de descargas del sitio web de la comunidad de DBeaver. En los siguientes pasos se explica cómo conectarse al clúster mediante DBeaver.

Configuración de una nueva conexión de Aurora DSQL en DBeaver

1. Elija Nueva conexión de base de datos.
2. En la ventana Nueva conexión de base de datos, elija PostgreSQL.
3. En la pestaña Configuración de la conexión/Principal, elija Conectar por: host e introduzca la siguiente información.

   - Host: utilice el punto de conexión del clúster.

     Base de datos: ingrese `postgres`

     Autenticación: elija `Database Native`

     Nombre de usuario: ingrese `admin`

     Contraseña: genere un token de autenticación. Copie el token generado y utilícelo como contraseña.
4. Ignore cualquier advertencia y pegue el token de autenticación en el campo Contraseña de DBeaver.

   > **Note**
   >
   > Debe establecer el modo SSL en las conexiones de cliente. Aurora DSQL admite `SSLMODE=require`. Aurora DSQL aplica la comunicación SSL en el servidor y rechaza las conexiones que no sean SSL.
5. Debe estar conectado al clúster y puede empezar a ejecutar instrucciones SQL.

> **Important**
>
> Las características de administración que proporciona DBeaver para las bases de datos PostgreSQL (como Session Manager y Lock Manager) no se aplican a una base de datos, debido a su arquitectura única. Aunque son accesibles, estas pantallas no proporcionan información fiable sobre el estado de la base de datos.

Caducidad de las credenciales de autenticación para DBeaver

Las sesiones establecidas permanecen autenticadas durante un máximo de 1 hora o bien hasta que DBeaver se desconecte o se agote el tiempo de espera. Para establecer nuevas conexiones, debe proporcionarse un token de autenticación válido en el campo Contraseña de la Configuración de conexión. El intento de abrir una nueva sesión (por ejemplo, para enumerar las tablas nuevas o una nueva consola SQL) fuerza un nuevo intento de autenticación. Si el token de autenticación configurado en la Configuración de conexión ya no es válido, se producirá un error en la nueva sesión y DBeaver invalida todas las sesiones abiertas anteriormente. Tenga esto en cuenta cuando elija la duración del token de autenticación de IAM con la opción `expires-in`.

Acceso a Aurora DSQL con JetBrains DataGrip

JetBrains DataGrip es un IDE multiplataforma para trabajar con SQL y bases de datos, incluido PostgreSQL. DataGrip incluye una GUI robusta con un editor SQL inteligente. Para descargar DataGrip, vaya a la página de descargas del sitio web de JetBrains.

Configuración de una nueva conexión de Aurora DSQL en JetBrains DataGrip

1. Elija Nuevo origen de datos y elija PostgreSQL.
2. En la pestaña Orígenes de datos/General, introduzca la siguiente información:
   - Host: utilice el punto de conexión del clúster.

     Puerto: Aurora DSQL utiliza el predeterminado de PostgreSQL: `5432`

     Base de datos: Aurora DSQL utiliza la predeterminada de PostgreSQL de `postgres`

     Autenticación: elija `User & Password` .

     Nombre de usuario: ingrese `admin`.

     Contraseña: genere un token y péguelo en este campo.

     URL: no modifique este campo. Se rellenará automáticamente según los demás campos.
3. Contraseña: proporciónela mediante la generación de un token de autenticación. Copie la salida resultante del generador de tokens y péguela en el campo de contraseña.

> **Note**
>
> Debe establecer el modo SSL en las conexiones de cliente. Aurora DSQL admite `PGSSLMODE=require`. Aurora DSQL aplica la comunicación SSL en el servidor y rechazará las conexiones que no sean SSL.

4. Debe estar conectado al clúster y puede empezar a ejecutar instrucciones SQL:

> **Important**
>
> Algunas vistas que proporciona DataGrip para las bases de datos de PostgreSQL (como Sesiones) no se aplican a una base de datos debido a su arquitectura única. Aunque son accesibles, estas pantallas no proporcionan información fiable sobre las sesiones reales conectadas a la base de datos.

Caducidad de las credenciales de autenticación

Las sesiones establecidas permanecen autenticadas durante un máximo de 1 hora o bien hasta que se produzca una desconexión explícita o se agote el tiempo de espera del cliente. Si es necesario establecer nuevas conexiones, debe generarse un nuevo token de autenticación y proporcionarlo en el campo Contraseña de las Propiedades del origen de datos. El intento de abrir una nueva sesión (por ejemplo, para enumerar las tablas nuevas o una nueva consola SQL) fuerza un nuevo intento de autenticación. Si el token de autenticación configurado en la configuración de Conexión ya no es válido, se producirá un error en esa nueva sesión y todas las sesiones abiertas anteriormente dejarán de ser válidas.

Uso del protocolo PostgreSQL con Aurora DSQL

PostgreSQL utiliza un protocolo basado en mensajes para la comunicación entre clientes y servidores. El protocolo se admite sobre TCP/IP y también sobre sockets de dominio Unix. En la siguiente tabla se muestra cómo Aurora DSQL admite el protocolo PostgreSQL.

PostgreSQL	Aurora DSQL	Notas
Rol (también conocido como Usuario o Grupo)	Rol de base de datos	Aurora DSQL crea un rol para usted denominado `admin`. Cuando crea roles de

PostgreSQL	Aurora DSQL	Notas
		base de datos personalizados, debe utilizar el rol de `admin` para asociarlos con los roles de IAM para la autenticación cuando se conecte al clúster. Para obtener más información, consulte Configuración de roles de base de datos personalizados.
Host (también conocido como hostname o hostspec)	Punto de conexión de clúster	Los clústeres de una sola región de Aurora DSQL proporcionan un único punto de conexión administrado y redirigen automátic amente el tráfico si no hay disponibilidad en la región.
Puerto	N/A: utilice el valor predeterminado `5432`	Este es el valor predeterminado de PostgreSQL.
Base de datos (dbname)	Utilizar `postgres`	Aurora DSQL crea esta base de datos para usted cuando crea el clúster.
Modo SSL	SSL siempre está habilitado en el servidor	En Aurora DSQL, Aurora DSQL admite el modo SSL `require`. Aurora DSQL rechaza las conexiones sin SSL.
Contraseña	Token de autentica ción	Aurora DSQL requiere tokens de autentica ción temporales en lugar de contraseñas de larga duración. Para obtener más informaci ón, consulte Generación de un token de autenticación en Amazon Aurora DSQL.

Paso 1: creación de un clúster de Aurora DSQL de una sola región

La unidad básica de Aurora DSQL es el clúster, que es donde se almacenan los datos. En esta tarea, crea un clúster en una sola Región de AWS.

Creación de un clúster de una sola región en Aurora DSQL

1. Inicie sesión en la AWS Management Console y abra la consola de Aurora DSQL en https://console.aws.amazon.com/dsql.
2. Elija Crear clúster y, a continuación, Una sola región.
3. (Opcional) En Configuración del clúster, seleccione cualquiera de las siguientes opciones:
   - Seleccione Personalizar la configuración de cifrado (avanzada) para elegir o crear una AWS KMS key.
   - Seleccione Habilitar la protección contra la eliminación para evitar que se elimine el clúster. De forma predeterminada, la protección contra la eliminación está seleccionada.
4. (Opcional) En Etiquetas, elija o ingrese una etiqueta para este clúster.
5. Elija Create cluster.

Paso 2: conexión al clúster de Aurora DSQL

Un punto de conexión del clúster se genera automáticamente al crear un clúster de Aurora DSQL en función de su ID de clúster y región. El formato de nomenclatura es `clusterid.dsql.region.on.aws`. Un cliente usa el punto de conexión para crear una conexión de red con el clúster.

La autenticación se administra mediante IAM, por lo que no necesita almacenar credenciales en la base de datos. Un token de autenticación es una cadena única de caracteres que se genera dinámicamente. El token solo se utiliza para la autenticación y no afecta la conexión después de establecerse. Antes de intentar conectarse, asegúrese de que la identidad de IAM tiene el permiso `dsql:DbConnectAdmin`, como se describe en Requisitos previos.

> **Note**
>
> Para optimizar la velocidad de conexión a la base de datos, utilice el cliente de PostgreSQL versión 17 y establezca `PGSSLNEGOTIATION` para dirigir: `PGSSLNEGOTIATION=direct`.

Conexión al clúster con un token de autenticación

1. En la consola de Aurora DSQL, elija el clúster al que desea conectarse.
2. Elija Conectar.

3. Copie el punto de conexión desde Punto de conexión (host).
4. Asegúrese de elegir Conectarse como administrador en la sección Token de autenticación (contraseña).
5. Copie el token de autenticación generado. Este token es válido durante 15 minutos.
6. En la línea de comandos del sistema operativo, utilice el siguiente comando para iniciar `psql` y conectarse al clúster. Reemplace *`your_cluster_endpoint`* por el punto de conexión de clúster que ha copiado anteriormente.

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

   Cuando se le pida una contraseña, introduzca el token de autenticación que ha copiado anteriormente. Si intenta conectarse mediante un token caducado, se denegará la solicitud de conexión. Para obtener más información, consulte Generación de un token de autenticación en Amazon Aurora DSQL.
7. Pulse Intro. Debe ver una petición de PostgreSQL.

```
postgres=>
```

   Si obtiene un error de acceso denegado, asegúrese de que la identidad de IAM tiene el permiso `dsql:DbConnectAdmin`. Si tiene el permiso y sigue teniendo errores de acceso denegado, consulte Solución de problemas de IAM y ¿Cómo puedo solucionar los errores de acceso denegado u operación no autorizada con una política de IAM?

Paso 3: ejecución de comandos SQL de ejemplo en Aurora DSQL

Pruebe el clúster de Aurora DSQL mediante la ejecución de instrucciones SQL. Las siguientes instrucciones de ejemplo requieren los archivos de datos denominados `department-insert-multirow.sql` y `invoice.csv`, que puede descargar del repositorio aws-samples/aurora-dsql-samples en GitHub.

Ejecución de comandos SQL de ejemplo en Aurora DSQL

1. Cree un esquema denominado `example`.

```
CREATE SCHEMA example;
```

2. Cree una tabla de facturas que utilice un UUID generado automáticamente como la clave principal.

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

3. Cree un índice secundario que utilice la tabla vacía.

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

4. Cree una tabla de departamentos.

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

5. Utilice el comando `psql \include` para cargar el archivo denominado `department-insert-multirow.sql` que descargó del repositorio aws-samples/aurora-dsql-samples en GitHub. Reemplace *my-path* por la ruta a la copia local.

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

6. Utilice el comando `psql \copy` para cargar el archivo denominado `invoice.csv` que descargó del repositorio aws-samples/aurora-dsql-samples en GitHub. Reemplace *my-path* por la ruta a la copia local.

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

7. Consulte los departamentos y ordénelos por las ventas totales.

```
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;
```

En la siguiente salida de ejemplo se muestra que el departamento tres es el que tiene más ventas.

```
           name           |     sum_amount
--------------------------+--------------------
 Example Department Three |  54061.67752854594
 Example Department Seven |  53869.65965365204
 Example Department Eight |  52199.73742066634
 Example Department One   | 52034.078869900826
 Example Department Six   |  50886.15556256385
 Example Department Two   |  50589.98422247931
 Example Department Five  | 49549.852635496005
 Example Department Four  |  49266.15578027619
(8 rows)
```

Paso 4: creación de un clúster de varias regiones

Cuando crea un clúster de varias regiones, especifica las siguientes regiones:

Región remota

Esta es la región en la que crea un segundo clúster. Crea un segundo clúster en esta región y lo empareja con el clúster inicial. Aurora DSQL replica todas las escrituras del clúster inicial en el clúster remoto. Puede leer y escribir en cualquier clúster.

Región testigo

Esta región recibe todos los datos que se escriben en el clúster de varias regiones. Sin embargo, las regiones testigo no alojan puntos de conexión de cliente y no proporcionan acceso a los datos de usuario. En las regiones testigo se mantiene un intervalo limitado del registro cifrado de transacciones. Este registro facilita la recuperación y admite el cuórum transaccional si una región no está disponible.

El siguiente ejemplo muestra cómo crear un clúster inicial, crear un segundo clúster en una región diferente y, a continuación, emparejar los dos clústeres para crear un clúster de varias regiones. Demuestra también la replicación de escritura entre regiones y las lecturas coherentes desde ambos puntos de conexión regionales.

Creación de un clúster de varias regiones

1. Inicie sesión en la AWS Management Console y abra la consola de Aurora DSQL en https://console.aws.amazon.com/dsql.
2. En el panel de navegación, seleccione Clusters (Clústeres).
3. Elija Crear clúster y, a continuación, Varias regiones.
4. (Opcional) En Configuración del clúster, seleccione cualquiera de las siguientes opciones para el clúster inicial:
   - Seleccione Personalizar la configuración de cifrado (avanzada) para elegir o crear una AWS KMS key.
   - Seleccione Habilitar la protección contra la eliminación para evitar que se elimine el clúster. De forma predeterminada, la protección contra la eliminación está seleccionada.
5. En Configuración de varias regiones, elija las siguientes opciones para el clúster inicial:
   - En Región testigo, elija una región. Actualmente, solo se admiten las regiones ubicadas en EE. UU. como regiones testigo en los clústeres de varias regiones.
   - (Opcional) En ARN del clúster de región remota, ingrese un ARN para un clúster existente en otra región. Si no existe ningún clúster que sirva como segundo clúster en el clúster de varias regiones, complete la configuración después de crear el clúster inicial.
6. (Opcional) Elija etiquetas para el clúster inicial.
7. Elija Crear clúster para crear el clúster inicial. Si no ingresó un ARN en el paso anterior, la consola mostrará la notificación Configuración del clúster pendiente.
8. En la notificación Configuración del clúster pendiente, elija Completar la configuración del clúster de varias regiones. Esta acción inicia la creación de un segundo clúster en otra región.
9. Elija una de las siguientes opciones para el segundo clúster:
   - Agregar el ARN del clúster de región remota: elija esta opción si existe un clúster y desea que sea el segundo clúster del clúster de varias regiones.
   - Crear clúster en otra región: elija esta opción para crear un segundo clúster. En Región remota, elija la región para este segundo clúster.
10. Elija Crear clúster en ***your-second-region***, donde *your-second-region* es la ubicación del segundo clúster. La consola se abre en la segunda región.
11. (Opcional) Elija la configuración del clúster para el segundo clúster. Por ejemplo, puede elegir una AWS KMS key.

12. Elija Crear clúster para crear el segundo clúster.
13. Elija Emparejar en ***initial-cluster-region***, donde *initial-cluster-region* es la región que aloja el primer clúster que creó.
14. Cuando se le pida confirmación, elija Confirmar. Este paso completa la creación del clúster de varias regiones.

Conexión al segundo clúster

1. Abra la consola de Aurora DSQL y elija la región del segundo clúster.
2. Seleccione Clusters (Clústeres).
3. Seleccione la fila del segundo clúster en el clúster de varias regiones.
4. En Acciones, elija Abrir en CloudShell.
5. Elija Conectar como administrador.
6. Elija Launch CloudShell.
7. Seleccione Ejecutar.
8. Cree un esquema de ejemplo siguiendo los pasos de Paso 3: ejecución de comandos SQL de ejemplo en Aurora DSQL.

   Transacciones de ejemplo

   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. Utilice los comandos `copy` y `include` de `psql` para cargar datos de ejemplo. Para obtener más información, consulte Paso 3: ejecución de comandos SQL de ejemplo en Aurora DSQL.

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

Consulta de los datos del segundo clúster desde la región que aloja el clúster inicial

1. En la consola de Aurora DSQL, elija la región del clúster inicial.
2. Seleccione Clusters (Clústeres).
3. Seleccione la fila del segundo clúster en el clúster de varias regiones.
4. En Acciones, elija Abrir en CloudShell.
5. Elija Conectar como administrador.
6. Elija Launch CloudShell.
7. Seleccione Ejecutar.
8. Consulte los datos que ha insertado en el segundo clúster.

   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;
```

Autenticación y autorización para Aurora DSQL

Aurora DSQL utiliza roles de IAM y políticas para la autorización del clúster. Asocie los roles de IAM con los roles de base de datos PostgreSQL para la autorización de la base de datos. En este enfoque se combinan los beneficios de IAM con los privilegios de PostgreSQL. Aurora DSQL utiliza estas características para proporcionar una autorización completa y una política de acceso para el clúster, la base de datos y los datos.

Administración del clúster mediante IAM

Para administrar el clúster, utilice IAM para la autenticación y la autorización:

Autenticación de IAM

Para autenticar la identidad de IAM cuando administre clústeres de Aurora DSQL, debe utilizar IAM. Puede proporcionar autenticación mediante la AWS Management Console, la AWS CLI o el AWS SDK.

Autorización de IAM

Para administrar clústeres de Aurora DSQL, conceda autorización mediante las acciones de IAM para Aurora DSQL. Por ejemplo, para crear un clúster, asegúrese de que la identidad de IAM tiene permisos para la acción de IAM `dsql:CreateCluster`, como en el siguiente ejemplo de acción de política.

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

Para obtener más información, consulte Uso de las acciones de política de IAM para administrar clústeres.

Conexión al clúster mediante IAM

Para conectarse al clúster, utilice IAM para la autenticación y la autorización:

Autenticación de IAM

Genere un token de autenticación temporal mediante una identidad de IAM con autorización para conectarse al clúster. Para obtener más información, consulte Generación de un token de autenticación en Amazon Aurora DSQL.

Autorización de IAM

Conceda las siguientes acciones de política de IAM a la identidad de IAM que esté utilizando para establecer la conexión con el punto de conexión del clúster:

- Use `dsql:DbConnectAdmin` si está utilizando el rol `admin`. Aurora DSQL crea y administra este rol por usted. El siguiente ejemplo de acción de política de IAM permite que `admin` se conecte a *`my-cluster`*.

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

- Use `dsql:DbConnect` si está utilizando un rol de base de datos personalizado. Este rol se crea y administra mediante comandos SQL en la base de datos. La acción de política de IAM de ejemplo siguiente permite que un rol de base de datos personalizado se conecte a *`my-cluster`* durante un máximo de una hora.

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

Después de establecer una conexión, el rol estará autorizado durante un máximo de una hora para la conexión.

Interacción con la base de datos mediante roles de base de datos PostgreSQL y roles de IAM

PostgreSQL administra los permisos de acceso a la base de datos con el concepto de roles. Un rol se puede considerar como un usuario de base de datos, o un grupo de usuarios de base de datos,

en función de cómo esté configurado el rol. Los roles de PostgreSQL se crean mediante comandos SQL. Para administrar la autorización de la base de datos, conceda permisos de PostgreSQL a los roles de base de datos PostgreSQL.

Aurora DSQL admite dos tipos de roles de base de datos: un rol `admin` y roles personalizados. Aurora DSQL crea automáticamente un rol `admin` predefinido para usted en el clúster de Aurora DSQL. No puede modificar el rol `admin`. Cuando se conecta a la base de datos como `admin`, puede emitir SQL para crear nuevos roles de base de datos para asociarlos con los roles de IAM. Para permitir que los roles de IAM se conecten a la base de datos, asocie los roles de base de datos personalizados con los roles de IAM.

Autenticación

Utilice el rol `admin` para conectarse al clúster. Después de conectar la base de datos, utilice el comando `AWS IAM GRANT` para asociar un rol de base de datos personalizado con la identidad de IAM autorizada para conectarse al clúster, como en el siguiente ejemplo.

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

Para obtener más información, consulte Autorización de roles de base de datos para conectarse al clúster.

Autorización

Utilice el rol `admin` para conectarse al clúster. Ejecute comandos SQL para establecer roles de base de datos personalizados y conceder permisos. Para obtener más información, consulte PostgreSQL database roles y PostgreSQL privileges en la documentación de PostgreSQL.

Utilizar acciones de política de IAM con Aurora DSQL

La acción de política de IAM que utilice depende del rol que utilice para conectarse al clúster: `admin` o un rol de base de datos personalizado. La política también depende de las acciones de IAM necesarias para este rol.

Uso de las acciones de política de IAM para conectarse a los clústeres

Cuando se conecte al clúster con el rol de base de datos predeterminado de `admin`, utilice una identidad de IAM con autorización para realizar la siguiente acción de política de IAM.

```
"dsql:DbConnectAdmin"
```

Cuando se conecte al clúster con un rol de base de datos personalizado, asocie primero el rol de IAM con el rol de base de datos. La identidad de IAM que utilice para conectarse al clúster debe tener autorización para realizar la siguiente acción de política de IAM.

```
"dsql:DbConnect"
```

Para obtener más información sobre los roles de base de datos personalizados, consulte Uso de roles de base de datos y autenticación de IAM.

Uso de las acciones de política de IAM para administrar clústeres

Cuando administre los clústeres de Aurora DSQL, especifique acciones de política solo para las acciones que el rol necesite realizar. Por ejemplo, si el rol solo necesita obtener información del clúster, podría limitar los permisos del rol solo a los permisos `GetCluster` y `ListClusters`, como en la siguiente política de ejemplo

JSON

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

En la siguiente política de ejemplo se muestran todas las acciones de política de IAM disponibles para administrar clústeres.

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" : "*"
    }
  ]
}
```

Revocación de autorización mediante IAM y PostgreSQL

Puede revocar los permisos de los roles de IAM para acceder a los roles de base de datos:

Revocación de la autorización de administrador para conectarse a los clústeres

Para revocar la autorización para conectarse al clúster con el rol `admin`, revoque el acceso de la identidad de IAM a `dsql:DbConnectAdmin`. Edite la política de IAM o desvincule la política de la identidad.

Después de revocar la autorización de conexión de la identidad de IAM, Aurora DSQL rechaza todos los nuevos intentos de conexión desde esa identidad de IAM. Las conexiones activas que utilicen la identidad de IAM pueden permanecer autorizadas mientras dure la conexión. Para obtener más información sobre las duraciones de las conexiones, consulte Cuotas y límites.

Revocación de la autorización de rol personalizado para conectarse a los clústeres

Para revocar el acceso a roles de base de datos distintos de `admin`, revoque el acceso de la identidad de IAM a `dsql:DbConnect`. Edite la política de IAM o desvincule la política de la identidad.

También puede eliminar la asociación entre el rol de base de datos e IAM mediante el comando `AWS IAM REVOKE` en la base de datos. Para obtener más información sobre la revocación del acceso de los roles de base de datos, consulte Revocación de la autorización de base de datos de un rol de IAM.

No puede administrar los permisos del rol de base de datos `admin` predefinido. Para obtener información sobre cómo administrar los permisos de los roles de base de datos personalizados, consulte PostgreSQL privileges. Las modificaciones de los privilegios surten efecto en la siguiente transacción después de que Aurora DSQL confirme correctamente la transacción de modificación.

Generación de un token de autenticación en Amazon Aurora DSQL

Para conectarse a Amazon Aurora DSQL con un cliente de SQL, genere un token de autenticación para utilizarlo como contraseña. Este token se usa solo para autenticar la conexión. Una vez establecida la conexión, esta sigue siendo válida incluso si el token de autenticación caduca.

Si crea un token de autenticación mediante la consola de AWS, el token caduca automáticamente en una hora de forma predeterminada. Si utiliza la AWS CLI o los SDK para crear el token, el tiempo predeterminado es de 15 minutos. La duración máxima es de 604 800 segundos, lo que equivale a una semana. Para volver a conectarse a Aurora DSQL desde el cliente, puede utilizar el mismo token de autenticación si no ha caducado o puede generar uno nuevo.

Para empezar a generar un token, cree una política de IAM y un clúster en Aurora DSQL. A continuación, use la consola de AWS, la AWS CLI o el SDK de AWS para generar un token.

Como mínimo, debe tener los permisos de IAM indicados en Conexión al clúster mediante IAM, según el rol de base de datos que utilice para conectarse.

Temas

- Uso de la consola de AWS para generar un token de autenticación en Aurora DSQL
- Uso de AWS CloudShell para generar un token de autenticación en Aurora DSQL

- Uso de la AWS CLI para generar un token de autenticación en Aurora DSQL
- Uso de los SDK para generar un token en Aurora DSQL

Uso de la consola de AWS para generar un token de autenticación en Aurora DSQL

Aurora DSQL autentica a los usuarios con un token en lugar de una contraseña. Puede generar el token desde la consola.

Para generar un token de autenticación

1. Inicie sesión en la AWS Management Console y abra la consola de Aurora DSQL en https://console.aws.amazon.com/dsql.
2. Elija el ID del clúster para el que desea generar un token de autenticación. Si aún no ha creado un clúster, siga los pasos descritos en Paso 1: creación de un clúster de Aurora DSQL de una sola región o Paso 4: creación de un clúster de varias regiones.
3. Elija Conectar y, a continuación, seleccione Obtener token.
4. Elija si desea conectarse como `admin` o con un rol de base de datos personalizado.
5. Copie el token de autenticación generado y úselo para Acceso a Aurora DSQL con clientes de SQL.

Para obtener más información sobre los roles de base de datos personalizados e IAM en Aurora DSQL, consulte Autenticación y autorización.

Uso de AWS CloudShell para generar un token de autenticación en Aurora DSQL

Antes de poder generar un token de autenticación mediante AWS CloudShell, asegúrese de hacer lo siguiente:

- Cree un clúster de Aurora DSQL.
- Agregue un permiso para ejecutar la operación de Amazon S3 `get-object` para recuperar objetos de una Cuenta de AWS ajena a la organización. Para obtener más información, consulte la Guía del usuario de Amazon S3.

Generación de un token de autenticación mediante AWS CloudShell

1. Inicie sesión en la AWS Management Console y abra la consola de Aurora DSQL en https://console.aws.amazon.com/dsql.
2. En la parte inferior izquierda de la consola de AWS, elija AWS CloudShell.
3. Siga Instalación o actualización de la última versión de la AWS CLI para instalar la AWS CLI.

```
sudo ./aws/install --update
```

4. Ejecute el siguiente comando para generar un token de autenticación para el rol `admin`. Reemplace *`us-east-1`* por la región y *`your_cluster_endpoint`* por el punto de conexión de su propio clúster.

> **Note**
>
> Si no se conecta como `admin`, utilice `generate-db-connect-auth-token` en su lugar.

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

Si tiene problemas, consulte Solución de problemas de IAM y ¿Cómo puedo solucionar los errores de acceso denegado u operación no autorizada con una política de IAM?

5. Utilice el siguiente comando para usar `psql` e iniciar una conexión con el clúster.

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

6. Debe ver una petición para proporcionar una contraseña. Copie el token que ha generado y asegúrese de no incluir espacios ni caracteres adicionales. Péguelo en la siguiente petición de `psql`.

```
Password for user admin:
```

7. Pulse Intro. Debe ver una petición de PostgreSQL.

```
postgres=>
```

Si obtiene un error de acceso denegado, asegúrese de que la identidad de IAM tiene el permiso `dsql:DbConnectAdmin`. Si tiene el permiso y sigue teniendo errores de acceso denegado, consulte Solución de problemas de IAM y ¿Cómo puedo solucionar los errores de acceso denegado u operación no autorizada con una política de IAM?

Para obtener más información sobre los roles de base de datos personalizados e IAM en Aurora DSQL, consulte Autenticación y autorización.

Uso de la AWS CLI para generar un token de autenticación en Aurora DSQL

Cuando el clúster esté `ACTIVE`, puede generar un token de autenticación en la CLI mediante el comando `aws dsql`. Utilice cualquiera de las siguientes técnicas:

- Si se conecta con el rol `admin`, utilice la opción `generate-db-connect-admin-auth-token`.
- Si se conecta con un rol de base de datos personalizado, utilice la opción `generate-db-connect-auth-token`.

En el siguiente ejemplo se utilizan los siguientes atributos para generar un token de autenticación para el rol `admin`.

- *`your_cluster_endpoint`*: el punto de conexión del clúster. Sigue el formato *`your_cluster_identifier`*`.dsql.`*`region`*`.on.aws`, como en el ejemplo `01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`.
- *`region`*: la Región de AWS, como `us-east-2` o `us-east-1`.

Los siguientes ejemplos establecen el tiempo de caducidad del token en 3600 segundos (1 hora).

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
```

Uso de los SDK para generar un token en Aurora DSQL

Puede generar un token de autenticación para el clúster cuando se encuentre en el estado `ACTIVE`. En los ejemplos de SDK se utilizan los siguientes atributos para generar un token de autenticación para el rol `admin`:

- *`your_cluster_endpoint`* (o *`yourClusterEndpoint`*): el punto de conexión del clúster de Aurora DSQL. El formato de nomenclatura es *`your_cluster_identifier`*`.dsql.`*`region`*`.on.aws`, como en el ejemplo `01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`.
- *`region`* (o *`RegionEndpoint`*): la Región de AWS en la que se encuentra el clúster, como `us-east-2` o `us-east-1`.

Python SDK

Puede generar el token de las siguientes formas:

- Si se conecta con el rol `admin`, utilice `generate_db_connect_admin_auth_token`.
- Si se conecta con un rol de base de datos personalizado, utilice `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

Puede generar el token de las siguientes formas:

- Si se conecta con el rol `admin`, utilice `GenerateDBConnectAdminAuthToken`.
- Si se conecta con un rol de base de datos personalizado, utilice `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

Puede generar el token de las siguientes formas:

- Si se conecta con el rol `admin`, utilice `getDbConnectAdminAuthToken`.
- Si se conecta con un rol de base de datos personalizado, utilice `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

Puede generar el token de las siguientes formas:

- Si se conecta con el rol `admin`, utilice `generateDbConnectAdminAuthToken`.
- Si se conecta con un rol de base de datos personalizado, utilice `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

Puede generar el token de las siguientes formas:

- Si se conecta con el rol `admin`, utilice `db_connect_admin_auth_token`.
- Si se conecta con un rol de base de datos personalizado, utilice `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

Puede generar el token de las siguientes formas:

- Si se conecta con el rol `admin`, utilice `generate_db_connect_admin_auth_token`.
- Si se conecta con un rol de base de datos personalizado, utilice `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**
>
> El SDK oficial para .NET no incluye una llamada a la API integrada para generar un token de autenticación para Aurora DSQL. En su lugar, debe utilizar `DSQLAuthTokenGenerator`, que es una clase de utilidad. En el siguiente ejemplo de código se muestra cómo generar el token de autenticación para .NET.

Puede generar el token de las siguientes formas:

- Si se conecta con el rol `admin`, utilice `DbConnectAdmin`.
- Si se conecta con un rol de base de datos personalizado, utilice `DbConnect`.

En el siguiente ejemplo se utiliza la clase de utilidad `DSQLAuthTokenGenerator` para generar el token de autenticación para un usuario con el rol `admin`. Reemplace *`insert-dsql-cluster-endpoint`* por el punto de conexión del clúster.

```
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**
>
> El SDK de Golang no proporciona un método integrado para generar un token prefirmado. Debe construir de forma manual la solicitud firmada, como se muestra en el siguiente ejemplo de código.

En el siguiente ejemplo de código, especifique la `action` en función del usuario de PostgreSQL:

- Si se conecta con el rol `admin`, utilice la acción `DbConnectAdmin`.
- Si se conecta con un rol de base de datos personalizado, utilice la acción `DbConnect`.

Además de *`yourClusterEndpoint`* y *`region`*, en el siguiente ejemplo se utiliza *`action`*. Especifique en *`action`* la acción en función del usuario de 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
}
```

Uso de roles de base de datos y autenticación de IAM

Aurora DSQL admite la autenticación mediante roles de IAM y usuarios de IAM. Puede usar cualquiera de los métodos para autenticarse y acceder a las bases de datos de Aurora DSQL.

Roles de IAM

Un rol de IAM es una identidad dentro de la Cuenta de AWS que tiene permisos específicos pero no está asociada a una persona específica. El uso de roles de IAM proporciona credenciales de seguridad temporales. Puede asumir temporalmente un rol de IAM de varias maneras:

- Mediante el cambio de roles en la AWS Management Console
- Mediante una llamada a una operación de API de AWS CLI o AWS
- Mediante una URL personalizada

Tras asumir un rol, puede acceder a Aurora DSQL con las credenciales temporales del rol. Para obtener más información sobre los métodos para el uso de roles, consulte Identidades de IAM en la Guía del usuario de IAM.

Usuarios de IAM

Un usuario de IAM es una identidad dentro de la Cuenta de AWS que tiene permisos específicos y está asociada a una sola persona o aplicación. Los usuarios de IAM tienen credenciales de larga duración, como contraseñas y claves de acceso, que se pueden utilizar para acceder a Aurora DSQL.

> **Note**
>
> Para ejecutar comandos SQL con la autenticación de IAM, puede utilizar los ARN del rol de IAM o los ARN del usuario de IAM en los ejemplos siguientes.

Autorización de roles de base de datos para conectarse al clúster

Cree un rol de IAM y conceda la autorización de conexión con la acción de política de IAM: `dsql:DbConnect`.

La política de IAM también debe conceder permiso para acceder a los recursos del clúster. Utilice un comodín (*) o siga las instrucciones en Uso de claves de condición de IAM con Amazon Aurora DSQL.

Autorización de roles de base de datos para utilizar SQL en la base de datos

Debe utilizar un rol de IAM con autorización para conectarse al clúster.

1. Conéctese al clúster de Aurora DSQL con una utilidad SQL.

   Utilice el rol de base de datos `admin` con una identidad de IAM que esté autorizada para la acción de IAM `dsql:DbConnectAdmin` para conectarse al clúster.

2. Cree un nuevo rol de base de datos y asegúrese de especificar la opción `WITH LOGIN`.

```
CREATE ROLE example WITH LOGIN;
```

3. Asocie el rol de base de datos con el ARN del rol de IAM.

```
AWS IAM GRANT example TO 'arn:aws:iam::012345678912:role/example';
```

4. Concesión de permisos de base de datos al rol de base de datos

   En los siguientes ejemplos se utiliza el comando `GRANT` para proporcionar autorización dentro de la base de datos.

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

Para obtener más información, consulte PostgreSQL GRANT y Privilegios de PostgreSQL en la documentación de PostgreSQL.

Revocación de la autorización de base de datos de un rol de IAM

Para revocar la autorización de base de datos, utilice la operación `AWS IAM REVOKE`.

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

Para obtener más información sobre la revocación de la autorización, consulte Revocación de autorización mediante IAM y PostgreSQL.

Aurora DSQL y PostgreSQL

Aurora DSQL es una base de datos relacional distribuida, compatible con PostgreSQL, diseñada para cargas de trabajo transaccionales. Aurora DSQL utiliza componentes principales de PostgreSQL, como el analizador, el planificador, el optimizador y el sistema de tipos.

El diseño de Aurora DSQL garantiza que toda la sintaxis de PostgreSQL admitida proporcione un comportamiento compatible y arroje resultados de consulta idénticos. Por ejemplo, Aurora DSQL proporciona conversiones de tipo, operaciones aritméticas y precisión y escala numéricas que son idénticas a PostgreSQL. Cualquier desviación queda documentada.

Aurora DSQL también incorpora capacidades avanzadas como el control de simultaneidad optimizada y la administración de esquemas distribuida. Con estas características, puede utilizar las herramientas familiares de PostgreSQL mientras se beneficia del rendimiento y la escalabilidad necesarios para las aplicaciones modernas, distribuidas y nativas en la nube.

Aspectos destacados de la compatibilidad con PostgreSQL

Aurora DSQL se basa actualmente en la versión 16 de PostgreSQL. Entre las principales compatibilidades se incluyen las siguientes:

Protocolo de conexión

Aurora DSQL utiliza el protocolo de conexión estándar de PostgreSQL v3. Esto habilita la integración con clientes, controladores y herramientas de PostgreSQL estándar. Por ejemplo, Aurora DSQL es compatible con `psql`, `pgjdbc` y `psycopg`.

Compatibilidad con SQL

Aurora DSQL es compatible con un amplio intervalo de expresiones y funciones de PostgreSQL estándar que se utilizan habitualmente en cargas de trabajo transaccionales. Las expresiones SQL admitidas producen resultados idénticos a los de PostgreSQL, incluidos los siguientes:

- Gestión de valores nulos
- Comportamiento del orden de clasificación
- Escala y precisión para operaciones numéricas
- Equivalencia para operaciones con cadenas

Para obtener más información, consulte Compatibilidad con características SQL en Aurora DSQL.

Administración de transacciones

Aurora DSQL conserva las principales características de PostgreSQL, como las transacciones ACID y un nivel de aislamiento equivalente a la lectura repetible de PostgreSQL. Para obtener más información, consulte Control de simultaneidad en Aurora DSQL.

Diferencias arquitectónicas clave

El diseño distribuido sin recursos compartidos de Aurora DSQL da como resultado algunas diferencias fundacionales con respecto a PostgreSQL tradicional. Estas diferencias forman parte integral de la arquitectura de Aurora DSQL y proporcionan muchas ventajas en cuanto a rendimiento y escalabilidad. Entre las diferencias clave se incluyen las siguientes:

Control de simultaneidad optimista (OCC)

Aurora DSQL utiliza un modelo de control de simultaneidad optimista. Este enfoque sin bloqueos impide que las transacciones se bloqueen entre sí, elimina los bloqueos y habilita una ejecución paralela de alto rendimiento. Estas características hacen que Aurora DSQL tenga especial valor en aplicaciones que requieren un rendimiento coherente a escala. Para obtener más ejemplos, consulte Control de simultaneidad en Aurora DSQL.

Operaciones DDL asíncronas

Aurora DSQL ejecuta las operaciones DDL de forma asíncrona, lo que permite lecturas y escrituras ininterrumpidas durante los cambios de esquema. Su arquitectura distribuida permite a Aurora DSQL realizar las siguientes acciones:

- Ejecutar operaciones DDL como tareas en segundo plano, lo que minimiza las interrupciones.
- Coordinar los cambios de catálogo como transacciones distribuidas de alta coherencia. Esto garantiza visibilidad atómica en todos los nodos, incluso durante errores u operaciones simultáneas.
- Operar de forma totalmente distribuida y sin nodo principal en múltiples zonas de disponibilidad con capas de computación y almacenamiento desacopladas.

Para obtener más información, consulte DDL y las transacciones distribuidas en Aurora DSQL.

Compatibilidad con características SQL en Aurora DSQL

Aurora DSQL y PostgreSQL devuelven resultados idénticos para todas las consultas SQL. Tenga en cuenta que Aurora DSQL se diferencia de PostgreSQL sin una cláusula `ORDER BY`. En las siguientes secciones, conozca la compatibilidad de Aurora DSQL con los tipos de datos y comandos SQL de PostgreSQL.

Temas

Tipos de datos admitidos en Aurora DSQL

Aurora DSQL admite un subconjunto de los tipos comunes de PostgreSQL.

Temas

Tipos de datos numéricos

Aurora DSQL admite los siguientes tipos de datos numéricos de PostgreSQL.

Nombre	Alias	Intervalo y precisión	Tamaño del almacenamiento	Compatibilidad con índices
`smallint`	`int2`	De -32768 a +32767	2 bytes	Sí
`integer`	`int`, `int4`	De -2147483648 a 2147483647	4 bytes	Sí

Nombre	Alias	Intervalo y precisión	Tamaño del almacenamiento	Compatibilidad con índices
`bigint`	`int8`	De -9223372036854775808 a +9223372036854775807	8 bytes	Sí
`real`	`float4`	Precisión de 6 dígitos decimales	4 bytes	Sí
`double precision`	`float8`	Precisión de 15 dígitos decimales	8 bytes	Sí

`numeric [ (p, s) ]`	`decimal [ (p, s) ]`
	
	`dec[ (p,s)]`
		Numérico exacto de precisión seleccionable. La precisión máxima es 38 y la escala máxima es 37.[1] El valor predeterminado es `numeric (18,6)`.	8 bytes + 2 bytes por dígito de precisión. El tamaño máximo es de 27 bytes.	No

[1]: si no especifica explícitamente un tamaño cuando ejecuta `CREATE TABLE` o `ALTER TABLE ADD COLUMN`, Aurora DSQL aplica los valores predeterminados. Aurora DSQL aplica límites cuando ejecuta las instrucciones `INSERT` o `UPDATE`.

Tipos de datos de caracteres

Aurora DSQL admite los siguientes tipos de datos de caracteres de PostgreSQL.

Nombre	Alias	Descripción	Límite de Aurora DSQL	Tamaño del almacenamiento	Compatibilidad con índices
`character [ (n) ]`	`char [ (n) ]`	Cadena de caracteres de longitud fija	4096 bytes[1]	Variable hasta 4100 bytes	Sí

Nombre	Alias	Descripción	Límite de Aurora DSQL	Tamaño del almacenamiento	Compatibilidad con índices
`character varying [ (n) ]`	`varchar [ (n) ]`	Cadena de caracteres de longitud variable	65 535 bytes[1]	Variable hasta 65539 bytes	Sí
`bpchar [ (n) ]`		Si es de longitud fija, es un alias de `char`. Si es de longitud variable, es un alias de `varchar`, donde los espacios al final no tienen significado semántico.	4096 bytes[1]	Variable hasta 4100 bytes	Sí
`text`		Cadena de caracteres de longitud variable	1 MiB[1]	Variable de hasta 1 MiB	Sí

[1]: si no especifica explícitamente un tamaño cuando ejecuta `CREATE TABLE` o `ALTER TABLE ADD COLUMN`, Aurora DSQL aplica los valores predeterminados. Aurora DSQL aplica límites cuando ejecuta las instrucciones `INSERT` o `UPDATE`.

Tipos de datos de fecha y hora

Aurora DSQL admite los siguientes tipos de datos de fecha y hora de PostgreSQL.

Nombre	Alias	Descripción	Range	Resolución	Tamaño del almacenamiento	Compatibilidad con índices
`date`		Fecha de calendario	De 4713 a. C. a 5874897 d. C.	1 día	4 bytes	Sí

Nombre	Alias	Descripción	Range	Resolución	Tamañ del almace iento	Compatibi lidad con índices
		(año, mes, día)				
`time[(p)][without time zone]`	`time`	Hora del día, sin zona horaria	De 0 a 1	1 microsegundo	8 bytes	Sí
`time[(p)] with time zone`	`timet`	Hora del día, con zona horaria	De 00:00:00+1559 a 24:00:00 -1559	1 microsegundo	12 byte	No
`timestamp [(p)][ without time zone]`		Fecha y hora, sin zona horaria	De 4713 a. C. a 294276 d. C.	1 microsegundo	8 bytes	Sí
`timestamp [(p)]with time zone`	`times tz`	Fecha y hora, con zona horaria	De 4713 a. C. a 294276 d. C.	1 microsegundo	8 bytes	Sí
`interval [fields][(p)]`		Intervalo de tiempo	De -178000000 años a 178000000 años	1 microsegundo	16 byte	No

Tipos de datos varios

Aurora DSQL soporta los siguientes tipos de datos varios de PostgreSQL.

Nombre	Alias	Descripción	Límite de Aurora DSQL	Tamaño del almacenamiento	Compatibilidad con índices
`boolean`	`bool`	Booleano lógico (true/false)		1 byte	Sí
`bytea`		Datos binarios (“matriz de bytes”)	1 MiB[1]	Variable hasta un límite de 1 MiB	No
`UUID`		Identificador único universal		16 bytes	Sí

[1]: si no especifica explícitamente un tamaño cuando ejecuta `CREATE TABLE` o `ALTER TABLE ADD COLUMN`, Aurora DSQL aplica los valores predeterminados. Aurora DSQL aplica límites cuando ejecuta las instrucciones `INSERT` o `UPDATE`.

Tipos de datos de tiempo de ejecución de consultas

Los tipos de datos de tiempo de ejecución de consultas son tipos de datos internos que se utilizan en tiempo de ejecución de consultas. Estos tipos son distintos de los tipos compatibles con PostgreSQL como `varchar` y `integer` que define en el esquema. En su lugar, estos tipos son representaciones en tiempo de ejecución que Aurora DSQL utiliza al procesar una consulta.

Los siguientes tipos de datos son compatibles solo durante el tiempo de ejecución de consultas:

Tipo de matriz

Aurora DSQL admite matrices de los tipos de datos admitidos. Por ejemplo, puede tener una matriz de enteros. La función `string_to_array` divide una cadena en una matriz al estilo de PostgreSQL mediante el delimitador de comas (`,`) como se muestra en el ejemplo siguiente. Puede utilizar matrices en expresiones, salidas de funciones o cálculos temporales durante la ejecución de consultas.

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

La función devuelve una respuesta similar a lo siguiente:

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

Tipo inet

Este tipo de datos representa direcciones de host IPv4, IPv6 y las subredes. Este tipo es útil al analizar registros, filtrar por subredes IP o realizar cálculos de red en una consulta. Para obtener más información, consulte inet en la documentación de PostgreSQL.

SQL compatible con Aurora DSQL

Aurora DSQL admite un amplio intervalo de características de SQL de PostgreSQL básicas. En las siguientes secciones, podrá conocer la compatibilidad general con expresiones PostgreSQL. Esta lista no es exhaustiva.

SELECT command

Aurora DSQL admite las siguientes cláusulas del comando `SELECT`.

Cláusula principal	Cláusulas admitidas
`FROM`	
`GROUP BY`	`ALL`, `DISTINCT`
`ORDER BY`	`ASC`, `DESC`, `NULLS`
`LIMIT`	
`DISTINCT`	
`HAVING`	

Cláusula principal	Cláusulas admitidas
USING	
WITH (expresiones de tabla comunes)	
INNER JOIN	ON
OUTER JOIN	LEFT, RIGHT, FULL, ON
CROSS JOIN	ON
UNION	ALL
INTERSECT	ALL
EXCEPT	ALL
OVER	RANK (), PARTITION BY
FOR UPDATE	

Lenguaje de definición de datos (DDL)

Aurora DSQL admite los siguientes comandos DDL de PostgreSQL.

Comando	Cláusula principal	Cláusulas admitidas
CREATE	TABLE	Para obtener información sobre la sintaxis admitida del comando CREATE TABLE, consulte CREATE TABLE.
ALTER	TABLE	Para obtener información sobre la sintaxis admitida del comando ALTER TABLE, consulte ALTER TABLE.
DROP	TABLE	

Comando	Cláusula principal	Cláusulas admitidas
CREATE	[UNIQUE] INDEX ASYNC	Puede utilizar este comando con los siguientes parámetros: ON, NULLS FIRST y NULLS LAST. Para obtener información sobre la sintaxis admitida del comando CREATE INDEX ASYNC, consulte Índices asíncronos en Aurora DSQL.
DROP	INDEX	
CREATE	VIEW	Para obtener más información sobre la sintaxis admitida del comando CREATE VIEW, consulte CREATE VIEW.
ALTER	VIEW	Para obtener información sobre la sintaxis admitida del comando ALTER VIEW, consulte ALTER VIEW.
DROP	VIEW	Para obtener información sobre la sintaxis admitida del comando DROP VIEW, consulte DROP VIEW.
CREATE	ROLE, WITH	
CREATE	FUNCTION	LANGUAGE SQL
CREATE	DOMAIN	

Lenguaje de manipulación de datos (DML)

Aurora DSQL admite los siguientes comandos DML de PostgreSQL.

Comando	Cláusula principal	Cláusulas admitidas
INSERT	INTO	VALUES

Comando	Cláusula principal	Cláusulas admitidas
		SELECT
UPDATE	SET	WHERE (SELECT) FROM, WITH
DELETE	FROM	USING, WHERE

Lenguaje de control de datos (DCL)

Aurora DSQL admite los siguientes comandos DCL de PostgreSQL.

Comando	Cláusulas admitidas
GRANT	ON, TO
REVOKE	ON, FROM, CASCADE, RESTRICT

Lenguaje de control de transacciones (TCL)

Aurora DSQL admite los siguientes comandos TCL de PostgreSQL.

Comando	Cláusulas admitidas
COMMIT	
BEGIN	[WORK | TRANSACTION] [READ ONLY | READ WRITE]

Comandos de utilidad

Aurora DSQL admite los siguientes comandos de utilidad de PostgreSQL:

- EXPLAIN
- ANALYZE (solo el nombre de la relación)

Subconjuntos de comandos SQL admitidos en Aurora DSQL

Esta sección de PostgreSQL proporciona información detallada sobre las expresiones compatibles, centrándose en los comandos con amplios conjuntos de parámetros y subcomandos. Por ejemplo, CREATE TABLE en PostgreSQL ofrece muchas cláusulas y parámetros. En esta sección se describen todos los elementos de la sintaxis de PostgreSQL que Aurora DSQL admite para estos comandos.

Temas

CREATE TABLE

CREATE TABLE define una nueva tabla.

```
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 cambia la definición de una tabla.

```
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 define una nueva vista persistente. Aurora DSQL no admite vistas temporales; solo admite vistas permanentes.

Sintaxis admitida

```
CREATE [ OR REPLACE ] [ RECURSIVE ] VIEW name [ ( column_name [, ...] ) ]
    [ WITH ( view_option_name [= view_option_value] [, ... ] ) ]
```

```
    AS query
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

Descripción

`CREATE VIEW` define una vista de una consulta. La vista no está materializada físicamente. En su lugar, la consulta se ejecuta cada vez que se hace referencia a la vista en una consulta.

`CREATE or REPLACE VIEW` es similar, pero, si ya existe una vista con el mismo nombre, se reemplaza. La nueva consulta debe generar las mismas columnas que generó la consulta de la vista existente (es decir, los mismos nombres de columna en el mismo orden y con los mismos tipos de datos), pero puede agregar columnas adicionales al final de la lista. Los cálculos que dan lugar a las columnas de salida pueden ser diferentes.

Si se indica un nombre de esquema, como `CREATE VIEW myschema.myview ...`), la vista se crea en el esquema especificado. En caso contrario, se crea en el esquema actual.

El nombre de la vista debe ser distinto del nombre de cualquier otra relación (tabla, índice o vista) en el mismo esquema.

Parámetros

`CREATE VIEW` admite varios parámetros para controlar el comportamiento de las vistas actualizables automáticamente.

`RECURSIVE`

Crea una vista recursiva. La sintaxis `CREATE RECURSIVE VIEW [ schema . ] view_name (column_names) AS SELECT ...;` es equivalente a `CREATE VIEW [ schema . ] view_name AS WITH RECURSIVE view_name (column_names) AS (SELECT ...) SELECT column_names FROM view_name;`.

Para una vista recursiva, debe especificarse una lista de nombres de columna de vista.

`name`

El nombre de la vista que se va a crear, que puede estar opcionalmente cualificado por el esquema. Para una vista recursiva, debe especificarse una lista de nombres de columna.

`column_name`

Una lista opcional de nombres que se utilizarán para las columnas de la vista. Si no se indican, los nombres de columna se deducen de la consulta.

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

Esta cláusula especifica parámetros opcionales para una vista; se admiten los siguientes parámetros.

- `check_option (enum)`: este parámetro puede ser `local` o `cascaded`, y equivale a especificar `WITH [ CASCADED | LOCAL ] CHECK OPTION`.
- `security_barrier (boolean)`: se debe utilizar si se pretende que la vista proporcione seguridad en la fila. Aurora DSQL no admite actualmente la seguridad en la fila, pero esta opción aún forzará a que las condiciones `WHERE` de la vista (y cualquier condición que utilice operadores marcados como `LEAKPROOF`) se evalúen primero.
- `security_invoker (boolean)`: esta opción hace que las relaciones base subyacentes se comprueben con los privilegios del usuario de la vista en lugar del propietario de la vista. Consulte las notas siguientes para obtener todos los detalles.

Todas las opciones anteriores pueden modificarse en las vistas existentes mediante `ALTER VIEW`.

`query`

Un comando `SELECT` o `VALUES` que proporcionará las columnas y filas de la vista.

- `WITH [ CASCADED | LOCAL ] CHECK OPTION`: esta opción controla el comportamiento de las vistas actualizables automáticamente. Cuando se especifica esta opción, los comandos `INSERT` y `UPDATE` de la vista se comprobarán para asegurarse de que las nuevas filas satisfacen la condición que define la vista (es decir, se comprueba que las nuevas filas son visibles a través de la vista). Si no lo son, se rechazará la actualización. Si no se especifica `CHECK OPTION`, se permite que los comandos `INSERT` y `UPDATE` en la vista creen filas que no son visibles a través de la vista. Se admiten las siguientes opciones de comprobación.
- `LOCAL`: las nuevas filas solo se comprueban con las condiciones definidas directamente en la propia vista. Las condiciones definidas en las vistas base subyacentes no se comprueban (a menos que también especifiquen `CHECK OPTION`).
- `CASCADED`: las filas nuevas se comprueban con las condiciones de la vista y de todas las vistas base subyacentes. Si se especifica `CHECK OPTION` y no se especifican `LOCAL` ni `CASCADED`, entonces se supone `CASCADED`.

> **Note**
>
> `CHECK OPTION` no se puede usar con vistas `RECURSIVE`. `CHECK OPTION` solo se admite en las vistas que se actualizan automáticamente.

Notas

Utilice la instrucción `DROP VIEW` para descartar vistas.

Los nombres y tipos de datos de las columnas de la vista deben considerarse cuidadosamente. Por ejemplo, no se recomienda CREATE VIEW vista AS SELECT 'Hello World'; ya que el nombre de la columna está predeterminado como `?column?;`. Además, el tipo de datos de la columna es `text` de forma predeterminada, que puede no ser lo que se deseaba.

Un enfoque mejor es especificar explícitamente el nombre de la columna y el tipo de datos, como por ejemplo: `CREATE VIEW vista AS SELECT text 'Hello World' AS hello;`.

De forma predeterminada, el acceso a las relaciones base subyacentes a las que se hace referencia en la vista viene determinado por los permisos del propietario de la vista. En algunos casos, esto puede utilizarse para proporcionar un acceso seguro pero restringido a las tablas subyacentes. No obstante, no todas las vistas están protegidas contra la manipulación.

- Si la vista tiene la propiedad `security_invoker` establecida en true, el acceso a las relaciones base subyacentes viene determinado por los permisos del usuario que ejecuta la consulta, en lugar de por el propietario de la vista. Así, el usuario de una vista de invocación de seguridad debe tener los permisos pertinentes en la vista y las relaciones base subyacentes.
- Si alguna de las relaciones base subyacentes es una vista de invocación de seguridad, se tratará como si se hubiera accedido a ella directamente desde la consulta original. Así, una vista de invocación de seguridad siempre comprobará las relaciones base subyacentes mediante los permisos del usuario actual, incluso si se accede a ella desde una vista sin la propiedad `security_invoker`.
- Las funciones a las que se llama en la vista se tratan igual que si se hubieran llamado directamente desde la consulta que utiliza la vista. Por lo tanto, el usuario de una vista debe tener permisos para llamar a todas las funciones que utiliza la vista. Las funciones en la vista se ejecutan con los privilegios del usuario que ejecuta la consulta o del propietario de la función, dependiendo de si las funciones están definidas como `SECURITY INVOKER` o `SECURITY DEFINER`. Por ejemplo, llamar a `CURRENT_USER` directamente en una vista siempre devolverá

el usuario que hace la invocación, no el propietario de la vista. Esto no se ve afectado por la configuración `security_invoker` de la vista, por lo que una vista con `security_invoker` establecido en false no es equivalente a una función `SECURITY DEFINER`.

- El usuario que crea o reemplaza una vista debe tener privilegios `USAGE` en cualquier esquema al que se haga referencia en la consulta de la vista, para poder buscar los objetos a los que se hace referencia en esos esquemas. No obstante, tenga en cuenta que esta búsqueda solo se produce cuando se crea o reemplaza la vista. Por lo tanto, el usuario de la vista solo necesita el privilegio `USAGE` sobre el esquema que contiene la vista, no sobre los esquemas a los que se hace referencia en la consulta de la vista, incluso para una vista de invocación de seguridad.
- Cuando se utiliza `CREATE OR REPLACE VIEW` en una vista existente, solo se modifica la regla `SELECT` de definición de la vista, además de cualquier parámetro `WITH ( ... )` y `CHECK OPTION`. Las demás propiedades de la vista, incluidas la propiedad, los permisos y las reglas no SELECT, permanecen sin alterarse. Debe tener la propiedad de la vista para reemplazarla (esto incluye ser miembro del rol de propietario).

Vistas actualizables

Las vistas simples son actualizables automáticamente: el sistema permitirá que se utilicen las instrucciones `INSERT`, `UPDATE` y `DELETE` en la vista del mismo modo que en una tabla normal. Una vista es actualizable automáticamente si cumple todas las condiciones siguientes:

- La vista debe tener exactamente una entrada en la lista `FROM`, que debe ser una tabla u otra vista actualizable.
- La definición de la vista no debe contener las cláusulas `WITH`, `DISTINCT`, `GROUP BY`, `HAVING`, `LIMIT` o `OFFSET` en el nivel superior.
- La definición de la vista no debe contener operaciones de conjunto (`UNION`, `INTERSECT` o `EXCEPT`) en el nivel superior.
- La lista de selección de la vista no debe contener agregados, funciones de ventana ni funciones que devuelvan conjuntos.

Una vista actualizable automáticamente puede contener una mezcla de columnas actualizables y no actualizables. Una columna es actualizable si es una simple referencia a una columna actualizable de la relación base subyacente. En caso contrario, la columna es de solo lectura y se produce un error si una instrucción `INSERT` o `UPDATE` intenta asignarle un valor.

En el caso de las vistas actualizables automáticamente, el sistema convierte cualquier instrucción `INSERT`, `UPDATE` o `DELETE` de la vista en la instrucción correspondiente de la relación base subyacente. Las instrucciones `INSERT` con una cláusula `ON CONFLICT UPDATE` son totalmente compatibles.

Si una vista actualizable automáticamente contiene una condición `WHERE`, esta restringe las filas de la relación base que pueden modificar las instrucciones `UPDATE` y `DELETE` en la vista. No obstante, `UPDATE` puede modificar una fila de modo que ya no satisfaga la condición `WHERE`, lo que la hace invisible en la vista. Del mismo modo, un comando `INSERT` puede insertar potencialmente filas de la relación base que no satisfagan la condición`WHERE`, lo que las hace invisibles en la vista. `ON CONFLICT UPDATE` puede afectar de forma similar una fila existente no visible en la vista.

Puede utilizar `CHECK OPTION` para evitar que los comandos `INSERT` y`UPDATE` creen filas que no sean visibles en la vista.

Si una vista actualizable automáticamente está marcada con la propiedad security_barrier, todas las condiciones `WHERE` de la vista (y cualquier condición que utilice operadores marcados como `LEAKPROOF`) se evalúan siempre antes que cualquier condición que haya agregado un usuario de la vista. Tenga en cuenta que, debido a esto, las filas que finalmente no se devuelven (porque no superan las condiciones `WHERE` del usuario) pueden acabar bloqueadas. Puede utilizar `EXPLAIN` para ver qué condiciones se aplican a la relación (y por tanto no bloquean filas) y cuáles no.

Una vista más compleja que no satisfaga todas estas condiciones es de solo lectura de forma predeterminada: el sistema no permite una inserción, actualización o eliminación en la vista.

> **Note**
>
> El usuario que realiza la inserción, actualización o eliminación en la vista debe tener el correspondiente privilegio de inserción, actualización o eliminación en la vista. De forma predeterminada, el propietario de la vista debe tener los privilegios correspondientes en las relaciones base subyacentes, mientras que el usuario que realiza la actualización no necesita ningún permiso en las relaciones base subyacentes. No obstante, si la vista tiene security_invoker establecido en true, el usuario que realiza la actualización, en lugar del propietario de la vista, debe tener los privilegios pertinentes en las relaciones base subyacentes.

Ejemplos

Crear una vista compuesta por todas las películas de comedia.

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

Esto creará una vista que contiene las columnas que están en la tabla `film` en el momento de la creación de la vista. Aunque `*` se utilizó para crear la vista, las columnas agregadas posteriormente a la tabla no formarán parte de la vista.

Cree una vista con `LOCAL CHECK OPTION`.

```
CREATE VIEW pg_comedies AS
    SELECT *
    FROM comedies
    WHERE classification = 'PG'
    WITH CASCADED CHECK OPTION;
```

De este modo, se creará una vista que comprueba los valores `kind` y `classification` de las nuevas filas.

Cree una vista con una mezcla de columnas actualizables y no actualizables.

```
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';
```

Esta vista admitirá `INSERT`, `UPDATE` y `DELETE`. Todas las columnas de la tabla de películas serán actualizables, mientras que las columnas computadas `country` y `avg_rating` serán de solo lectura.

```
CREATE RECURSIVE VIEW public.nums_1_100 (n) AS
    VALUES (1)
```

```
UNION ALL
    SELECT n+1 FROM nums_1_100 WHERE n < 100;
```

> **Note**
>
> Aunque el nombre de la vista recursiva está cualificado por el esquema en `CREATE`, la autorreferencia interna no está cualificada por el esquema. Esto se debe a que el nombre de la expresión de tabla común (CTE) creada implícitamente no puede calificarse por el esquema.

Compatibilidad

`CREATE OR REPLACE VIEW` es una extensión del lenguaje PostgreSQL. La cláusula `WITH ( ... )` también es una extensión, al igual que las vistas de barrera de seguridad y las vistas de invocación de seguridad. Aurora DSQL admite estas extensiones de lenguaje.

ALTER VIEW

La instrucción `ALTER VIEW` permite cambiar varias propiedades de una vista existente y Aurora DSQL soporta toda la sintaxis de PostgreSQL para este comando.

Sintaxis admitida

```
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 [, ... ] )
```

Descripción

`ALTER VIEW` modifica varias propiedades auxiliares de una vista. (Si desea modificar la consulta que define la vista, utilice `CREATE OR REPLACE VIEW`). Debe ser propietario de la vista para utilizar `ALTER VIEW`. Para modificar el esquema de una vista, también debe tener el privilegio `CREATE` en el nuevo esquema. Para modificar el propietario, debe poder utilizar `SET ROLE` en el nuevo rol de propietario y ese rol debe tener el privilegio `CREATE` en el esquema de la vista. Estas restricciones

hacen que alterar el propietario no haga nada que no pudiera hacer con descartar y volver a crear la vista).

Parámetros

Parámetros `ALTER VIEW`

`name`

El nombre (opcionalmente cualificado por el esquema) de una vista existente.

`column_name`

Nuevo nombre para una columna existente.

`IF EXISTS`

No genere un error si la vista no existe. En este caso, se emite un aviso.

`SET/DROP DEFAULT`

Estas formas establecen o eliminan el valor predeterminado de una columna. El valor predeterminado para una columna de una vista se sustituye en cualquier comando `INSERT` o `UPDATE` donde el objetivo sea la vista. El valor predeterminado de la vista tendrá prioridad sobre cualquier valor predeterminado de las relaciones subyacentes.

new_owner

El nombre de usuario del nuevo propietario de la vista.

new_name

El nuevo nombre de la vista.

new_schema

El nuevo esquema de la vista.

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

Establece o restablece una opción de vista. A continuación, se muestran las opciones admitidas.

- `check_option (enum)`: cambia la opción de comprobación de la vista. El valor debe ser `local` o `cascaded`.
- `security_barrier (boolean)`: cambia la propiedad de barrera de seguridad de la vista. Debe ser un valor booleano, como `true` o `false`.

- `security_invoker (boolean)`: cambia la propiedad de barrera de seguridad de la vista. Debe ser un valor booleano, como `true` o `false`.

Notas

Por razones históricas de PostgreSQL, `ALTER TABLE` también puede utilizarse con vistas, pero las únicas variantes de `ALTER TABLE` que se permiten con vistas son equivalentes a las mostradas anteriormente.

Ejemplos

Cambio del nombre de la vista `foo` por `bar`.

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

Asociar un valor de columna predeterminado a una vista actualizable.

```
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
```

Compatibilidad

`ALTER VIEW` es una extensión de PostgreSQL del estándar SQL que Aurora DSQL admite.

DROP VIEW

La instrucción `DROP VIEW` elimina una vista existente. Aurora DSQL admite la sintaxis de PostgreSQL completa para este comando.

Sintaxis admitida

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

Descripción

`DROP VIEW` descarta una vista existente. Para ejecutar este comando, debe ser el propietario de la vista.

Parámetros

`IF EXISTS`

No genere un error si la vista no existe. En este caso, se emite un aviso.

`name`

El nombre (opcionalmente cualificado por el esquema) de la vista que se eliminará.

`CASCADE`

Eliminar automáticamente los objetos que dependan de la vista (como otras vistas) y, a su vez, todos los objetos que dependan de esos objetos.

`RESTRICT`

Rechazar el descarte de la vista si algún objeto depende de ella. Esta es la opción predeterminada.

Ejemplos

```
DROP VIEW kinds;
```

Compatibilidad

Este comando se ajusta al estándar SQL, excepto que el estándar solo permite descartar una vista por comando y, aparte de la opción `IF EXISTS`, que es una extensión de PostgreSQL que Aurora DSQL admite.

Características de PostgreSQL no admitidas en Aurora DSQL

Aurora DSQL es compatible con PostgreSQL. Esto significa que Aurora DSQL admite características relacionales básicas como transacciones ACID, índices secundarios, uniones, inserciones y actualizaciones. Para obtener información general de las características de SQL compatibles, consulte Expresiones SQL admitidas.

En las siguientes secciones se destacan las características de PostgreSQL que actualmente no se admiten en Aurora DSQL.

Objetos no admitidos

Entre los objetos no compatibles con Aurora DSQL se incluyen los siguientes:

- Varias bases de datos en un único clúster de Aurora DSQL
- Tablas temporales
- Desencadenadores
- Tipos (compatibilidad parcial)
- Espacios de tabla
- Funciones escritas en lenguajes distintos de SQL
- Secuencias
- Particiones

Restricciones no admitidas

- Claves externas
- Restricciones de exclusión

Comandos admitidos

- `ALTER SYSTEM`
- `TRUNCATE`
- `SAVEPOINT`
- `VACUUM`

> **Note**
>
> Aurora DSQL no requiere vaciado. El sistema mantiene las estadísticas y administra la optimización del almacenamiento automáticamente sin comandos de vaciado manuales.

Extensiones no admitidas

Aurora DSQL no admite las extensiones de PostgreSQL. La tabla siguiente muestra las extensiones que no se admiten:

- `PL/pgSQL`
- `PostGIS`

- PGVector
- PGAudit
- Postgres_FDW
- PGCron
- pg_stat_statements

Expresiones SQL no admitidas

En la siguiente tabla se describen las cláusulas que no se admiten en Aurora DSQL.

Categoría	Cláusula principal	Cláusula no admitida
CREATE	INDEX ASYNC	ASC | DESC
CREATE	INDEX [1]	
TRUNCATE		
ALTER	SYSTEM	Todos los comandos ALTER SYSTEM están bloqueados.
CREATE	TABLE	COLLATE, AS SELECT, INHERITS, PARTITION
CREATE	FUNCTION	LANGUAGE *non-sql-lang* , donde *non-sql-lang* es cualquier lenguaje distinto de SQL
CREATE	TEMPORARY	TABLES
CREATE	EXTENSION	
CREATE	SEQUENCE	
CREATE	MATERIALIZED	VIEW
CREATE	TABLESPACE	

Categoría	Cláusula principal	Cláusula no admitida
CREATE	TRIGGER	
CREATE	TYPE	
CREATE	DATABASE	No puede crear bases de datos adicionales.

[1] Consulte Índices asíncronos en Aurora DSQL para crear un índice en una columna de una tabla específica.

Consideraciones de Aurora DSQL para la compatibilidad de PostgreSQL

Tenga en cuenta las siguientes limitaciones de compatibilidad cuando utilice Aurora DSQL. Para obtener información general, consulte Consideraciones para trabajar con Amazon Aurora DSQL. Para obtener información acerca de las cuotas y los límites, consulte Cuotas de clúster y límites de base de datos en Amazon Aurora DSQL.

- Aurora DSQL utiliza una única base de datos integrada denominada `postgres`. No puede crear bases de datos adicionales ni cambiar el nombre o eliminar la base de datos de `postgres`.
- La base de datos de `postgres` utiliza la codificación de caracteres UTF-8. No puede cambiar la codificación.
- La base de datos usa solo la intercalación de `C`.
- Aurora DSQL utiliza `UTC` como la zona horaria del sistema. No puede modificar la zona horaria mediante parámetros o instrucciones SQL como `SET TIMEZONE`.
- El nivel de aislamiento de las transacciones se fija en PostgreSQL `Repeatable Read`.
- Las transacciones tienen las siguientes restricciones:
  - Una transacción no puede mezclar operaciones DDL y DML
  - Una transacción solo puede incluir 1 instrucción DDL
  - Una transacción puede modificar hasta 3000 filas, independientemente del número de índices secundarios
  - El límite de 3000 filas se aplica a todas las instrucciones DML (`INSERT`, `UPDATE`, `DELETE`)
- Las conexiones a la base de datos caducan después de 1 hora.

- En estos momentos, Aurora DSQL no le permite ejecutar `GRANT [permission] ON DATABASE`. Si intenta ejecutar esa instrucción, Aurora DSQL devuelve el mensaje de error `ERROR: unsupported object type in GRANT`.
- Aurora DSQL no permite que los roles de usuario que no son administradores ejecuten el comando `CREATE SCHEMA`. No puede ejecutar el comando `GRANT [permission] on DATABASE` ni conceder permisos `CREATE` en la base de datos. Si un rol de usuario que no es administrador intenta crear un esquema, Aurora DSQL devuelve el mensaje de error `ERROR: permission denied for database postgres`.
- Los usuarios que no son administradores no pueden crear objetos en el esquema público. Solo los usuarios administradores pueden crear objetos en el esquema público. El rol de usuario administrador tiene permisos para conceder acceso de lectura, escritura y modificación a estos objetos a los usuarios que no son administradores, pero no puede conceder permisos `CREATE` al propio esquema público. Los usuarios que no son administradores deben utilizar esquemas diferentes, creados por el usuario, para la creación de objetos.
- Aurora DSQL no admite el comando `ALTER ROLE [] CONNECTION LIMIT`. Póngase en contacto con el servicio de asistencia de AWS si necesita aumentar el límite de conexiones.
- Aurora DSQL no admite asyncpg, el controlador asíncrono de base de datos PostgreSQL para Python.

Control de simultaneidad en Aurora DSQL

La simultaneidad permite que varias sesiones accedan y modifiquen datos simultáneamente sin poner en riesgo la integridad y la coherencia de los datos. Aurora DSQL proporciona compatibilidad con PostgreSQL a la vez que implementa un moderno mecanismo de control de simultaneidad sin bloqueos. Mantiene el pleno cumplimiento de ACID mediante el aislamiento de instantáneas, lo que garantiza la coherencia y la fiabilidad de los datos.

Una ventaja clave de Aurora DSQL es la arquitectura sin bloqueo, que elimina los cuellos de botella habituales en el rendimiento de las bases de datos. Aurora DSQL impide que las transacciones lentas bloqueen otras operaciones y elimina el riesgo de bloqueos. Gracias a este enfoque, Aurora DSQL resulta especialmente valioso para aplicaciones de alto rendimiento en las que el rendimiento y la escalabilidad son fundamentales.

Conflictos de transacción

Aurora DSQL utiliza el control de simultaneidad optimista (OCC), que funciona de forma diferente a los sistemas tradicionales basados en bloqueos. En lugar de utilizar bloqueos, OCC evalúa los conflictos en el momento de la confirmación. Cuando varias transacciones entran en conflicto al actualizar la misma fila, Aurora DSQL administra las transacciones de la siguiente manera:

- Aurora DSQL procesa la transacción con el tiempo de confirmación más temprano.
- Las transacciones en conflicto reciben un error de serialización de PostgreSQL, lo que indica la necesidad de que se vuelvan a intentar.

Diseñe las aplicaciones para implementar la lógica de reintento para gestionar los conflictos. El patrón de diseño ideal es idempotente, lo que habilita el reintento de transacciones como primer recurso siempre que sea posible. La lógica recomendada es similar a la lógica de anular y reintentar en una situación estándar de tiempo de espera de bloqueo o interbloqueo de PostgreSQL. No obstante, OCC requiere que las aplicaciones ejerciten esta lógica con mayor frecuencia.

Directrices para optimizar el rendimiento de las transacciones

Para optimizar el rendimiento, minimice la alta contención en claves únicas o en intervalos de claves pequeños. Para alcanzar este objetivo, diseñe el esquema de forma que las actualizaciones se repartan por el intervalo de claves del clúster mediante las siguientes directrices:

- Elija una clave principal aleatoria para las tablas.
- Evite patrones que aumenten la contención en claves únicas. Este enfoque garantiza un rendimiento óptimo incluso a medida que crezca el volumen de transacciones.

DDL y las transacciones distribuidas en Aurora DSQL

En Aurora DSQL, el comportamiento del lenguaje de definición de datos (DDL) es distinto al de PostgreSQL. Aurora DSQL presenta una capa de base de datos distribuida y compartida Multi-AZ basada en flotas de computación y almacenamiento de varios inquilinos. Dado que no existe un único nodo principal o líder de base de datos, el catálogo de base de datos está distribuido. Por tanto, Aurora DSQL administra los cambios de esquema DDL como transacciones distribuidas.

En concreto, el comportamiento de DDL en Aurora DSQL es distinto según se indica a continuación:

Errores de control de simultaneidad

Aurora DSQL devuelve un error de infracción de control de simultaneidad si ejecuta una transacción mientras otra transacción actualiza un recurso. Por ejemplo, considere la siguiente secuencia de acciones:

1. En la sesión 1, un usuario agrega una columna a la tabla `mytable`.
2. En la sesión 2, un usuario intenta insertar una fila en `mytable`.

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

DDL y DML en la misma transacción

Las transacciones en Aurora DSQL pueden contener solo una instrucción DDL y no pueden tener tanto instrucciones DDL como DML. Esta restricción significa que no se puede crear una tabla e insertar datos en la misma tabla dentro de la misma transacción. Por ejemplo, Aurora DSQL admite las siguientes transacciones secuenciales.

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

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

Aurora DSQL no admite la siguiente transacción, que incluye instrucciones `CREATE` e `INSERT`.

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

DDL asíncrono

En PostgreSQL estándar, operaciones DDL como `CREATE INDEX` bloquean la tabla afectada, por lo que no está disponible para lecturas y escrituras desde otras sesiones. En Aurora DSQL, estas instrucciones DDL se ejecutan de forma asíncrona mediante un administrador en segundo

plano. El acceso a la tabla afectada no se bloquea. De este modo, las sentencias DDL en tablas de gran tamaño pueden ejecutarse sin tiempo de inactividad ni impacto en el rendimiento. Para obtener más información sobre el administrador de trabajos asíncronos en Aurora DSQL, consulte Índices asíncronos en Aurora DSQL.

Claves principales en Aurora DSQL

En Aurora DSQL, una clave principal es una característica que organiza físicamente los datos de las tablas. Es similar a la operación `CLUSTER` en PostgreSQL o a un índice en clúster en otras bases de datos. Cuando se define una clave principal, Aurora DSQL crea un índice que incluye todas las columnas de la tabla. La estructura de clave principal en Aurora DSQL garantiza que el acceso a los datos y la administración sean eficientes.

Estructura y almacenamiento de datos

Cuando define una clave principal, Aurora DSQL almacena los datos de la tabla en orden de clave principal. Esta estructura organizada en índices permite que una búsqueda de clave principal recupere todos los valores de las columnas directamente, en lugar de seguir un puntero a los datos como en un índice de árbol B tradicional. A diferencia de la operación `CLUSTER` en PostgreSQL, que reorganiza los datos solo una vez, Aurora DSQL mantiene este orden de forma automática y continua. Este enfoque mejora el rendimiento de las consultas que dependen del acceso a la clave principal.

Aurora DSQL también utiliza la clave principal para generar una clave única en todo el clúster para cada fila en las tablas y los índices. Esta clave única también sustenta la gestión de datos distribuidos. Habilita la partición automática de los datos en varios nodos, lo que permite un almacenamiento escalable y una alta simultaneidad. Como resultado, la estructura de clave principal ayuda a Aurora DSQL a escalar automáticamente y a administrar con eficiencia las cargas de trabajo simultáneas.

Directrices para elegir una clave principal

Al elegir y utilizar una clave principal en Aurora DSQL, tenga en cuenta las siguientes directrices:

- Defina una clave principal cuando cree una tabla. No podrá cambiar esta clave ni agregar una nueva clave principal más adelante. La clave principal pasa a formar parte de la clave de todo el clúster utilizada para la partición de datos y el escalado automático del rendimiento de escritura. Si no especifica una clave principal, Aurora DSQL asigna un ID oculto sintético.

- Para tablas con grandes volúmenes de escritura, evite utilizar números enteros que aumenten de forma monótona como claves principales. Esto puede provocar problemas de rendimiento al dirigir todas las nuevas inserciones a una única partición. En su lugar, utilice claves principales con distribución aleatoria para garantizar una distribución uniforme de las escrituras entre las particiones de almacenamiento.
- En el caso de las tablas que cambian con poca frecuencia o son de solo lectura, puede utilizar una clave ascendente. Ejemplos de claves ascendentes son las marcas temporales o los números de secuencia. Una clave densa tiene muchos valores poco espaciados o duplicados. Puede utilizar una clave ascendente aunque sea densa porque el rendimiento de escritura es menos crítico.
- Si un examen completo de la tabla no satisface los requisitos de rendimiento, elija un método de acceso más eficiente. En la mayoría de los casos, esto significa utilizar una clave principal que coincida con la clave de unión y búsqueda más común en las consultas.
- El tamaño máximo combinado de las columnas de una clave principal es de 1 kibibyte. Para obtener más información, consulte Límites de base de datos en Aurora DSQL y Tipos de datos admitidos en Aurora DSQL.
- Puede incluir hasta 8 columnas en una clave principal o un índice secundario. Para obtener más información, consulte Límites de base de datos en Aurora DSQL y Tipos de datos admitidos en Aurora DSQL.

Índices asíncronos en Aurora DSQL

El comando `CREATE INDEX ASYNC` crea un índice en una o más columnas de una tabla específica. Este comando es una operación DDL asíncrona que no bloquea otras transacciones. Cuando ejecute `CREATE INDEX ASYNC`, Aurora DSQL devuelve de forma inmediata un `job_id`.

Puede supervisar el estado de este trabajo asíncrono mediante la vista de sistema `sys.jobs`. Mientras el trabajo de creación de índices esté en curso, puede usar estos procedimientos y comandos:

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

Bloquea la sesión actual hasta que el trabajo especificado finalice o se produzca un error. Devuelve un valor booleano que indica éxito o error.

`DROP INDEX`

Cancela un trabajo de creación de índices en curso.

Cuando se completa la creación de índices asíncrona, Aurora DSQL actualiza el catálogo del sistema para marcar el índice como activo.

> **Note**
>
> Tenga en cuenta que las transacciones simultáneas que acceden a objetos en el mismo espacio de nombres durante esta actualización pueden encontrar errores de simultaneidad.

Cuando Aurora DSQL finaliza una tarea de índice asíncrona, actualiza el catálogo del sistema para mostrar que el índice está activo. Si otras transacciones hacen referencia a los objetos en el mismo espacio de nombres en ese momento, podría aparecer un error de simultaneidad.

Sintaxis

`CREATE INDEX ASYNC` utiliza la siguiente sintaxis.

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

Parámetros

`UNIQUE`

Indica a Aurora DSQL que compruebe si hay valores duplicados en la tabla cuando crea el índice y cada vez que agrega datos. Si especifica este parámetro, las operaciones de inserción y actualización que dan lugar a entradas duplicadas generan un error.

`IF NOT EXISTS`

Indica que Aurora DSQL no debe lanzar una excepción si ya existe un índice con el mismo nombre. En esta situación, Aurora DSQL no crea el nuevo índice. Tenga en cuenta que el índice que intenta crear podría tener una estructura muy diferente del índice que existe. Si especifica este parámetro, el nombre del índice es obligatorio.

`name`

El nombre del índice. No puede incluir el nombre del esquema en este parámetro.

Aurora DSQL crea el índice en el mismo esquema que la tabla principal. El nombre del índice debe ser distinto del nombre de cualquier otro objeto, como una tabla o un índice, en el esquema.

Si no especifica un nombre, Aurora DSQL genera un nombre automáticamente basándose en el nombre de la tabla principal y la columna indexada. Por ejemplo, si ejecuta `CREATE INDEX ASYNC on table1 (col1, col2)`, Aurora DSQL asigna automáticamente el nombre `table1_col1_col2_idx` al índice.

`NULLS FIRST | LAST`

El orden de clasificación de las columnas nulas y no nulas. `FIRST` indica que Aurora DSQL debe ordenar las columnas nulas antes que las columnas no nulas. `LAST` indica que Aurora DSQL debe ordenar las columnas nulas después de las columnas no nulas.

`INCLUDE`

Una lista de columnas para incluir en el índice como columnas no clave. No puede utilizar una columna no clave en una cualificación de búsqueda de examen de índice. Aurora DSQL ignora la columna en términos de unicidad para un índice.

`NULLS DISTINCT | NULLS NOT DISTINCT`

Especifica si Aurora DSQL debe considerar los valores nulos como distintos en un índice único. El valor predeterminado es `DISTINCT`, lo que significa que un índice único puede contener múltiples valores nulos en una columna. `NOT DISTINCT` indica que un índice no puede contener múltiples valores nulos en una columna.

Notas de uso

Tenga en cuenta estas directrices:

- El comando `CREATE INDEX ASYNC` no introduce bloqueos. Tampoco afecta la tabla base que Aurora DSQL utiliza para crear el índice.
- Durante las operaciones de migración de esquemas, el procedimiento `sys.wait_for_job(job_id)`*`'your_index_creation_job_id'`* resulta útil. Garantiza que las operaciones DDL y DML posteriores se dirijan al índice recién creado.
- Cada vez que Aurora DSQL ejecuta una nueva tarea asíncrona, comprueba la vista `sys.jobs` y elimina las tareas que tienen un estado de `completed` o `failed` durante más de 30 minutos. Así, `sys.jobs` muestra principalmente las tareas en curso y no contiene información sobre las tareas antiguas.

- Si Aurora DSQL no crea un índice asíncrono, el índice permanece como `INVALID`. Para los índices únicos, las operaciones DML están sujetas a restricciones de unicidad hasta que descarte el índice. Le recomendamos que elimine los índices no válidos y los vuelva a crear.

Creación de un índice: ejemplo

En el siguiente ejemplo se muestra cómo crear un esquema, una tabla y, a continuación, un índice.

1. Cree una tabla denominada `test.departments`.

```
CREATE SCHEMA test;

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

2. Inserte una fila en la tabla.

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

3. Cree un índice asíncrono.

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

   El comando `CREATE INDEX` devuelve un ID de trabajo, como se muestra a continuación.

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

   Con `job_id` se indica que Aurora DSQL ha enviado un nuevo trabajo para crear el índice. Puede utilizar el procedimiento `sys.wait_for_job(job_id)` *`'your_index_creation_job_id'`* para bloquear otros trabajos en la sesión hasta que el trabajo finalice o se agote el tiempo de espera.

Consulta del estado de la creación del índice: ejemplo

Consulte la vista del sistema `sys.jobs` para comprobar el estado de creación del índice, como se muestra en el siguiente ejemplo.

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

Aurora DSQL devuelve una respuesta similar a la siguiente.

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

La columna de estado puede ser uno de los siguientes valores.

submitted	**processing**	**failed**	**completed**
La tarea se ha enviado, pero Aurora DSQL aún no ha empezado a procesarla.	Aurora DSQL está procesando la tarea.	La tarea ha fallado. Consulte la columna de detalles para obtener más información. Si Aurora DSQL no ha podido crear el índice, no elimina automáticamente la definición del índice. Debe eliminar manualmente el índice con el comando `DROP INDEX`.	Aurora DSQL

También puede consultar el estado del índice a través de las tablas `pg_index` y `pg_class` del catálogo. En concreto, los atributos `indisvalid` y `indisimmediate` pueden indicarle en qué estado se encuentra el índice. Mientras Aurora DSQL crea el índice, este tiene un estado inicial de

INVALID. La marca indisvalid del índice devuelve FALSE o f, lo que indica que el índice no es válido. Si la marca devuelve TRUE o t, el índice está listo.

```
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)
```

Errores en la creación de índices únicos

Si el trabajo de creación de índices únicos asíncrona muestra un estado erróneo con el detalle Found duplicate key while validating index for UCVs, esto indica que no se pudo crear un índice único debido a infracciones de restricciones de exclusividad.

Resolución de errores en la creación de índices únicos

1. Elimine las filas de la tabla principal que tengan entradas duplicadas para las claves especificadas en su índice secundario único.
2. Elimine el índice erróneo.
3. Emita un nuevo comando de creación de índice.

Detección de infracciones de unicidad en las tablas principales

La siguiente consulta SQL le ayuda a identificar los valores duplicados en una columna específica de la tabla. Esto resulta especialmente útil cuando se necesita imponer la unicidad en una columna que actualmente no está configurada como clave principal o que no tiene una restricción única, como las direcciones de correo electrónico en una tabla de usuarios.

Los ejemplos siguientes muestran cómo crear una tabla de usuarios de ejemplo, rellenarla con datos de ejemplo que contengan duplicados conocidos y, a continuación, ejecutar la consulta de detección.

Definición del esquema de la tabla

```
-- 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
);
```

Inserción de datos de ejemplo que incluyan conjuntos de direcciones de correo electrónico duplicadas

```
-- 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');
```

Ejecución de una consulta de detección de duplicados

```
-- 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;
```

Visualización de todos los registros con direcciones de correo electrónico duplicadas

```
 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)
```

Si intentáramos la instrucción de creación del índice ahora, produciría un error:

```
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)
```

Tablas y comandos del sistema en Aurora DSQL

Consulte las siguientes secciones para conocer las tablas y los catálogos del sistema admitidos en Aurora DSQL.

Tablas del sistema

Aurora DSQL es compatible con PostgreSQL, por lo que muchas tablas y vistas de catálogo del sistema de PostgreSQL también existen en Aurora DSQL.

Tablas y vistas de catálogo de PostgreSQL importantes

En la siguiente tabla se describen las tablas y las vistas más comunes que podría utilizar en Aurora DSQL.

Nombre	Descripción
`pg_namespace`	Información sobre todos los esquemas
`pg_tables`	Información sobre todas las tablas
`pg_attribute`	Información sobre todos los atributos
`pg_views`	Información sobre vistas (pre)definidas
`pg_class`	Describe todas las tablas, columnas, índices y objetos similares
`pg_stats`	Una vista sobre las estadísticas del planificador
`pg_user`	Información sobre usuarios
`pg_roles`	Información sobre usuarios y grupos
`pg_indexes`	Enumera todos los índices
`pg_constraint`	Enumera las restricciones de las tablas

Tablas de catálogo admitidas y no admitidas

En la siguiente tabla se indican las tablas admitidas y no admitidas en Aurora DSQL.

Nombre	Aplicable a Aurora DSQL
`pg_aggregate`	No

Nombre	Aplicable a Aurora DSQL
pg_am	Sí
pg_amop	No
pg_amproc	No
pg_attrdef	Sí
pg_attribute	Sí
pg_authid	No (utilice pg_roles)
pg_auth_members	Sí
pg_cast	Sí
pg_class	Sí
pg_collation	Sí
pg_constraint	Sí
pg_conversion	No
pg_database	No
pg_db_role_setting	Sí
pg_default_acl	Sí
pg_depend	Sí
pg_description	Sí
pg_enum	No
pg_event_trigger	No
pg_extension	No

Nombre	Aplicable a Aurora DSQL
pg_foreign_data_wrapper	No
pg_foreign_server	No
pg_foreign_table	No
pg_index	Sí
pg_inherits	Sí
pg_init_privs	No
pg_language	No
pg_largeobject	No
pg_largeobject_metadata	Sí
pg_namespace	Sí
pg_opclass	No
pg_operator	Sí
pg_opfamily	No
pg_parameter_acl	Sí
pg_partitioned_table	No
pg_policy	No
pg_proc	No
pg_publication	No
pg_publication_namespace	No
pg_publication_rel	No

Nombre	Aplicable a Aurora DSQL
pg_range	Sí
pg_replication_origin	No
pg_rewrite	No
pg_seclabel	No
pg_sequence	No
pg_shdepend	Sí
pg_shdescription	Sí
pg_shseclabel	No
pg_statistic	Sí
pg_statistic_ext	No
pg_statistic_ext_data	No
pg_subscription	No
pg_subscription_rel	No
pg_tablespace	No
pg_transform	No
pg_trigger	No
pg_ts_config	Sí
pg_ts_config_map	Sí
pg_ts_dict	Sí
pg_ts_parser	Sí

Nombre	Aplicable a Aurora DSQL
pg_ts_template	Sí
pg_type	Sí
pg_user_mapping	No

Vistas del sistema admitidas y no admitidas

En la siguiente tabla se indican las vistas admitidas y no admitidas en Aurora DSQL.

Nombre	Aplicable a Aurora DSQL
pg_available_extensions	No
pg_available_extension_versions	No
pg_backend_memory_contexts	Sí
pg_config	No
pg_cursors	No
pg_file_settings	No
pg_group	Sí
pg_hba_file_rules	No
pg_ident_file_mappings	No
pg_indexes	Sí
pg_locks	No
pg_matviews	No
pg_policies	No

Nombre	Aplicable a Aurora DSQL
pg_prepared_statements	No
pg_prepared_xacts	No
pg_publication_tables	No
pg_replication_origin_status	No
pg_replication_slots	No
pg_roles	Sí
pg_rules	No
pg_seclabels	No
pg_sequences	No
pg_settings	Sí
pg_shadow	Sí
pg_shmem_allocations	Sí
pg_stats	Sí
pg_stats_ext	No
pg_stats_ext_exprs	No
pg_tables	Sí
pg_timezone_abbrevs	Sí
pg_timezone_names	Sí
pg_user	Sí
pg_user_mappings	No

Nombre	Aplicable a Aurora DSQL
pg_views	Sí
pg_stat_activity	No
pg_stat_replication	No
pg_stat_replication_slots	No
pg_stat_wal_receiver	No
pg_stat_recovery_prefetch	No
pg_stat_subscription	No
pg_stat_subscription_stats	No
pg_stat_ssl	Sí
pg_stat_gssapi	No
pg_stat_archiver	No
pg_stat_io	No
pg_stat_bgwriter	No
pg_stat_wal	No
pg_stat_database	No
pg_stat_database_conflicts	No
pg_stat_all_tables	No
pg_stat_all_indexes	No
pg_statio_all_tables	No
pg_statio_all_indexes	No

Nombre	Aplicable a Aurora DSQL
pg_statio_all_sequences	No
pg_stat_slru	No
pg_statio_user_tables	No
pg_statio_user_sequences	No
pg_stat_user_functions	No
pg_stat_user_indexes	No
pg_stat_progress_analyze	No
pg_stat_progress_basebackup	No
pg_stat_progress_cluster	No
pg_stat_progress_create_index	No
pg_stat_progress_vacuum	No
pg_stat_sys_indexes	No
pg_stat_sys_tables	No
pg_stat_xact_all_tables	No
pg_stat_xact_sys_tables	No
pg_stat_xact_user_functions	No
pg_stat_xact_user_tables	No
pg_statio_sys_indexes	No
pg_statio_sys_sequences	No
pg_statio_sys_tables	No

Nombre	Aplicable a Aurora DSQL
pg_statio_user_indexes	No

Vistas sys.jobs y sys.iam_pg_role_mappings

Aurora DSQL admite las siguientes vistas del sistema:

sys.jobs

sys.jobs proporciona información sobre el estado de los trabajos asíncronos. Por ejemplo, después de crear un índice asíncrono, Aurora DSQL devuelve un job_uuid. Puede utilizar este job_uuid con sys.jobs para consultar el estado del trabajo.

```
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

La vista sys.iam_pg_role_mappings proporciona información sobre los permisos concedidos a los usuarios de IAM. Por ejemplo, si DQSLDBConnect es un rol de IAM que da acceso a Aurora DSQL a usuarios no administradores y a un usuario llamado testuser se le concede el rol DQSLDBConnect y los permisos correspondientes, puede consultar la vista sys.iam_pg_role_mappings para ver qué permisos se han concedido a qué usuarios.

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

Tabla pg_class

La tabla pg_class almacena metadatos sobre los objetos de la base de datos. Para obtener el recuento aproximado de cuántas filas hay en una tabla, ejecute el siguiente comando.

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

El comando devuelve un resultado similar al siguiente.

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

El comando **ANALYZE**

El comando ANALYZE recopila estadísticas sobre el contenido de las tablas de la base de datos y almacena los resultados en la vista del sistema `pg_stats`. Posteriormente, el planificador de consultas utiliza estas estadísticas para ayudar a determinar los planes de ejecución más eficaces para las consultas.

En Aurora DSQL, no puede ejecutar el comando ANALYZE en una transacción explícita. ANALYZE no está sujeto al límite de tiempo de espera de la transacción de la base de datos.

Para reducir la necesidad de intervención manual y mantener las estadísticas actualizadas de manera coherente, Aurora DSQL ejecuta automáticamente ANALYZE como un proceso en segundo plano. Este trabajo en segundo plano se activa automáticamente en función de la tasa de cambio observada en la tabla. Está vinculado al número de filas (tuplas) que se han insertado, actualizado o eliminado desde el último análisis.

ANALYZE se ejecuta de forma asíncrona en segundo plano y su actividad se puede supervisar en la vista del sistema sys.jobs con la siguiente consulta:

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

Consideraciones clave

> **Note**
>
> Los trabajos ANALYZE se facturan como otros trabajos asíncronos en Aurora DSQL. Cuando modifica una tabla, esto puede desencadenar indirectamente un trabajo automático de recopilación de estadísticas en segundo plano, lo que puede provocar cargos de medición debido a la actividad asociada al nivel del sistema.

Los trabajos ANALYZE en segundo plano, que se activan automáticamente, recopilan los mismos tipos de estadísticas que los ANALYZE manuales y los aplican de forma predeterminada a las tablas de los usuarios. Las tablas del sistema y del catálogo se excluyen de este proceso automatizado.

Administración de clústeres de Aurora DSQL

Aurora DSQL proporciona varias opciones de configuración para ayudarlo a establecer la infraestructura de base de datos adecuada a las necesidades. Para configurar la infraestructura de clústeres de Aurora DSQL, consulte las secciones siguientes.

Temas

- Configuración de clústeres de una sola región
- Configuración de clústeres multirregionales

Las características y las funcionalidades que se analizan en esta guía garantizan que el entorno de Aurora DSQL será más resiliente, receptivo y capaz de admitir las aplicaciones a medida que crecen y evolucionan.

Configuración de clústeres de una sola región

Creación de un clúster

Cree un clúster mediante el comando create-cluster.

> **Note**
>
> La creación de un clúster es una operación asíncrona. Llame a la API GetCluster hasta que el estado cambie a `ACTIVE`. Puede conectarse al clúster después de que se active.

Example Comando

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

> **Note**
>
> Para desactivar la protección contra eliminación durante la creación, incluya la marca `--no-deletion-protection-enabled`.

Example Respuesta

```
{
    "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"
    }
}
```

Descripción de un clúster

Obtenga información sobre un clúster mediante el comando get-cluster.

Example Comando

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

Example Respuesta

```
{
    "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"
    }
}
```

Actualización de un clúster

Actualice un clúster existente mediante el comando update-cluster.

> **Note**
>
> Las actualizaciones son operaciones asíncronas. Llame a la API GetCluster hasta que el estado cambie a `ACTIVE` para ver los cambios.

Example Comando

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

Example Respuesta

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

Eliminación de un clúster

Elimine un clúster existente mediante el comando delete-cluster.

> **Note**
>
> Solo puede eliminar clústeres que tengan desactivada la protección contra eliminación. De forma predeterminada, la protección contra eliminación está activada cuando crea nuevos clústeres.

Example Comando

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

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

Example Respuesta

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

Mostrar clústeres

Enumere los clústeres mediante el comando list-clusters.

Example Comando

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

Example Respuesta

```
{
    "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"
        }
    ]
}
```

Configuración de clústeres multirregionales

En este capítulo se explica cómo configurar y administrar clústeres que abarcan múltiples Regiones de AWS.

Conexión al clúster multirregional

Los clústeres emparejados multirregionales proporcionan dos puntos de conexión regionales, uno en cada Región de AWS de clúster interconectado. Ambos puntos de conexión presentan una única base de datos lógica que admite operaciones de lectura y escritura simultáneas con una coherencia de datos sólida. Además de los clústeres emparejados, un clúster de varias regiones también tiene una región testigo que almacena una ventana limitada de registros de transacciones cifrados, que se utiliza para mejorar la durabilidad y la disponibilidad de varias regiones. Las regiones testigo de varias regiones no tienen puntos de conexión.

Creación de clústeres multirregionales

Para crear clústeres de varias regiones, primero debe crear un clúster con una región testigo. A continuación, empareje este clúster con un segundo clúster que comparte la misma región testigo que el primer clúster. En el siguiente ejemplo se muestra cómo crear clústeres en Este de EE. UU. (Norte de Virginia) y Este de EE. UU. (Ohio) con Oeste de EE. UU. (Oregón) como región testigo.

Paso 1: creación del clúster 1 en Este de EE. UU. (Norte de Virginia)

Para crear un clúster en la Región de AWS Este de EE. UU. (Norte de Virginia) con propiedades multirregionales, utilice el siguiente comando.

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

Example Respuesta:

```
{
    "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**
>
> Cuando la operación de la API se realiza correctamente, el clúster cambia al estado `PENDING_SETUP`. La creación del clúster permanece en `PENDING_SETUP` hasta que actualice el clúster con el ARN del clúster emparejado.

Paso 2: creación del clúster 2 en Este de EE. UU. (Ohio)

Para crear un clúster en la Región de AWS Este de EE. UU. (Ohio) con propiedades multirregionales, utilice el siguiente comando.

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

Example Respuesta:

```
{
    "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"
        ]
    }
}
```

Cuando la operación de la API se realiza correctamente, el clúster realiza la transición al estado `PENDING_SETUP`. La creación del clúster permanece en el estado `PENDING_SETUP` hasta que lo actualice con el ARN de otro clúster para el emparejamiento.

Paso 3: emparejamiento del clúster en Este de EE. UU. (Norte de Virginia) con Este de EE. UU. (Ohio)

Para emparejar el clúster de Este de EE. UU. (Norte de Virginia) con el clúster de Este de EE. UU. (Ohio), utilice el comando `update-cluster`. Especifique el nombre del clúster del Este de EE. UU. (Norte de Virginia) y una cadena JSON con el ARN del clúster de Este de EE. UU. (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 Respuesta

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

Paso 4: emparejamiento del clúster de Este de EE. UU. (Ohio) con el clúster de Este de EE. UU. (Norte de Virginia)

Para emparejar el clúster de Este de EE. UU. (Ohio) con el clúster de Este de EE. UU. (Norte de Virginia), utilice el comando `update-cluster`. Especifique el nombre del clúster Este de EE. UU. (Ohio) y una cadena JSON con el ARN del clúster Este de EE. UU. (Norte de 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 Respuesta

```
{
```

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

> **Note**
>
> Tras un emparejamiento correcto, ambos clústeres pasan del estado “PENDING_SETUP” al estado “CREATING” y, finalmente, al estado “ACTIVE” cuando están listos para utilizarse.

Visualización de propiedades de clústeres multirregionales

Al describir un clúster, puede ver las propiedades multirregionales de los clústeres en diferentes Regiones de AWS.

Example

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

Example Respuesta

```
{
    "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"
       ]
```

```
    }
}
```

Emparejamiento de clústeres durante la creación

Puede reducir el número de pasos si incluye información de emparejamiento durante la creación del clúster. Después de crear el primer clúster en Este de EE. UU. (Norte de Virginia) (paso 1), puede crear el segundo clúster en Este de EE. UU. (Ohio) mientras inicia simultáneamente el proceso de emparejamiento mediante la inclusión del ARN del primer clúster.

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"]}'
```

Esto combina los pasos 2 y 4, pero aún deberá completar el paso 3 (actualizar el primer clúster con el ARN del segundo clúster) para establecer la relación de emparejamiento. Una vez completados todos los pasos, ambos clústeres pasarán por los mismos estados que en el proceso estándar: de PENDING_SETUP a CREATING y, finalmente, a ACTIVE cuando estén listos para utilizarse.

Eliminación de clústeres multirregionales

Para eliminar un clúster multirregional, debe completar dos pasos.

1. Desactive la protección contra eliminación de cada clúster.
2. Elimine cada clúster emparejado por separado en la Región de AWS correspondiente

Actualización y eliminación del clúster en Este de EE. UU. (Norte de Virginia)

1. Desactive la protección contra eliminación mediante el comando `update-cluster`.

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

2. Elimine el clúster con el comando `delete-cluster`.

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

El comando devuelve el siguiente resultado.

```
{
    "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**
>
> El clúster pasa al estado `PENDING_DELETE`. La eliminación no estará completa hasta que elimine el clúster emparejado en Este de EE. UU. (Ohio).

Actualización y eliminación del clúster en Este de EE. UU. (Ohio)

1. Desactive la protección contra eliminación mediante el comando `update-cluster`.

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

2. Elimine el clúster con el comando `delete-cluster`.

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

El comando devuelve la siguiente respuesta:

```
{
    "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**
>
> El clúster pasa al estado `PENDING_DELETE`. Después de unos segundos, el sistema realiza de forma automática la transición de ambos clústeres emparejados al estado `DELETING` después de la validación.

Configuración de clústeres de varias regiones con AWS CloudFormation

Puede usar el mismo recurso AWS CloudFormation `AWS::DSQL::Cluster` para implementar y administrar clústeres de Aurora DSQL de una sola región y de varias regiones.

Consulte la referencia del tipo de recurso de Amazon Aurora DSQL para obtener más información sobre cómo crear, modificar y administrar clústeres mediante el recurso `AWS::DSQL::Cluster`.

Creación de la configuración inicial del clúster

En primer lugar, cree una plantilla de AWS CloudFormation para definir el clúster de varias regiones:

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

Cree pilas en ambas regiones mediante los siguientes comandos de la CLI de AWS:

```
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
```

Búsqueda de identificadores de clúster

Recupere los ID de recursos físicos para los clústeres:

```
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"
]
```

Actualización de la configuración del clúster

Actualice la plantilla de AWS CloudFormation para incluir los ARN del clúster:

```
---
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
```

Aplique la configuración actualizada a ambas regiones:

```
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
```

Programación con Aurora DSQL

Aurora DSQL le proporciona las siguientes herramientas para administrar los recursos de Aurora DSQL mediante programación.

AWS Command Line Interface (AWS CLI)

Puede crear y administrar los recursos mediante la AWS CLI en un intérprete de comandos de la línea de comandos. La AWS CLI ofrece acceso directo a las API para Servicios de AWS, por ejemplo, Aurora DSQL. Para ver la sintaxis y ejemplos de los comandos para Aurora DSQL, consulte dsql en la Referencia de comandos de AWS CLI.

Kits de desarrollo de software (SDK) de AWS

AWS proporciona SDK para muchas tecnologías y lenguajes de programación conocidos. Le facilitan las llamadas a los Servicios de AWS desde las aplicaciones en ese lenguaje o tecnología. Para obtener más información sobre estos SDK, consulte Tools for developing and managing applications on AWS.

API de Aurora DSQL

Esta API es otra interfaz de programación para Aurora DSQL. Al utilizar esta API, debe formatear correctamente todas las solicitudes de HTTPS y añadir una firma digital válida a cada solicitud. Para obtener más información, consulte Referencia de la API.

AWS CloudFormation

AWS::DSQL::Cluster es un recurso de AWS CloudFormation que le permite crear y administrar clústeres de Aurora DSQL como parte de la infraestructura como código. AWS CloudFormation lo ayuda a definir todo el entorno de AWS en código, lo que facilita el aprovisionamiento, la actualización y la replicación de la infraestructura de forma coherente y fiable.

Cuando utiliza el recurso AWS::DSQL::Cluster en las plantillas de AWS CloudFormation, puede aprovisionar de forma declarativa clústeres de Aurora DSQL junto con otros recursos en la nube. Esto lo ayuda a garantizar que la infraestructura de datos se implementa y administra junto con el resto de la pila de aplicaciones.

SDK, controladores y código de muestra de Amazon Aurora DSQL

Los kits de desarrollo de software (SDK) de AWS se encuentran disponibles en muchos lenguajes de programación populares. Cada SDK proporciona una API, ejemplos de código y documentación que facilitan a los desarrolladores la creación de aplicaciones en su lenguaje preferido.

Adaptadores y dialectos

En la siguiente tabla se muestran los adaptadores ORM y los dialectos de base de datos disponibles para Aurora DSQL.

Lenguaje de programación	Marcos	Enlace al repositorio
Java	Hibernate	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/

Ejemplos de código

Administración de clústeres mediante AWS SDK

En la siguiente tabla se muestran ejemplos de código de administración de clústeres para diferentes lenguajes de programación que mediante AWS SDK.

Lenguaje de programación	Enlace al repositorio de ejemplos
C++	https://github.com/aws-samples/aurora-dsql-samples/tree/main/cpp/cluster_management
C# (.NET)	https://github.com/aws-samples/aurora-dsql-samples/tree/main/dotnet/cluster_management

Lenguaje de programación	Enlace al repositorio de ejemplos
Go	https://github.com/aws-samples/aurora-dsql-samples/tree/main/go/cluster_management
Java	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/cluster_management
JavaScript	https://github.com/aws-samples/aurora-dsql-samples/tree/main/javascript/cluster_management
Python	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/cluster_management
Ruby	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/cluster_management
Rust	https://github.com/aws-samples/aurora-dsql-samples/tree/main/rust/cluster_management

Controladores y ejemplos de asignación relacional de objeto (ORM)

En la siguiente tabla se muestran ejemplos de código de controladores de bases de datos y marcos ORM para diferentes lenguajes de programación.

Lenguaje de programación	Controlador o marco	Enlace al repositorio de ejemplos
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

Lenguaje de programación	Controlador o marco	Enlace al repositorio de ejemplos
Go	pgx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/go/pgx
Java	Hibernate	https://github.com/awslabs/aurora-dsql-hibernate/tree/main/examples/pet-clinic-app
Java	pgJDBC	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/pgjdbc
JavaScript	node-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-clinic-app

Lenguaje de programación	Controlador o marco	Enlace al repositorio de ejemplos
Ruby	Rails	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/rails
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 con la AWS CLI

Consulte las secciones siguientes para obtener información sobre cómo administrar los clústeres con la AWS CLI.

CreateCluster

Para crear un clúster, utilice el comando `create-cluster`.

> **Note**
>
> La creación del clúster se produce de forma asíncrona. Llame a la API `GetCluster` hasta que el estado sea `ACTIVE`. Puede conectarse a un clúster una vez que se convierta en `ACTIVE`.

Comando de ejemplo

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

> **Note**
>
> Si desea desactivar la protección contra eliminación en el momento de la creación, incluya la marca `--no-deletion-protection-enabled`.

Respuesta de ejemplo

```
{
    "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

Para describir un clúster, utilice el comando `get-cluster`.

Comando de ejemplo

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

Respuesta de ejemplo

```
{
    "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

Para actualizar un clúster existente, utilice el comando `update-cluster`.

> **Note**
>
> Las actualizaciones se producen de forma asíncrona. Llame a la API `GetCluster` hasta que el estado sea `ACTIVE` y observará los cambios.

Comando de ejemplo

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

Respuesta de ejemplo

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

DeleteCluster

Para eliminar un clúster existente, utilice el comando `delete-cluster`.

> **Note**
>
> Solo puede eliminar un clúster que tenga desactivada la protección contra eliminación. La protección contra eliminación está activada de forma predeterminada al crear nuevos clústeres.

Comando de ejemplo

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

Respuesta de ejemplo

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

ListClusters

Para obtener la lista de los clústeres, utilice el comando `list-clusters`.

Comando de ejemplo

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

Respuesta de ejemplo

```
{
    "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 en clústeres multirregionales

Para obtener información sobre un clúster multirregional, utilice el comando `get-cluster`. En el caso de los clústeres multirregionales, la respuesta incluirá los ARN de los clústeres vinculados.

Comando de ejemplo

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

Respuesta de ejemplo

```
{
    "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"
    }
}
```

Creación, lectura, actualización y eliminación de clústeres de Aurora DSQL

Se proporcionan ejemplos de creación, lectura, actualización y eliminación (CRUD) tanto para implementaciones en una sola región como multirregionales. Se incluye una sección `cluster_management` dedicada para cada lenguaje de programación que demuestra estas tareas de administración clave.

Las implementaciones de una sola región son ideales para aplicaciones que dan servicio a usuarios de una zona geográfica específica, ya que ofrecen una administración simplificada y una latencia menor. Las implementaciones multirregionales lo ayudan a conseguir una mayor disponibilidad y capacidad de recuperación ante desastres mediante la distribución de la base de datos en varias Regiones de AWS.

Elija el tipo de implementación que se ajuste a los requisitos de disponibilidad, rendimiento y distribución geográfica de la aplicación.

Temas

Creación de un clúster

Consulte la siguiente información para aprender a crear clústeres de una región y multirregionales en Aurora DSQL.

Python

Para crear un clúster en una sola Región de AWS, utilice el siguiente ejemplo.

```
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()
```

Para crear un clúster multirregional, utilice el siguiente ejemplo. Crear un clúster multirregional puede llevar algún tiempo.

```
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++

El siguiente ejemplo le permite crear un clúster en una sola Región de 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;
}
```

Para crear un clúster multirregional, utilice el siguiente ejemplo. Crear un clúster multirregional puede llevar algún tiempo.

```
#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

Para crear un clúster en una sola Región de AWS, utilice el siguiente ejemplo.

```
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();
```

Para crear un clúster multirregional, utilice el siguiente ejemplo. Crear un clúster multirregional puede llevar algún tiempo.

```
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

Utilice el siguiente ejemplo para crear un clúster en una sola Región de 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);
        }
    }
}
```

Para crear un clúster multirregional, utilice el siguiente ejemplo. Crear un clúster multirregional puede llevar algún tiempo.

```
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

Utilice el siguiente ejemplo para crear un clúster en una sola Región de 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(())
}
```

Para crear un clúster multirregional, utilice el siguiente ejemplo. Crear un clúster multirregional puede llevar algún tiempo.

```
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

Utilice el siguiente ejemplo para crear un clúster en una sola Región de 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__
```

Para crear un clúster multirregional, utilice el siguiente ejemplo. Crear un clúster multirregional puede llevar algún tiempo.

```
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

Utilice el siguiente ejemplo para crear un clúster en una sola Región de 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);
```

```
        }
    }
}
```

Para crear un clúster multirregional, utilice el siguiente ejemplo. Crear un clúster multirregional puede llevar algún tiempo.

```
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

Utilice el siguiente ejemplo para crear un clúster en una sola Región de 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)
 }
}
```

Para crear un clúster multirregional, utilice el siguiente ejemplo. Crear un clúster multirregional puede llevar algún tiempo.

```
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)
 }

}
```

Obtención de un clúster

Consulte la siguiente información para aprender a devolver información de un clúster en Aurora DSQL.

Python

Para obtener información sobre un clúster de una región o multirregional, utilice el siguiente ejemplo.

```
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++

Utilice el siguiente ejemplo para obtener información sobre un clúster de una región o multirregional.

```
#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

Para obtener información sobre un clúster de una región o multirregional, utilice el siguiente ejemplo.

```
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

El siguiente ejemplo le permite obtener información sobre un clúster de una región o multirregional.

```
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

El siguiente ejemplo le permite obtener información sobre un clúster de una región o multirregional.

```
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

El siguiente ejemplo le permite obtener información sobre un clúster de una región o multirregional.

```
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

El siguiente ejemplo le permite obtener información sobre un clúster de una región o multirregional.

```
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

El siguiente ejemplo le permite obtener información sobre un clúster de una región o multirregional.

```
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)
 }
}
```

Actualice un clúster de

Consulte la siguiente información para aprender a actualizar un clúster en Aurora DSQL. La actualización de un clúster puede tardar uno o dos minutos. Le recomendamos que espere algún tiempo y, a continuación, ejecute get clúster para obtener el estado del clúster.

Python

Para actualizar un clúster de una región o multirregional, utilice el siguiente ejemplo.

```
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++

Utilice el siguiente ejemplo para actualizar un clúster de una región o multirregional.

```
#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

Para actualizar un clúster de una región o multirregional, utilice el siguiente ejemplo.

```
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

Utilice el siguiente ejemplo para actualizar un clúster de una región o multirregional.

```
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

Utilice el siguiente ejemplo para actualizar un clúster de una región o multirregional.

```
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

Utilice el siguiente ejemplo para actualizar un clúster de una región o multirregional.

```
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

Utilice el siguiente ejemplo para actualizar un clúster de una región o multirregional.

```
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

Utilice el siguiente ejemplo para actualizar un clúster de una región o multirregional.

```
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)
 }
```

```
}
```

Eliminar un clúster

Consulte la siguiente información para aprender a eliminar un clúster en Aurora DSQL.

Python

Para eliminar un clúster en una sola Región de AWS, utilice el siguiente ejemplo.

```
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()
```

Para eliminar un clúster multirregional, utilice el siguiente ejemplo.

```
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++

Para eliminar un clúster en una sola Región de AWS, utilice el siguiente ejemplo.

```
#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;
}
```

Para eliminar un clúster multirregional, utilice el siguiente ejemplo. Eliminar un clúster multirregional puede llevar algún tiempo.

```
#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

Para eliminar un clúster en una sola Región de AWS, utilice el siguiente ejemplo.

```
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();
```

Para eliminar un clúster multirregional, utilice el siguiente ejemplo. Eliminar un clúster multirregional puede llevar algún tiempo.

```
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

Para eliminar un clúster en una sola Región de AWS, utilice el siguiente ejemplo.

```
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);
        }
    }
}
```

Para eliminar un clúster multirregional, utilice el siguiente ejemplo. Eliminar un clúster multirregional puede llevar algún tiempo.

```
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

Para eliminar un clúster en una sola Región de AWS, utilice el siguiente ejemplo.

```
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(())
}
```

Para eliminar un clúster multirregional, utilice el siguiente ejemplo. Eliminar un clúster multirregional puede llevar algún tiempo.

```
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

Para eliminar un clúster en una sola Región de AWS, utilice el siguiente ejemplo.

```
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__
```

Para eliminar un clúster multirregional, utilice el siguiente ejemplo. Eliminar un clúster multirregional puede llevar algún tiempo.

```
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

Para eliminar un clúster en una sola Región de AWS, utilice el siguiente ejemplo.

```
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);
        }
    }
}
```

Para eliminar un clúster multirregional, utilice el siguiente ejemplo. Eliminar un clúster multirregional puede llevar algún tiempo.

```
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

Para eliminar un clúster en una sola Región de AWS, utilice el siguiente ejemplo.

```
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)
 }

}
```

Para eliminar un clúster multirregional, utilice el siguiente ejemplo. Eliminar un clúster multirregional puede llevar algún tiempo.

```
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)
 }
}
```

Tutoriales

Los siguientes tutoriales y códigos de ejemplo en GitHub lo ayudan a realizar tareas comunes en Aurora DSQL.

- Uso de Benchbase con Aurora DSQL: una bifurcación de la utilidad de análisis comparativo de código abierto Benchbase que está verificada para funcionar con Aurora DSQL.
- Cargador de Aurora DSQL: este script en Python de código abierto le facilita la carga de datos en Aurora DSQL para los casos de uso, como rellenar tablas para pruebas o transferir datos a Aurora DSQL.
- Muestras de Aurora DSQL: el repositorio `aws-samples/aurora-dsql-samples` en GitHub contiene ejemplos de código sobre cómo conectar y utilizar Aurora DSQL en varios lenguajes de programación mediante los AWS SDK, asignadores de objetos relacionales (ORM) y marcos web. En los ejemplos se demuestra cómo realizar tareas comunes, por ejemplo, instalar clientes, gestionar la autenticación y realizar operaciones CRUD.

Uso de AWS Lambda con Amazon Aurora DSQL

En el siguiente tutorial se describe cómo utilizar Lambda con Aurora DSQL

Requisitos previos

- Autorización para crear funciones de Lambda. Para obtener más información, consulte Introducción a Lambda.
- Autorización para crear o modificar la política de IAM creada por Lambda. Necesita los permisos `iam:CreatePolicy` y `iam:AttachRolePolicy`. Para obtener más información, consulte Acciones, recursos y claves de condición para IAM.
- Debe tener instalado npm v8.5.3 o superior.

- Debe tener instalado zip v3.0 o superior.

Cree una nueva función en AWS Lambda.

1. Inicie sesión en la AWS Management Console y abra la consola AWS Lambda en https://console.aws.amazon.com/lambda/.
2. Seleccione Creación de función.
3. Proporcione un nombre, como `dsql-sample`.
4. No edite la configuración predeterminada para asegurarse de que Lambda crea un nuevo rol con los permisos de Lambda básicos.
5. Seleccione Creación de función.

Autorización del rol de ejecución de Lambda para conectarse al clúster

1. En la función de Lambda, elija Configuración > Permisos.
2. Elija el nombre del rol para abrir el rol de ejecución en la consola de IAM.
3. Elija Agregar permisos > Crear política insertada y utilice el editor JSON.
4. En Acción pegue la siguiente acción para autorizar a la identidad de IAM para que se conecte mediante el rol de base de datos administrador.

```
"Action": ["dsql:DbConnectAdmin"],
```

> **Note**
>
> Estamos utilizando un rol admin para minimizar los pasos de requisitos previos para empezar. No debe utilizar un rol de base de datos administración para las aplicaciones de producción. Consulte Uso de roles de base de datos y autenticación de IAM para obtener información sobre cómo crear roles de base de datos personalizados con autorización que tengan el menor número de permisos para la base de datos.

5. En Recurso, agregue el nombre de recurso de Amazon (ARN) del clúster. También puede utilizar un carácter comodín.

```
"Resource": ["*"]
```

6. Elija Siguiente.

7. Introduzca un Nombre para la política, como `dsql-sample-dbconnect`.
8. Seleccione Crear política.

Cree un paquete para cargarlo en Lambda.

1. Cree una carpeta denominada `myfunction`.
2. En la carpeta, cree un nuevo archivo llamado `package.json` con el siguiente contenido.

```
{
  "dependencies": {
    "@aws-sdk/dsql-signer": "^3.705.0",
    "assert": "2.1.0",
    "pg": "^8.13.1"
  }
}
```

3. En la carpeta, cree un archivo denominado `index.mjs` en el directorio con el siguiente contenido.

```
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. Utilice los siguientes comandos para crear un paquete.

```
npm install
zip -r pkg.zip .
```

Carga del paquete de código y prueba de la función de Lambda

1. En la pestaña Código de la función de Lambda, elija Cargar desde > Archivo .zip
2. Cargue el archivo `pkg.zip` que ha creado. Para obtener más información, consulte Implementación de funciones de Lambda de Node.js con archivos .zip.
3. En la pestaña Prueba de la función de Lambda, pegue la siguiente carga útil JSON y modifíquela para utilizar el ID de clúster.
4. En la pestaña Prueba de la función de Lambda, utilice el siguiente Evento JSON modificado para especificar el punto de conexión del clúster.

```
{"endpoint": "replace_with_your_cluster_endpoint"}
```

5. Escriba un nombre de evento, como `dsql-sample-test`. Seleccione Save.
6. Seleccione Test (Probar).
7. Elija Detalles para ampliar la respuesta de ejecución y registrar la salida.
8. Si se ha realizado correctamente, la respuesta de ejecución de la función de Lambda debe devolver un código de estado 200.

```
{"statusCode": 200, "endpoint": "your_cluster_endpoint"}
```

Si la base de datos devuelve un error o si no se realiza la conexión con la base de datos, la respuesta de ejecución de la función de Lambda devuelve un código de estado 500.

```
{"statusCode": 500,"endpoint": "your_cluster_endpoint"}
```

Copia de seguridad y restauración para Amazon Aurora DSQL

Amazon Aurora DSQL le ayuda a cumplir los requisitos de cumplimiento normativo y de continuidad empresarial a través de la integración con AWS Backup, un servicio de protección de datos completamente administrado que facilita la centralización y automatización de las copias de seguridad en todos los servicios de AWS, en la nube y en las instalaciones. El servicio optimiza la creación, la administración y la restauración de copias de seguridad para los clústeres de Aurora DSQL de una sola región y de varias regiones.

Estas son algunas de las principales características:

- Administración centralizada de copias de seguridad mediante la AWS Management Console, el SDK o la AWS CLI
- Copias de seguridad de clúster completas
- Programación de copias de seguridad automatizada y políticas de retención
- Capacidades entre regiones y entre cuentas
- Configuración de WORM (escritura única y lectura múltiple) para todas las copias de seguridad que almacene

Para obtener más información sobre las características del bloqueo de almacenes de AWS Backup y una lista exhaustiva de las características de AWS Backup disponibles para Aurora DSQL, consulte Beneficios del bloqueo de almacenes y Disponibilidad de características de AWS Backup en la Guía para desarrolladores de AWS Backup.

Introducción a AWS Backup

AWS Backup crea copias completas de los clústeres de Aurora DSQL. Puede empezar a utilizar AWS Backup para Aurora DSQL siguiendo los pasos en Introducción a AWS Backup:

1. Cree copias de seguridad bajo demanda para una protección inmediata.
2. Establezca planes de copia de seguridad para copias de seguridad automatizadas y programadas.
3. Configure los periodos de retención y la copia entre regiones.
4. Configure la supervisión y las notificaciones de las actividades de copia de seguridad.

Restauración de las copias de seguridad

Al restaurar los clústeres de Aurora DSQL, AWS Backup siempre crea nuevos clústeres para conservar los datos de origen. Para restaurar un clúster de Aurora DSQL de una sola región, utilice https://console.aws.amazon.com/backup o la CLI para seleccionar el punto de recuperación (copia de seguridad) que desea restaurar. Configure los ajustes del nuevo clúster que se creará a partir de la copia de seguridad.

La restauración de un clúster de varias regiones de Aurora DSQL solo se admite a través de la AWS CLI. Para restaurar un clúster de varias regiones de Aurora DSQL, debe usar tanto AWS Backup como la CLI de Aurora DSQL.

Para restaurar un clúster de varias regiones de Aurora DSQL.

1. Seleccione el punto de recuperación del clúster de varias regiones.
2. Copie el punto de recuperación en otra Región de AWS que admita clústeres de varias regiones.

   > **Note**
   >
   > Las regiones que no admiten clústeres de varias regiones provocarán un error en la operación de restauración.

3. Inicie un trabajo de restauración para cada clúster mediante la CLI de AWS Backup.
4. Utilice la documentación de Configuración de clústeres multirregionales para emparejar los clústeres de Aurora DSQL recién creados.

Para obtener instrucciones detalladas sobre estos pasos, consulte la documentación de restauración de Amazon Aurora DSQL.

Para restaurar en un clúster de Aurora DSQL de varias regiones, puede utilizar una copia de seguridad realizada en una sola Región de AWS. Sin embargo, antes de iniciar el proceso de restauración, primero debe copiar la copia de seguridad en otra Región de AWS que admita clústeres de varias regiones. Este paso garantiza que la operación de restauración se complete correctamente. Recomendamos crear copias de seguridad en las Regiones de AWS clave como Este de EE. UU. (Norte de Virginia), Este de EE. UU. (Ohio) u Oeste de EE. UU. (Oregón) para habilitar opciones sólidas de recuperación ante desastres y cumplir con los requisitos de conformidad.

Supervisión y conformidad

AWS Backup proporciona una visibilidad completa de las operaciones de copia de seguridad y restauración con los siguientes recursos.

- Un panel centralizado para el seguimiento de los trabajos de copia de seguridad y restauración
- Integración con CloudWatch y CloudTrail.
- AWS Backup Audit Manager para la elaboración de informes y auditorías de conformidad.

Consulte Registro de operaciones de Aurora DSQL mediante AWS CloudTrail para obtener más información sobre el registro de las acciones realizadas por un usuario, un rol o un Servicio de AWS durante el uso de Aurora DSQL.

Recursos adicionales

Para obtener más información sobre las características de AWS Backup y su uso en conjunto con Aurora DSQL, consulte los siguientes recursos:

- Políticas administradas para AWS Backup
- Restauración de Amazon Aurora DSQL
- Servicios admitidos por Región de AWS
- Cifrado para copias de seguridad en AWS Backup

Al usar AWS Backup para Aurora DSQL, se implementa una estrategia de copia de seguridad sólida, compatible y automatizada que protege los recursos críticos de la base de datos y, al mismo tiempo, minimiza la sobrecarga administrativa. Si administra un solo clúster o una implementación compleja de varias regiones, AWS Backup proporciona las herramientas que necesita para garantizar que los datos permanezcan seguros y recuperables.

Supervisión y registro para Aurora DSQL

La supervisión y el registro son una parte importante del mantenimiento de la fiabilidad, la disponibilidad y el rendimiento de los recursos de Amazon Aurora DSQL. Debe supervisar y recopilar datos de registro de todas las partes de las soluciones de Aurora DSQL para poder depurar fácilmente un error de varios puntos.

- Amazon CloudWatch monitorea los recursos de AWS y las aplicaciones que ejecuta en AWS en tiempo real. Puede recopilar métricas y realizar un seguimiento de las métricas, crear paneles personalizados y definir alarmas que le advierten o que toman medidas cuando una métrica determinada alcanza el umbral que se especifique. Por ejemplo, puede hacer que CloudWatch haga un seguimiento del uso de la CPU u otras métricas de las instancias de Amazon EC2 y abrir nuevas instancias automáticamente cuando sea necesario. Para obtener más información, consulte la Guía del usuario de Amazon CloudWatch.
- AWS CloudTrail captura las llamadas a la API y otros eventos relacionados que realiza la Cuenta de AWS o que se realizan en nombre de esta. Además, entrega los archivos de registro a un bucket de Amazon S3 especificado. También pueden identificar qué usuarios y cuentas llamaron a AWS, la dirección IP de origen de las llamadas y el momento en que estas se realizaron. Para obtener más información, consulte la Guía del usuario de AWS CloudTrail.

Visualización del estado del clúster de Aurora DSQL

El estado del clúster de Aurora DSQL proporciona información fundamental sobre el estado y la conectividad del clúster. Puede ver el estado de los clústeres y las instancias del clúster mediante la AWS Management Console, la AWS CLI o la API de Aurora DSQL.

Estados y definiciones de los clústeres de Aurora DSQL

En la siguiente tabla se describe cada estado posible de un clúster de Aurora DSQL y lo que significa cada estado.

Estado	Descripción
Creando	Aurora DSQL está intentando crear o configurar recursos para el clúster. Los intentos de conexión producirán un error mientras el clúster esté en este estado.

Estado	Descripción
Activo	El clúster está en funcionamiento y listo para utilizarse.
Inactivo	Un clúster queda inactivo cuando permanece inactivo el tiempo suficiente para que Aurora DSQL recupere los recursos configurados para él. Cuando se conecta a un clúster inactivo, Aurora DSQL devuelve el clúster al estado Activo.
Inactivo	Un clúster se vuelve inactivo cuando no ha habido actividad en el clúster durante un periodo prolongado. Cuando intenta conectarse a un clúster inactivo, Aurora DSQL devuelve el clúster al estado Activo de forma automática.
Actualización	Un clúster pasa al estado Actualizando al realizar cambios en la configuración del clúster.
Eliminando	Un clúster pasa al estado Eliminando cuando se envía una solicitud para eliminarlo.
Eliminado	El clúster se ha eliminado correctamente.
Con error	Aurora DSQL no pudo crear el clúster porque encontró un error.
Configuración pendiente	Solo para clústeres de varias regiones. Un clúster de varias regiones pasa al estado Configuración pendiente al crear un clúster de varias regiones en la primera región con una región testigo. La creación del clúster se detiene hasta que se crea otro clúster en una región secundaria y se emparejan los dos clústeres.
Eliminación pendiente	Solo para clústeres de varias regiones. Un clúster de varias regiones pasa al estado Eliminación pendiente al eliminar un clúster de él. El clúster pasa al estado Eliminando una vez que se elimina el último clúster emparejado.

Visualización del estado del clúster de Aurora DSQL

Para ver el estado del clúster, utilice la AWS Management Console, la AWS CLI o la API de Aurora DSQL.

Consola

Siga estos pasos para ver el estado del clúster en la AWS Management Console:

Visualización del estado del clúster en la consola

1. Abra la consola de Aurora DSQL en https://console.aws.amazon.com/dsql.
2. Seleccione Clusters (Clústeres) en el panel de navegación.
3. Vea el estado de cada clúster en el panel de control.

AWS CLI

Use el siguiente comando de la AWS CLI para comprobar el estado de un solo clúster.

```
aws dsql get-cluster --identifier cluster-id --query status --output text
```

Ejecute el siguiente comando para mostrar el estado de todos los clústeres.

```
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
```

Este resultado de ejemplo muestra dos clústeres activos y un clúster en proceso de eliminación.

```
aaabbb2bkx555xa7p42qd5cdef    ACTIVE
abcde123efghi77t35abcdefgh    ACTIVE
12abc6lqasc5bbbbbbbbbbbbbb    DELETING
```

Supervisión de Aurora DSQL con Amazon CloudWatch

Supervise Aurora DSQL mediante CloudWatch, que recopila y procesa los datos sin procesar y los convierte en métricas legibles y casi en tiempo real. CloudWatch conserva estas estadísticas durante 15 meses, lo que lo ayuda a obtener una perspectiva mejor del rendimiento de la aplicación o servicio web. Establezca alarmas para vigilar umbrales específicos y enviar notificaciones o realizar acciones cuando se cumplan. Revise las siguientes métricas de uso y observabilidad disponibles para Aurora DSQL.

Para obtener más información, consulte la Guía del usuario de Amazon CloudWatch.

Observabilidad y rendimiento

En esta tabla se describen las métricas de observabilidad de Aurora DSQL. Incluye métricas para el seguimiento de las transacciones totales y de solo lectura a fin de proporcionar una caracterización general de la carga de trabajo. Se incluyen métricas procesables, como los tiempos de espera de las consultas y la tasa de conflictos de OCC, para ayudar a identificar los problemas de rendimiento y los conflictos de concurrencia. Las métricas relacionadas con la sesión, activas y totales, ofrecen información sobre la carga actual del sistema.

Nombre de métrica de CloudWatch	Métrica	Unidad	Descripción
ReadOnlyTransactions	Read-only transactions	none	The number of read-only transactions
TotalTransactions	Total transactions	none	The total number of transactions executed on the system, including read-only transactions.
QueryTimeouts	Query timeouts	none	The number of queries which have timed out due to hitting the maximum transaction time
OccConflicts	OCC conflicts	none	The number of transactions aborted due to key level OCC
CommitLatency	Commit Latency	milliseconds	Time spent by commit phase of query execution (P50)
BytesWritten	Bytes Written	bytes	Bytes written to storage

Nombre de métrica de CloudWatch	Métrica	Unidad	Descripción
BytesRead	Bytes Read	bytes	Bytes read from storage
ComputeTime	QP compute time	milliseconds	QP wall clock time
ClusterStorageSize	Cluster Storage Size	bytes	Cluster size

Métricas de uso

Aurora DSQL mide toda la actividad basada en solicitudes, como el procesamiento de consultas, las lecturas y las escrituras, mediante una única unidad de facturación normalizada llamada Unidad de procesamiento distribuido (DPU).

Nombre de métrica de CloudWatch	Métrica	Dimensión: ResourceId	Unidad	Descripción
WriteDPU	Write Units	<cluster-id>	DPU	Approximates the write active-use component of your Aurora DSQL cluster DPU usage.
MultiRegionWriteDPU	Multi-Region Write Units	<cluster-id>	DPU	Applicable for Multi-Region clusters: Approximates the multi-Region write active-use component of your Aurora DSQL cluster DPU usage.

Nombre de métrica de CloudWatch	Métrica	Dimensión: ResourceId	Unidad	Descripción
ReadDPU	Read Units	<cluster-id>	DPU	Approximates the read active-use component of your Aurora DSQL cluster DPU usage.
ComputeDPU	Compute Units	<cluster-id>	DPU	Approximates the compute active-use component of your Aurora DSQL cluster DPU usage.
TotalDPU	Total Units	<cluster-id>	DPU	Approximates the total active-use component of your Aurora DSQL cluster DPU usage.

Registro de operaciones de Aurora DSQL mediante AWS CloudTrail

Amazon Aurora DSQL está integrado con AWS CloudTrail, un servicio que proporciona un registro de las acciones realizadas por un usuario, un rol o un Servicio de AWS. Existen dos tipos de eventos en CloudTrail: eventos de administración y eventos de datos. Los eventos de administración se emiten para auditar los cambios de configuración de los recursos de AWS. Los eventos de datos capturan el uso de los recursos de AWS normalmente en el plano de datos de servicio.

CloudTrail captura todas las llamadas a la API para Aurora DSQL como eventos. Aurora DSQL registra la actividad de la consola como eventos de administración. También captura los intentos de conexión autenticados a los clústeres como eventos de datos.

Mediante la información que recopila CloudTrail, puede determinar la solicitud que se hizo a Aurora DSQL, la dirección IP desde la que se hizo la solicitud, cuándo se hizo, la identidad del usuario que hizo la solicitud y detalles adicionales.

CloudTrail está habilitado de forma predeterminada en la Cuenta de AWS cuando crea la cuenta y tiene acceso al Historial de eventos de CloudTrail. El Historial de eventos de CloudTrail proporciona un registro visible e inmutable, que se puede buscar y descargar, de los últimos 90 días de eventos de gestión registrados en una Región de AWS. Para obtener más información, consulte Trabajar con el historial de eventos de CloudTrail en la Guía del usuario de AWS CloudTrail. No se cobran cargos de CloudTrail por registrar el Historial de eventos.

Para crear un registro continuo de eventos en la cuenta de AWS, incluidos eventos para Aurora DSQL, cree un registro de seguimiento o un almacén de datos de eventos de AWS CloudTrail Lake (una solución centralizada de almacenamiento y análisis de eventos de AWS CloudTrail). Para obtener más información sobre la creación de registros de seguimiento, consulte Trabajar con registros de seguimiento de CloudTrail. Para obtener información sobre cómo configurar y administrar almacenes de datos de eventos, consulte Almacenes de datos de eventos de CloudTrail Lake.

Eventos de administración de Aurora DSQL en CloudTrail

Los eventos de administración de CloudTrail proporcionan información sobre las operaciones de administración que se realizan en los recursos de la cuenta de AWS. Se denominan también operaciones del plano de control. De forma predeterminada, CloudTrail captura los eventos de administración en el Historial de eventos.

Amazon Aurora DSQL registra todas las operaciones del plano de control de Aurora DSQL como eventos de administración. Para obtener una lista de las operaciones del plano de control de Amazon Aurora DSQL que Aurora DSQL registra en CloudTrail, consulte la referencia de la API de Aurora DSQL.

Registros del plano de control

Amazon Aurora DSQL registra las siguientes operaciones del plano de control de Aurora DSQL en CloudTrail como eventos de administración.

- CreateCluster
- DeleteCluster
- GetCluster
- GetVpcEndpointServiceName
- ListClusters
- ListTagsForResource
- TagResource
- UntagResource
- UpdateCluster

Registros de copia de seguridad y restauración

Amazon Aurora DSQL registra las siguientes operaciones de copia de seguridad y restauración de Aurora DSQL en CloudTrail como eventos de administración.

- `StartBackupJob`
- `StopBackupJob`
- `GetBackupJob`
- `StartRestoreJob`
- `StopRestoreJob`
- `GetRestoreJob`

Para obtener más información sobre la protección de los clústeres de Aurora DSQL mediante AWS Backup, consulte Copia de seguridad y restauración para Amazon Aurora DSQL.

Registros de AWS KMS

Amazon Aurora DSQL registra las siguientes operaciones de AWS KMS en CloudTrail como eventos de administración.

- `GenerateDataKey`
- `Decrypt`

Para obtener más información sobre cómo los registros de CloudTrail rastrean las solicitudes que Aurora DSQL envía a AWS KMS en su nombre, consulte Supervisión de la interacción de Aurora DSQL con AWS KMS.

Eventos de datos DSQL de Aurora en CloudTrail

Los eventos de datos de CloudTrail suelen proporcionar información sobre las operaciones realizadas en un recurso o dentro de él. También se utilizan para capturar las operaciones del plano de datos del servicio. Los eventos de datos suelen ser actividades de gran volumen. De forma predeterminada, CloudTrail no registra eventos de datos. El Historial de eventos de CloudTrail no registra los eventos de datos.

Para obtener más información sobre cómo registrar los eventos de datos, consulte Registro de eventos de datos con la AWS Management Console y Registro de eventos de datos con la AWS Command Line Interface en la Guía del usuario de AWS CloudTrail.

Se aplican cargos adicionales a los eventos de datos. Para obtener más información sobre los precios de CloudTrail, consulte Precios de AWS CloudTrail.

Para Aurora DSQL, CloudTrail captura cualquier intento de conexión realizado a un clúster de Aurora DSQL como un evento de datos. En la siguiente tabla se muestran los tipos de recursos de Aurora DSQL para los que puede registrar eventos de datos. La columna Tipo de recurso (consola) muestra el valor que se debe elegir en la lista Tipo de recurso en la consola de CloudTrail. La columna resources.type value muestra el valor de `resources.type`, que especificaría al configurar los selectores de eventos avanzados mediante la AWS CLI o las API de CloudTrail. La columna API de datos registradas en CloudTrail muestra las llamadas a la API registradas en CloudTrail para el tipo de recurso.

Tipo de recurso (consola)	resources.type value	API de datos registradas en CloudTrail
Amazon Aurora DSQL	`AWS::DSQL::Cluster`	• `DbConnect` • `DbConnectAdmin`

Puede configurar los selectores de eventos avanzados para filtrar en función de los campos `eventName` y `resources.ARN` y registrar solo los eventos filtrados. Para obtener más información acerca de estos campos, consulte AdvancedFieldSelector en la Referencia de la API de AWS CloudTrail.

En el siguiente ejemplo se muestra cómo utilizar AWS CLI para configurar `dsql-data-events-trail` y recibir eventos de datos para 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"] } ]}]'
```

Seguridad en Amazon Aurora DSQL

La seguridad en AWS es la principal prioridad. Como cliente de AWS, se beneficiará de una arquitectura de red y de centros de datos diseñados para satisfacer los requisitos de seguridad de las organizaciones más exigentes.

La seguridad es una responsabilidad compartida entre AWS y usted. El modelo de responsabilidad compartida la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta servicios de AWS en la Nube de AWS. AWS también le proporciona servicios que puede utilizar de forma segura. Los auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad como parte de los Programas de conformidad de AWS . Para obtener información sobre los programas de cumplimiento que se aplican a Amazon Aurora DSQL, consulte Servicios de AWS en el ámbito del programa de cumplimiento.
- Seguridad en la nube: su responsabilidad se determina según el servicio de AWSque utiliza. También eres responsable de otros factores, incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y la normativa aplicables.

Esta documentación lo ayuda a comprender cómo aplicar el modelo de responsabilidad compartida cuando se utiliza Aurora DSQL. En los siguientes temas, se le mostrará cómo configurar Aurora DSQL para satisfacer los objetivos de seguridad y cumplimiento. También puede aprender a utilizar otros servicios de AWS que lo ayuden a supervisar y proteger los recursos de Aurora DSQL.

Temas

- Seguridad de infraestructuras en Amazon Aurora DSQL
- Configuración y análisis de vulnerabilidades en Amazon Aurora DSQL
- Prevención de la sustitución confusa entre servicios
- Prácticas recomendadas de seguridad para Aurora DSQL

Políticas administradas de AWS para Amazon Aurora DSQL

Una política administrada de AWS es una política independiente que AWS crea y administra. Las políticas administradas de AWS se diseñan para ofrecer permisos para muchos casos de uso comunes, por lo que puede empezar a asignar permisos a los usuarios, grupos y roles.

Considere que es posible que las políticas administradas de AWS no concedan permisos de privilegio mínimo para los casos de uso concretos, ya que están disponibles para que las utilicen todos los clientes de AWS. Se recomienda definir políticas administradas por el cliente específicas para sus casos de uso a fin de reducir aún más los permisos.

No puede cambiar los permisos definidos en las políticas administradas de AWS. Si AWS actualiza los permisos definidos en una política administrada de AWS, la actualización afecta a todas las identidades de entidades principales (usuarios, grupos y roles) a las que está adjunta la política. Lo más probable es que AWS actualice una política administrada de AWS cuando se lance un nuevo Servicio de AWS o las operaciones de la API nuevas estén disponibles para los servicios existentes.

Para obtener más información, consulte Políticas administradas de AWS en la Guía del usuario de IAM.

Política administrada de AWS: AmazonAuroraDSQLFullAccess

Puede asociar `AmazonAuroraDSQLFullAccess` a los usuarios, grupos y roles.

Esta política concede permisos que permiten el acceso administrativo completo a Aurora DSQL. Las entidades principales con estos permisos pueden crear, eliminar y actualizar clústeres de Aurora DSQL, incluidos los clústeres multirregionales. Pueden agregar y eliminar etiquetas de los clústeres. Pueden enumerar clústeres y ver información sobre clústeres individuales. Pueden ver las

etiquetas asociadas a los clústeres de Aurora DSQL. Pueden conectarse a la base de datos como cualquier usuario, incluido el administrador. Pueden realizar operaciones de copia de seguridad y restauración para los clústeres de Aurora DSQL, como iniciar, detener y supervisar los trabajos de copia de seguridad y restauración. La política incluye permisos de AWS KMS que permiten realizar operaciones en las claves administradas por el cliente utilizadas para el cifrado de clústeres. Pueden ver cualquier métrica de CloudWatch en la cuenta. También tienen permisos para crear roles vinculados al servicio para el servicio `dsql.amazonaws.com`, lo que es necesario para crear clústeres.

Detalles de los permisos

Esta política incluye los siguientes permisos.

- `dsql`: concede a las entidades principales acceso completo a Aurora DSQL.
- `cloudwatch`: concede permiso para publicar puntos de datos de métricas en Amazon CloudWatch.
- `iam`: concede permiso para crear un rol vinculado a servicios.
- `backup and restore`: concede permisos para iniciar, detener y supervisar los trabajos de copia de seguridad y restauración de los clústeres de Aurora DSQL.
- `kms`: concede los permisos necesarios para validar el acceso a las claves administradas por el cliente utilizadas para el cifrado de clústeres de Aurora DSQL al crear, actualizar o conectarse a los clústeres.

Puede encontrar la política `AmazonAuroraDSQLFullAccess` en la consola de IAM y AmazonAuroraDSQLFullAccess en la Guía de referencia de las políticas administradas de AWS.

Política administrada de AWS: AmazonAuroraDSQLReadOnlyAccess

Puede asociar `AmazonAuroraDSQLReadOnlyAccess` a los usuarios, grupos y roles.

Permite el acceso de lectura a Aurora DSQL. Las entidades principales con estos permisos pueden enumerar clústeres y ver información sobre clústeres individuales. Pueden ver las etiquetas asociadas a los clústeres de Aurora DSQL. Pueden recuperar y ver cualquier métrica de CloudWatch en la cuenta.

Detalles de los permisos

Esta política incluye los siguientes permisos.

- `dsql`: concede permisos de solo lectura a todos los recursos de Aurora DSQL.
- `cloudwatch`: concede permiso para recuperar cantidades de lotes de datos de métricas de CloudWatch y realizar cálculos de métricas en función de los datos recuperados

Puede encontrar la política `AmazonAuroraDSQLReadOnlyAccess` en la consola de IAM y AmazonAuroraDSQLReadOnlyAccess en la Guía de referencia de las políticas administradas de AWS.

Política administrada de AWS: AmazonAuroraDSQLConsoleFullAccess

Puede asociar `AmazonAuroraDSQLConsoleFullAccess` a los usuarios, grupos y roles.

Permite el acceso administrativo completo a Amazon Aurora DSQL a través de la AWS Management Console. Las entidades principales con estos permisos pueden crear, eliminar y actualizar clústeres de Aurora DSQL, incluidos los clústeres multirregionales, con la consola. Pueden enumerar clústeres y ver información sobre clústeres individuales. Pueden ver las etiquetas de cualquier recurso de la cuenta. Pueden conectarse a la base de datos como cualquier usuario, incluido el administrador. Pueden realizar operaciones de copia de seguridad y restauración para los clústeres de Aurora DSQL, como iniciar, detener y supervisar los trabajos de copia de seguridad y restauración. La política incluye permisos de AWS KMS que permiten realizar operaciones en las claves administradas por el cliente utilizadas para el cifrado de clústeres. Puede lanzar AWS CloudShell desde la AWS Management Console. Pueden ver cualquier métrica de CloudWatch en la cuenta. También tienen permisos para crear roles vinculados al servicio para el servicio `dsql.amazonaws.com`, lo que es necesario para crear clústeres.

Puede encontrar la política `AmazonAuroraDSQLConsoleFullAccess` en la consola de IAM y AmazonAuroraDSQLConsoleFullAccess en la Guía de referencia de las políticas administradas de AWS.

Detalles de los permisos

Esta política incluye los siguientes permisos.

- `dsql`: concede permisos administrativos completos a todos los recursos de Aurora DSQL a través de la AWS Management Console.
- `cloudwatch`: concede permiso para recuperar cantidades de lotes de datos de métricas de CloudWatch y realizar cálculos de métricas en función de los datos recuperados.
- `tag`: concede permiso para devolver valores y claves de etiqueta actualmente en uso en la Región de AWS especificada para la cuenta de llamada.
- `backup and restore`: concede permisos para iniciar, detener y supervisar los trabajos de copia de seguridad y restauración de los clústeres de Aurora DSQL.
- `kms`: concede los permisos necesarios para validar el acceso a las claves administradas por el cliente utilizadas para el cifrado de clústeres de Aurora DSQL al crear, actualizar o conectarse a los clústeres.
- `cloudshell`: concede permisos para lanzar AWS CloudShell para interactuar con Aurora DSQL.
- `ec2`: concede permiso para ver la información del punto de conexión de Amazon VPC necesaria para las conexiones de Aurora DSQL.

Puede encontrar la política `AmazonAuroraDSQLReadOnlyAccess` en la consola de IAM y AmazonAuroraDSQLReadOnlyAccess en la Guía de referencia de las políticas administradas de AWS.

Política administrada de AWS: AuroraDSQLServiceRolePolicy

No puede asociar AuroraDSQLServiceRolePolicy a las entidades de IAM. Esta política está asociada a un rol vinculado al servicio que permite a Aurora DSQL acceder a los recursos de la cuenta.

Puede encontrar la política `AuroraDSQLServiceRolePolicy` en la consola de IAM y AuroraDSQLServiceRolePolicy en la Guía de referencia de las políticas administradas de AWS.

Actualizaciones de Aurora DSQL en las políticas administradas de AWS

Vea los detalles sobre las actualizaciones de las políticas administradas de AWS para Aurora DSQL desde que este servicio comenzó a realizar el seguimiento de estos cambios. Para obtener alertas automáticas sobre los cambios realizados en esta página, suscríbase a la fuente RSS en la Página del historial de revisión de Aurora DSQL.

Cambio	Descripción	Fecha
Actualización de AmazonAuroraDSQLFullAccess	Agrega la capacidad de realizar operaciones de copia de seguridad y restauración para los clústeres de Aurora DSQL, incluidos los trabajos de inicio, detención y supervisión. También agrega la capacidad de utilizar claves de KMS administradas por el cliente para el cifrado de clústeres. Para obtener más información, consulte AmazonAuroraDSQLFullAccess y Uso de roles vinculados a servicios en Aurora DSQL.	21 de mayo de 2025
Actualización de AmazonAuroraDSQLConsoleFullAccess	Agrega la capacidad de realizar operaciones de copia de seguridad y restauración para los clústeres de Aurora DSQL a través de AWS Console Home. Esto incluye iniciar, detener y supervisar los trabajos. También admite el uso de claves de KMS administradas por el cliente para el cifrado de clústeres y el lanzamiento de AWS CloudShell. Para obtener más información, consulte AmazonAuroraDSQLConsoleFullAccess	21 de mayo de 2025

Cambio	Descripción	Fecha
	y Uso de roles vinculados al servicio en Aurora DSQL.	
Actualización de AmazonAuroraDSQLFullAccess	La política agrega cuatro nuevos permisos para crear y administrar clústeres de bases de datos en múltiples Regiones de AWS: `PutMultiRegionProperties` , `PutWitnessRegion` , `AddPeerCluster` y `RemovePeerCluster` . Estos permisos incluyen controles de recursos y claves de condición para que pueda controlar qué usuarios de clústeres puede modificar. La política también agrega el permiso `GetVpcEndpointServiceName` para ayudarlo a conectarse a los clústeres de Aurora DSQL a través de AWS PrivateLink. Para obtener más información, consulte AmazonAuroraDSQLFullAccess y Uso de los roles vinculados al servicio en Aurora DSQL.	13 de mayo de 2025

Cambio	Descripción	Fecha
Actualización de AmazonAuroraDSQLReadOnlyAccess	Incluye la capacidad de determinar el nombre de servicio de punto de conexión de VPC correcto al conectarse a los clústeres de Aurora DSQL a través de AWS PrivateLink Aurora DSQL crea puntos de conexión únicos por celda, por lo que esta API lo ayuda a asegurarse de que puede identificar el punto de conexión correcto para el clúster y evitar errores de conexión. Para obtener más información, consulte AmazonAuroraDSQLReadOnlyAccess y Uso de roles vinculados al servicio en Aurora DSQL.	13 de mayo de 2025

Cambio	Descripción	Fecha
Actualización de AmazonAuroraDSQLConsoleFullAccess	Agrega nuevos permisos a Aurora DSQL para admitir la administración de clústeres multirregionales y la conexión de puntos de conexión de VPC. Los nuevos permisos incluyen `PutMultiRegionProperties` , `PutWitnessRegion` , `AddPeerCluster` , `RemovePeerCluster` y `GetVpcEndpointServiceName` Para obtener más información, consulte AmazonAuroraDSQLConsoleFullAccess y Uso de roles vinculados al servicio en Aurora DSQL.	13 de mayo de 2025

Cambio	Descripción	Fecha
Actualización de AuroraDsqlServiceLinkedRolePolicy	Agrega a la política la capacidad de publicar métricas en `AWS/AuroraDSQL` y los espacios de nombres de `AWS/Usage` `CloudWatch` . Esto permite que el servicio o el rol asociado emita datos de uso y rendimiento más completos al entorno de CloudWatch. Para obtener más información, consulte AuroraDsqlServiceLinkedRolePolicy y Uso de roles vinculados al servicio en Aurora DSQL.	8 de mayo de 2025
Página creada	Inicio del seguimiento de políticas administradas de AWS relacionadas con Amazon Aurora DSQL	3 de diciembre de 2024

Protección de datos en Amazon Aurora DSQL

El modelo de responsabilidad compartida se aplica a la protección de datos en . Como se describe en este modelo, es responsable de proteger la infraestructura global que ejecuta toda la Nube de AWS. Eres responsable de mantener el control sobre el contenido alojado en esta infraestructura. También eres responsable de las tareas de administración y configuración de seguridad para los que utiliza. Para obtener más información sobre la privacidad de los datos, consulta las Preguntas frecuentes sobre la privacidad de datos. Para obtener información sobre la protección de datos en Europa, consulta la publicación de blog sobre el Modelo de responsabilidad compartida de y GDPR en el Blog de seguridad de .

Con fines de protección de datos, recomendamos proteger las credenciales y configurar usuarios individuales con AWS IAM Identity Center o AWS Identity and Access Management. De esta manera,

solo se otorgan a cada usuario los permisos necesarios para cumplir sus obligaciones laborales. También recomendamos proteger sus datos de la siguiente manera:

- Utiliza la autenticación multifactor (MFA) en cada cuenta.
- Utiliza SSL/TLS para comunicarse con los recursos de . Se recomienda el uso de TLS 1.2 y recomendamos TLS 1.3.
- Configure los registros de API y de actividad de los usuarios con AWS CloudTrail. Para obtener información sobre cómo utilizar registros de seguimiento para capturar actividades, consulte Cómo trabajar con los registros de seguimiento en la Guía del usuario.
- Utilice las soluciones de cifrado, junto con todos los controles de seguridad predeterminados dentro de Servicios de AWS.
- Utiliza servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger los datos confidenciales almacenados en Amazon S3.

Se recomienda encarecidamente no ingresar nunca información confidencial o sensible, como por ejemplo, direcciones de correo electrónico de clientes, en etiquetas o campos de texto de formato libre, como el campo Nombre. Esto incluye las situaciones en las que debe trabajar con u otros mediante la consola, la API, la AWS CLI o los SDK de AWS. Cualquier dato que ingrese en etiquetas o campos de texto de formato libre utilizados para nombres se puede emplear para los registros de facturación o diagnóstico. Si proporciona una URL a un servidor externo, recomendamos encarecidamente que no incluya información de credenciales en la URL a fin de validar la solicitud para ese servidor.

Cifrado de datos

Amazon Aurora DSQL proporciona una infraestructura de almacenamiento de alta durabilidad diseñada para el almacenamiento de datos principales y críticos. Los datos se almacenan de forma redundante en varios dispositivos de diversas instalaciones dentro de una región de Aurora DSQL.

Cifrado en tránsito

De forma predeterminada, el cifrado en tránsito está configurado para usted. Aurora DSQL utiliza TLS para cifrar todo el tráfico entre el cliente de SQL y Aurora DSQL.

Cifrado y firma de datos en tránsito entre los clientes de la AWS CLI, el SDK o la API y los puntos de conexión de Aurora DSQL:

- Aurora DSQL proporciona puntos de conexión HTTPS para cifrar los datos en tránsito.
- Para proteger la integridad de las solicitudes de la API a Aurora DSQL, las llamadas a la API deben estar firmadas por el intermediario. Las llamadas se firman con un certificado X.509 o con la clave de acceso secreta de AWS del cliente, según el proceso de firma de Signature Version 4 (Sigv4). Para obtener más información, consulte Proceso de firma Signature Version 4 en la Referencia general de AWS.
- Use la AWS CLI o alguno de los AWS SDK para efectuar solicitudes a AWS. Estas herramientas firman automáticamente las solicitudes con la clave de acceso especificada al configurar las herramientas.

Para el cifrado en reposo, consulte Cifrado en reposo en Aurora DSQL.

Privacidad del tráfico entre redes

Las conexiones están protegidas tanto entre Aurora DSQL y las aplicaciones en las instalaciones como entre Aurora DSQL y otros recursos de AWS dentro de la misma Región de AWS.

Tiene dos opciones de conectividad entre su red privada y AWS:

- Una conexión de Site-to-Site VPN de AWS. Para obtener más información, consulte ¿Qué es AWS Site-to-Site VPN?
- Una conexión de AWS Direct Connect. Para obtener más información, consulte ¿Qué es AWS Direct Connect?

Obtendrá acceso a Aurora DSQL a través de la red mediante las operaciones de la API publicadas por AWS. Los clientes deben admitir lo siguiente:

- Seguridad de la capa de transporte (TLS). Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Conjuntos de cifrado con confidencialidad directa total (PFS) como DHE (Ephemeral Diffie-Hellman) o ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Configuración de certificados SSL/TLS para conexiones de Aurora DSQL

Aurora DSQL requiere que todas las conexiones utilicen el cifrado de seguridad de la capa de transporte (TLS). Para establecer conexiones seguras, el sistema cliente debe confiar en la Autoridad

de certificación raíz de Amazon (Amazon Root CA 1). Este certificado viene preinstalado en muchos sistemas operativos. Esta sección proporciona instrucciones para verificar el certificado de Amazon Root CA 1 preinstalado en varios sistemas operativos y le guía a través del proceso de instalación manual del certificado si aún no está presente.

Recomendamos utilizar la versión 17 de PostgreSQL.

> **Important**
>
> Para los entornos de producción, recomendamos utilizar el modo de SSL `verify-full` para garantizar el máximo nivel de seguridad de la conexión. Este modo verifica que el certificado del servidor esté firmado por una autoridad de certificación de confianza y que el nombre de host del servidor coincida con el certificado.

Verificación de certificados preinstalados

En la mayoría de los sistemas operativos, Amazon Root CA 1 ya viene preinstalado. Para validarlo, puede seguir los pasos que se indican a continuación.

Linux (RedHat/CentOS/Fedora)

Ejecute el siguiente comando en el terminal:

```
trust list | grep "Amazon Root CA 1"
```

Si el certificado está instalado, verá el siguiente resultado:

```
label: Amazon Root CA 1
```

macOS

1. Abra la búsqueda Spotlight (Comando + Espacio)
2. Búsqueda de Keychain Access
3. Selección de System Roots en System Keychains
4. Búsqueda de Amazon Root CA 1 en la lista de certificados

Windows

> **Note**
>
> Debido a un problema conocido con el cliente psql de Windows, el uso de certificados raíz del sistema (`sslrootcert=system`) puede devolver el siguiente error: `SSL error: unregistered scheme`. Puede seguir Conexión desde Windows como forma alternativa de conectarse al clúster mediante SSL.

Si Amazon Root CA 1 no está instalado en el sistema operativo, siga los pasos que se indican a continuación.

Instalación de certificados

Si el certificado `Amazon Root CA 1` no está preinstalado en el sistema operativo, tendrá que instalarlo de forma manual para establecer conexiones seguras con el clúster de Aurora DSQL.

Instalación de certificados de Linux

Siga estos pasos para instalar el certificado de entidad de certificación de Amazon Root en sistemas de Linux.

1. Descargue el certificado raíz:

```
wget https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

2. Copie el certificado en el almacén de confianza:

```
sudo cp ./AmazonRootCA1.pem /etc/pki/ca-trust/source/anchors/
```

3. Actualice el almacén de confianza de la entidad de certificación:

```
sudo update-ca-trust
```

4. Verificar la instalación:

```
trust list | grep "Amazon Root CA 1"
```

Instalación de certificado de macOS

Estos pasos de instalación del certificado son opcionales. Instalación de certificados de Linux también funciona para macOS.

1. Descargue el certificado raíz:

```
wget https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

2. Agregue el certificado al llavero del sistema:

```
sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain
 AmazonRootCA1.pem
```

3. Verificar la instalación:

```
security find-certificate -a -c "Amazon Root CA 1" -p /Library/Keychains/
System.keychain
```

Conexión con verificación SSL/TLS

Antes de configurar los certificados SSL/TLS para conexiones seguras al clúster de Aurora DSQL, asegúrese de cumplir los siguientes requisitos previos.

- Se ha instalado la versión 17 de PostgreSQL
- La AWS CLI se ha configurado con las credenciales apropiadas
- Información del punto de conexión del clúster de Aurora DSQL

Conexión desde Linux

1. Genere y configure el token de autenticación:

```
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token --region=your-
cluster-region --hostname your-cluster-endpoint)
```

2. Conéctese mediante certificados del sistema (si están preinstalados):

```
PGSSLROOTCERT=system \
PGSSLMODE=verify-full \
```

```
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

3. O bien, conéctese mediante un certificado descargado:

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

> **Note**
>
> Para obtener más información sobre la configuración de PGSSLMODE, consulte sslmode en la documentación de Database Connection Control Functions de PostgreSQL 17.

Conexión desde macOS

1. Genere y configure el token de autenticación:

```
export PGPASSWORD=$(aws dsql generate-db-connect-admin-auth-token --region=your-cluster-region --hostname your-cluster-endpoint)
```

2. Conéctese mediante certificados del sistema (si están preinstalados):

```
PGSSLROOTCERT=system \
PGSSLMODE=verify-full \
psql --dbname postgres \
--username admin \
--host your-cluster-endpoint
```

3. O bien, descargue el certificado raíz y guárdelo como `root.pem` (si el certificado no está preinstalado)

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql –dbname postgres \
--username admin \
```

```
--host your_cluster_endpoint
```

4. Conexión mediante psql:

```
PGSSLROOTCERT=/full/path/to/root.pem \
PGSSLMODE=verify-full \
psql –dbname postgres \
--username admin \
--host your_cluster_endpoint
```

Conexión desde Windows

Uso del símbolo del sistema

1. Generación del token de autenticación:

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

2. Establezca la variables de entorno de contraseña:

```
set "PGPASSWORD=token-from-above"
```

3. Establezca la configuración de SSL:

```
set PGSSLROOTCERT=C:\full\path\to\root.pem
set PGSSLMODE=verify-full
```

4. Conexión a la base de datos:

```
"C:\Program Files\PostgreSQL\17\bin\psql.exe" --dbname postgres ^
--username admin ^
--host your-cluster-endpoint
```

Con PowerShell

1. Genere y configure el token de autenticación:

```
$env:PGPASSWORD = (aws dsql generate-db-connect-admin-auth-token --region=your-cluster-region --expires-in=3600 --hostname=your-cluster-endpoint)
```

2. Establezca la configuración de SSL:

```
$env:PGSSLROOTCERT='C:\full\path\to\root.pem'
$env:PGSSLMODE='verify-full'
```

3. Conexión a la base de datos:

```
 "C:\Program Files\PostgreSQL\17\bin\psql.exe" --dbname postgres `
--username admin `
--host your-cluster-endpoint
```

Recursos adicionales

- Documentación de PostgreSQL SSL
- Servicios de confianza de Amazon

Cifrado de datos para Amazon Aurora DSQL

Amazon Aurora DSQL cifra todos los datos en reposo del usuario. Para mejorar la seguridad, este cifrado utiliza AWS Key Management Service (AWS KMS). Esta funcionalidad ayuda a reducir la carga y la complejidad operativas que conlleva la protección de información confidencial. El cifrado en reposo le ayuda a:

- Reducción de la carga operativa de proteger la información confidencial
- Creación de aplicaciones sensibles a la seguridad que necesitan cumplimiento estricto de cifrado y requisitos normativos
- Agregación de una capa adicional de protección de datos protegiendo siempre los datos en un clúster cifrado
- Conformidad con las políticas de la organización, las normativas del sector o gubernamentales y los requisitos de conformidad

Con Aurora DSQL, puede crear aplicaciones sensibles a la seguridad que necesitan cumplimiento estricto de cifrado y requisitos normativos. En las siguientes secciones se explica cómo configurar el cifrado para las bases de datos de Aurora DSQL nuevas y existentes y cómo administrar las claves de cifrado.

Temas

Tipos de claves de KMS para Aurora DSQL

Aurora DSQL se integra con AWS KMS para administrar las claves de cifrado de los clústeres. Para obtener más información acerca de los tipos y estados de las claves, consulte los conceptos de AWS Key Management Service en la Guía para desarrolladores de AWS Key Management Service. Al crear un clúster nuevo, puede elegir entre los siguientes tipos de claves de KMS para cifrar el clúster:

Clave propiedad de AWS

> Tipo de cifrado predeterminado. Aurora DSQL es el propietario de la clave sin cargo adicional para usted. Amazon Aurora DSQL descifra de forma transparente los datos del clúster cuando accede a un clúster cifrado. No es necesario que cambie el código o las aplicaciones para utilizar o administrar clústeres cifrados y todas las consultas de Aurora DSQL funcionan con los datos cifrados.

Clave administrada por el cliente

> Puede crear, poseer y administrar la clave en la Cuenta de AWS. Tiene control total de la clave de KMS. Se aplican los cargos de AWS KMS.

El cifrado en reposo mediante la Clave propiedad de AWS está disponible sin cargo adicional. Sin embargo, se aplican cargos de AWS KMS para las claves administradas por el cliente. Para obtener más información, consulte la Página de precios de AWS KMS.

Puede cambiar entre estos tipos de claves en cualquier momento. Para obtener más información sobre los tipos de claves, consulte Claves administradas por el cliente y Claves propiedad de AWS en la Guía para desarrolladores de AWS Key Management Service.

> **Note**
>
> El cifrado en reposo de Aurora DSQL está disponible en todas las regiones de AWS donde esté disponible Aurora DSQL.

Cifrado en reposo en Aurora DSQL

Amazon Aurora DSQL utiliza el estándar de cifrado avanzado de 256 bits (AES-256) para cifrar los datos en reposo. Este cifrado ayuda a proteger los datos del acceso no autorizado al almacenamiento subyacente. AWS KMS administra las claves de cifrado de los clústeres. Puede usar la opción Claves propiedad de AWS predeterminada, o elegir usar su propia Claves administradas por el cliente de AWS KMS. Para obtener más información sobre cómo especificar y administrar las claves de los clústeres de Aurora DSQL, consulte Creación de un clúster de Aurora DSQL cifrado y Eliminación o actualización de una clave para el clúster de Aurora DSQL.

Temas

- Claves propiedad de AWS
- Claves administradas por el cliente

Claves propiedad de AWS

Aurora DSQL cifra todos los clústeres de forma predeterminada con Claves propiedad de AWS. Estas claves son de uso gratuito y se rotan anualmente para proteger los recursos de la cuenta. No necesita ver, administrar, usar ni auditar estas claves, por lo que no es necesario realizar ninguna acción para proteger los datos. Para obtener más información acerca de Claves propiedad de AWS, consulte Claves propiedad de AWS en la Guía para desarrolladores de AWS Key Management Service.

Claves administradas por el cliente

Cree, posea y administre las claves administradas por el cliente en la Cuenta de AWS. Tiene el control total sobre estas claves de KMS, incluidas sus políticas, el material de cifrado, las etiquetas y los alias. Para obtener más información acerca de cómo administrar los permisos, consulte Claves administradas por el cliente en la Guía para desarrolladores de AWS Key Management Service.

Cuando especifica una clave administrada por el cliente para el cifrado en el nivel de clúster, Aurora DSQL cifra el clúster y todos sus datos regionales con esa clave. Para evitar la pérdida de datos y mantener el acceso al clúster, Aurora DSQL necesita acceder a la clave de cifrado. Si desactiva la clave administrada por el cliente, programa su eliminación o tiene una política que restringe el acceso al servicio, el estado de cifrado del clúster cambia a `KMS_KEY_INACCESSIBLE`. Cuando Aurora DSQL no puede acceder a la clave, los usuarios no pueden conectarse al clúster, el estado de cifrado del clúster cambia a `KMS_KEY_INACCESSIBLE` y el servicio pierde el acceso a los datos del clúster.

En el caso de los clústeres de varias regiones, los clientes pueden configurar la clave de cifrado de AWS KMS de cada región de forma independiente y cada clúster regional utiliza su propia clave de cifrado en el nivel de clúster. Si Aurora DSQL no puede acceder a la clave de cifrado de un par en un clúster de varias regiones, el estado de ese par pasa a ser `KMS_KEY_INACCESSIBLE` y deja de estar disponible para las operaciones de lectura y escritura. Los demás pares continúan con sus operaciones normales.

> **Note**
>
> Si Aurora DSQL no puede acceder a su clave administrada por el cliente, el estado de cifrado del clúster cambia a `KMS_KEY_INACCESSIBLE`. Tras restaurar el acceso de la clave, el servicio detectará automáticamente la restauración en 15 minutos. Para obtener más información, consulte clúster en espera.
> Para los clústeres de varias regiones, si se pierde el acceso de la clave durante un periodo prolongado, el tiempo de restauración del clúster depende de la cantidad de datos que se hayan escrito mientras la clave no estaba accesible.

Uso de AWS KMS y claves de datos con Aurora DSQL

La característica de cifrado en reposo de Aurora DSQL utiliza una AWS KMS key y una jerarquía de claves de datos para proteger los datos del clúster.

Le recomendamos que planifique la estrategia de cifrado antes de implementar el clúster en Aurora DSQL. Si almacena datos confidenciales en Aurora DSQL, considere incluir el cifrado del cliente en el plan. De esta forma, puede cifrar los datos lo más cerca posible del origen y garantizar la protección durante todo el ciclo de vida.

Temas

Uso de AWS KMS key con Aurora DSQL

El cifrado en reposo protege el clúster de Aurora DSQL con una AWS KMS key. De forma predeterminada, Aurora DSQL utiliza una Clave propiedad de AWS, una clave de cifrado de varios inquilinos que se crea y administra en una cuenta de servicio de Aurora DSQL. Sin embargo, puede cifrar los clústeres de Aurora DSQL con una clave administrada por el cliente en la Cuenta de AWS. Puede seleccionar una clave de KMS diferente para cada clúster, incluso si participa en una configuración de varias regiones.

Seleccione la clave de KMS para un clúster al crear o actualizar el clúster. Puede cambiar la clave de KMS de un clúster en cualquier momento, ya sea en la consola de Aurora DSQL o utilizando la operación `UpdateCluster`. El proceso de cambio de claves no precisa tiempo de inactividad ni degradación del servicio.

> **Important**
>
> Aurora DSQL solo admite claves de KMS simétricas. No puede utilizar una clave de KMS asimétrica para cifrar los clústeres de Aurora DSQL.

Una clave administrada por el cliente proporciona los siguientes beneficios.

- Puede crear y administrar la clave de KMS, incluida la configuración de políticas de claves y políticas de IAM para controlar el acceso a la clave de KMS. Puede habilitar y deshabilitar la clave KMS, habilitar y deshabilitar la rotación de claves automática y eliminar la clave KMS cuando ya no esté en uso.

- Puede utilizar una clave administrada por el cliente con material de claves importado o una clave administrada por el cliente en un almacén de claves personalizado que tenga y administre.
- Puede auditar el cifrado y descifrado del clúster de Aurora DSQL examinando las llamadas a la API de Aurora DSQL a AWS KMS en los registros de AWS CloudTrail.

Sin embargo, la Clave propiedad de AWS es gratuita y su uso no se contabiliza en las cuotas de solicitudes o de recursos de AWS KMS. Las claves administradas por el cliente generan cargos por cada llamada a la API y se aplican cuotas de AWS KMS a estas claves.

Uso de claves de clúster con Aurora DSQL

Aurora DSQL utiliza la AWS KMS key para el clúster para generar y cifrar una clave de datos única para el clúster, conocida como clave de clúster.

La clave del clúster se utiliza como clave de cifrado de claves. Aurora DSQL utiliza esta clave de clúster para proteger las claves de cifrado de datos que se utilizan para cifrar los datos del clúster. Aurora DSQL genera una clave de cifrado de datos única para cada estructura subyacente en un clúster, pero varios elementos del clúster pueden protegerse mediante la misma clave de cifrado de datos.

Para descifrar la clave del clúster, Aurora DSQL envía una solicitud a AWS KMS cuando se accede por primera vez a un clúster cifrado. Para mantener el clúster disponible, Aurora DSQL verifica periódicamente el acceso de descifrado a la clave de KMS, incluso cuando no se está accediendo de forma activa al clúster.

Aurora DSQL almacena y utiliza la clave de clúster y las claves de cifrado de datos fuera de AWS KMS. Protege todas las claves con cifrado Advanced Encryption Standard (AES) y claves de cifrado de 256 bits. A continuación, almacena las claves cifradas con los datos cifrados para que estén disponibles para descifrar los datos del clúster bajo demanda.

Si cambia la clave de KMS del clúster, Aurora DSQL vuelve a cifrar la clave de clúster existente con la nueva clave de KMS.

Almacenamiento en caché de la clave de clúster

Para evitar llamar a AWS KMS para cada operación de Aurora DSQL, Aurora DSQL almacena en la memoria caché las claves del clúster de texto no cifrado para cada intermediario. Si Aurora DSQL recibe una solicitud de la clave de clúster almacenada en caché tras 15 minutos de inactividad,

envía una nueva solicitud a AWS KMS para descifrar la clave de clúster. Esta llamada capturará los cambios realizados en las políticas de acceso de la AWS KMS key, AWS KMS o AWS Identity and Access Management (IAM) desde la última solicitud para descifrar la clave del clúster.

Autorización del uso de la AWS KMS key para Aurora DSQL

Si utiliza una clave administrada por el cliente en la cuenta para proteger el clúster de Aurora DSQL, las políticas en esa clave deben conceder permiso a Aurora DSQL para utilizarla en su nombre.

Tiene control total sobre las políticas en una clave administrada por el cliente. Aurora DSQL no necesita autorización adicional para utilizar la Clave propiedad de AWS predeterminada para proteger los clústeres de Aurora DSQL de la Cuenta de AWS.

Política de claves para una clave administrada por el cliente

Cuando selecciona una clave administrada por el cliente para proteger un clúster de Aurora DSQL, Aurora DSQL necesita permiso para utilizar la AWS KMS key en nombre de la entidad principal que realiza la selección. Esa entidad principal, un usuario o rol, deben tener los permisos en la AWS KMS key que precisa Aurora DSQL. Puede proporcionar estos permisos en una política de claves o en una política de IAM.

Como mínimo, Aurora DSQL precisa los siguientes permisos en una clave administrada por el cliente:

- `kms:Encrypt`
- `kms:Decrypt`
- `kms:ReEncrypt*` (para kms:ReEncryptFrom y kms:ReEncryptTo)
- `kms:GenerateDataKey`
- `kms:DescribeKey`

Por ejemplo, la política de claves de ejemplo siguiente proporciona solo los permisos necesarios. La política tiene las siguientes consecuencias:

- Permite a Aurora DSQL utilizar la AWS KMS key en operaciones criptográficas, pero solo cuando actúa en nombre de las entidades principales de la cuenta que tienen permiso para usar Aurora DSQL. Si las entidades principales especificadas en la declaración de política no tienen permiso para utilizar Aurora DSQL, la llamada produce un error, incluso cuando proviene del servicio de Aurora DSQL.

- La clave de condición `kms:ViaService` permite los permisos solo cuando la solicitud proviene de Aurora DSQL en nombre de las entidades principales mostradas en la declaración de política. Estas entidades principales no pueden llamar a estas operaciones directamente.
- Otorga a los administradores de AWS KMS key (usuarios que pueden asumir el rol `db-team`) acceso de solo lectura a AWS KMS key

Antes de utilizar una política de claves de ejemplo, sustituya las entidades principales de ejemplo por las entidades principales reales de su cuenta de Cuenta de 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"
    }
```

```
  }
}
```

Contexto de cifrado de Aurora DSQL

Un contexto de cifrado es un conjunto de pares de clave-valor que contienen datos no secretos arbitrarios. Cuando se incluye un contexto de cifrado en una solicitud para cifrar datos, AWS KMS vincula criptográficamente el contexto de cifrado a los datos cifrados. Para descifrar los datos, es necesario pasar el mismo contexto de cifrado.

Aurora DSQL utiliza el mismo contexto de cifrado en todas las operaciones criptográficas de AWS KMS. Si utiliza una clave administrada por el cliente para proteger el clúster de Aurora DSQL, puede utilizar el contexto de cifrado para identificar el uso de la AWS KMS key en los registros de auditoría y en los registros. También aparece en texto sin formato en registros, como aquellos en AWS CloudTrail.

El contexto de cifrado también se puede utilizar como condición para la autorización en políticas.

En sus solicitudes a AWS KMS, Aurora DSQL utiliza un contexto de cifrado con un par clave-valor:

```
"encryptionContext": {
  "aws:dsql:ClusterId": "w4abucpbwuxx"
},
```

El par clave-valor identifica el clúster que Aurora DSQL está cifrando. La clave es `aws:dsql:ClusterId`. El valor es el identificador del clúster.

Supervisión de la interacción de Aurora DSQL con AWS KMS

Si utiliza una clave administrada por el cliente para proteger los clústeres de Aurora DSQL, puede utilizar los registros de AWS CloudTrail para realizar un seguimiento de las solicitudes que Aurora DSQL envía a AWS KMS en su nombre.

Expanda las siguientes secciones para obtener información sobre cómo Aurora DSQL utiliza las operaciones de AWS KMS `GenerateDataKey` y `Decrypt`.

`GenerateDataKey`

Cuando habilita el cifrado en reposo en un clúster, Aurora DSQL crea una clave de clúster única. Envía una solicitud de `GenerateDataKey` a AWS KMS que especifica la AWS KMS key del clúster.

El evento que registra la operación `GenerateDataKey` es similar al siguiente evento de ejemplo. El usuario es la cuenta del servicio de Aurora DSQL. Los parámetros incluyen el Nombre de recurso de Amazon (ARN) de la AWS KMS key, un especificador de claves que requiere una clave de 256 bits y el contexto de cifrado que identifica el clúster.


```
{
    "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"
}
```

Decrypt

Cuando accede a un clúster de Aurora DSQL cifrado, Aurora DSQL necesita descifrar la clave del clúster de manera que pueda descifrar las claves que hay debajo en la jerarquía. A continuación, descifra los datos del clúster. Para descifrar la clave del clúster, Aurora DSQL envía una solicitud `Decrypt` a AWS KMS que especifica la AWS KMS key del clúster.

El evento que registra la operación `Decrypt` es similar al siguiente evento de ejemplo. El usuario es la entidad principal en la Cuenta de AWS que está accediendo al clúster. Los parámetros incluyen la clave del clúster cifrada (como blob de texto cifrado) y el contexto de cifrado que identifica el clúster. AWS KMS deriva el ID de la AWS KMS key del texto cifrado.

```
{
  "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"
}
```

Creación de un clúster de Aurora DSQL cifrado

Todos los clústeres de Aurora DSQL se cifran en reposo. De forma predeterminada, los clústeres utilizan una Clave propiedad de AWS sin costo o puede especificar una clave de AWS KMS personalizada. Siga estos pasos para crear el clúster cifrado desde la AWS Management Console o desde la AWS CLI.

Console

Creación de un clúster cifrado en la AWS Management Console

1. Inicie sesión en la Consola de administración de AWS y abra la consola de Aurora DSQL en https://console.aws.amazon.com/dsql/.
2. En el panel de navegación en el lado izquierdo de la consola, en DAX, elija Clústeres.
3. Elija Crear clúster en la parte superior derecha y seleccione Región única.
4. En la Configuración de cifrado del clúster, elija una de las siguientes opciones.
   - Acepte la configuración predeterminada para cifrar con una Clave propiedad de AWS sin costo adicional.

- Seleccione Personalizar la configuración de cifrado (avanzada) para especificar una clave de KMS personalizada. A continuación, busque o ingrese el ID o el alias de la clave de KMS. Otra opción, elija Crear una clave de AWS KMS para crear una clave nueva en la consola de AWS KMS.

5. Elija Create cluster.

Para confirmar el tipo de cifrado del clúster, vaya a la página Clústeres y seleccione el ID del clúster para ver los detalles del clúster. Revise la pestaña Configuración del clúster. La configuración de la clave de KMS del clúster muestra la clave predeterminada de Aurora DSQL para los clústeres que utilizan claves propiedad de AWS o el ID de clave para otros tipos de cifrado.

> **Note**
>
> Si decide ser propietario y administrar su propia clave, asegúrese de establecer la política de claves de KMS adecuada. Para obtener más información y ejemplos, consulte the section called “Política de claves para una clave administrada por el cliente”.

CLI

Creación de un clúster que esté cifrado con la Clave propiedad de AWS predeterminada

- Utilice el siguiente comando para crear un clúster de Aurora DSQL.

```
aws dsql create-cluster
```

Como se muestra en los siguientes detalles de cifrado, el estado de cifrado del clúster está habilitado de forma predeterminada y el tipo de cifrado predeterminado es la clave propiedad de AWS. El clúster ahora está cifrado con la clave predeterminada propiedad de AWS en la cuenta de servicio de Aurora DSQL.

```
"encryptionDetails": {
  "encryptionType" : "AWS_OWNED_KMS_KEY",
  "encryptionStatus" : "ENABLED"
}
```

Creación de un clúster cifrado con la clave administrada por el cliente

- Utilice el siguiente comando para crear un clúster de Aurora DSQL y sustituya el ID de clave en texto rojo por el ID de la clave administrada por el cliente.

```
aws dsql create-cluster \
--kms-encryption-key d41d8cd98f00b204e9800998ecf8427e
```

Como se muestra en los siguientes detalles de cifrado, el estado de cifrado del clúster está habilitado de forma predeterminada y el tipo de cifrado es la clave de KMS administrada por el cliente. El clúster ahora está cifrado con su clave.

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:kms:us-east-1:111122223333:key/
d41d8cd98f00b204e9800998ecf8427e",
  "encryptionStatus" : "ENABLED"
}
```

Eliminación o actualización de una clave para el clúster de Aurora DSQL

Puede utilizar la AWS Management Console o la AWS CLI para actualizar o eliminar las claves de cifrado de los clústeres existentes en Amazon Aurora DSQL. Si quita una clave sin sustituirla, Aurora DSQL utilizará la Clave propiedad de AWS predeterminada. Siga estos pasos para actualizar las claves de cifrado de un clúster existente desde la consola de Aurora DSQL o desde la AWS CLI.

Console

Actualización o eliminación de una clave de cifrado en la AWS Management Console

1. Inicie sesión en la Consola de administración de AWS y abra la consola de Aurora DSQL en https://console.aws.amazon.com/dsql/.
2. En el panel de navegación en el lado izquierdo de la consola, en DAX, elija Clústeres.
3. En la vista de lista, busque y seleccione la fila del clúster que desea actualizar.
4. Seleccione el menú Acciones y, a continuación, elija Modificar.
5. En la Configuración de cifrado del clúster, elija una de las siguientes opciones para modificar la configuración de cifrado.

- Si desea cambiar de una clave personalizada a una Clave propiedad de AWS, desactive la opción Personalizar la configuración de cifrado (avanzada). Se aplicará la configuración predeterminada y se cifrará el clúster con una Clave propiedad de AWS sin costo alguno.
- Si desea cambiar de una clave de KMS personalizada a otra o de una Clave propiedad de AWS a una clave de KMS, seleccione la opción Personalizar la configuración de cifrado (avanzada) si aún no está seleccionada. A continuación, busque y seleccione el ID o el alias de la clave que desee utilizar. Otra opción, elija Crear una clave de AWS KMS para crear una clave nueva en la consola de AWS KMS.

6. Seleccione Save.

CLI

Los siguientes ejemplos muestran cómo usar la AWS CLI para actualizar un clúster cifrado.

Actualización de un clúster cifrado con la Clave propiedad de AWS predeterminada

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key "AWS_OWNED_KMS_KEY"
```

El `EncryptionStatus` de la descripción del clúster se establece en `ENABLED` y el `EncryptionType` es `AWS_OWNED_KMS_KEY`.

```
"encryptionDetails": {
  "encryptionType" : "AWS_OWNED_KMS_KEY",
  "encryptionStatus" : "ENABLED"
}
```

Este clúster está ahora cifrado mediante la Clave propiedad de AWS predeterminada en la cuenta de servicio de Aurora DSQL.

Actualización de un clúster cifrado con una clave administrada por el cliente para Aurora DSQL

Actualice el clúster cifrado, como en el siguiente ejemplo:

```
aws dsql update-cluster \
--identifier aiabtx6icfp6d53snkhseduiqq \
--kms-encryption-key arn:aws:kms:us-east-1:123456789012:key/abcd1234-abcd-1234-a123-ab1234a1b234
```

El `EncryptionStatus` de la descripción del clúster pasa a `UPDATING` y el `EncryptionType` es `CUSTOMER_MANAGED_KMS_KEY`. Cuando Aurora DSQL termine de propagar la nueva clave a través de la plataforma, el estado de cifrado pasará a `ENABLED`

```
"encryptionDetails": {
  "encryptionType" : "CUSTOMER_MANAGED_KMS_KEY",
  "kmsKeyArn" : "arn:aws:us-east-1:kms:key/abcd1234-abcd-1234-a123-ab1234a1b234",
  "encryptionStatus" : "ENABLED"
}
```

> **Note**
>
> Si decide ser propietario y administrar su propia clave, asegúrese de establecer la política de claves de KMS adecuada. Para obtener más información y ejemplos, consulte the section called “Política de claves para una clave administrada por el cliente”.

Consideraciones sobre el cifrado con Aurora DSQL

- Aurora DSQL cifra todos los datos en reposo del clúster. No puede desactivar este cifrado ni cifrar solo algunos elementos de un clúster.
- AWS Backup cifra las copias de seguridad y los clústeres restaurados a partir de estas copias de seguridad. Puede cifrar los datos de copia de seguridad en AWS Backup mediante la clave propiedad de AWS o una clave administrada por el cliente.
- Los siguientes estados de protección de datos están habilitados para Aurora DSQL:
  - Datos en reposo: Aurora DSQL cifra todos los datos estáticos de los medios de almacenamiento persistentes
  - Datos en tránsito: Aurora DSQL cifra todas las comunicaciones mediante la seguridad de la capa de transporte (TLS) de forma predeterminada

- Cuando realice la transición a una clave diferente, le recomendamos que mantenga la clave original habilitada hasta que se complete la transición. AWS necesita la clave original para descifrar los datos antes de cifrarlos con la nueva clave. El proceso finaliza cuando el `encryptionStatus` del clúster es `ENABLED` y usted ve el `kmsKeyArn` de la nueva clave administrada por el cliente.
- Cuando desactiva la clave administrada por el cliente o revoca el acceso para que Aurora DSQL utilice su clave, el clúster pasará al estado `IDLE`.
- La API de AWS Management Console y de Amazon Aurora DSQL utilizan términos diferentes para los tipos de cifrado:
  - Consola de AWS: en la consola, verá `KMS` cuando use una clave administrada por el cliente y `DEFAULT` cuando use una Clave propiedad de AWS.
  - API: la API de Amazon Aurora DSQL utiliza `CUSTOMER_MANAGED_KMS_KEY` para las claves administradas por el cliente y `AWS_OWNED_KMS_KEY` para Claves propiedad de AWS.
- Si no especifica una clave de cifrado durante la creación del clúster, Aurora DSQL cifra automáticamente los datos mediante Clave propiedad de AWS.
- Puede alternar entre una Clave propiedad de AWS y una clave administrada por el cliente en cualquier momento. Realice este cambio mediante la AWS Management Console, la AWS CLI o la API de Amazon Aurora DSQL.

Administración de la identidad y el acceso para Aurora DSQL

AWS Identity and Access Management (IAM) es un Servicio de AWS que ayuda a los gestionadores a controlar de forma segura el acceso a los recursos de AWS. Los administradores de IAM controlan a qué personas se puede autenticar (pueden iniciar sesión) y autorizar (tienen permisos) para utilizar recursos de Aurora DSQL. IAM es un servicio de Servicio de AWS que se puedes utilizar sin cargo adicional.

Temas

Público

La forma en que utilice AWS Identity and Access Management (IAM) difiere en función del trabajo que realice en Aurora DSQL.

Usuario del servicio: si utiliza el servicio Aurora DSQL para hacer el trabajo, el administrador le proporciona las credenciales y los permisos que necesita. A medida que utilice más características de Aurora DSQL para realizar el trabajo, es posible que necesite permisos adicionales. Entender cómo se administra el acceso puede ayudarle a solicitar los permisos correctos al administrador. Si no puede acceder a una característica en Aurora DSQL, consulte Solución de problemas de identidades y accesos en Amazon Aurora DSQL.

Administrador de servicio: si está a cargo de los recursos de Aurora DSQL en la empresa, probablemente tenga acceso completo a Aurora DSQL. Su trabajo consiste en determinar a qué características y recursos de Aurora DSQL deben acceder los usuarios del servicio. Luego, debe enviar solicitudes a su gestionador de IAM para cambiar los permisos de los usuarios de su servicio. Revise la información de esta página para conocer los conceptos básicos de IAM. Para obtener más información sobre cómo la empresa puede utilizar IAM con Aurora DSQL, consulte Cómo funciona Amazon Aurora DSQL con IAM.

Administrador de IAM: si es un administrador de IAM, es posible que quiera conocer más detalles sobre cómo escribir políticas para administrar el acceso a Aurora DSQL. Para ver ejemplos de políticas de Aurora DSQL basadas en identidades que puede utilizar en IAM, consulte Ejemplos de políticas basadas en identidad para Amazon Aurora DSQL.

Autenticación con identidades

La autenticación es la manera de iniciar sesión en AWS mediante credenciales de identidad. Debe estar autenticado (haber iniciado sesión en AWS) como Usuario raíz de la cuenta de AWS, como un usuario de IAM o asumiendo un rol de IAM.

Puede iniciar sesión en AWS como una identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad. AWS IAM Identity Center Los usuarios (del IAM Identity Center), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, su gestionador habrá configurado previamente la federación de identidades mediante roles de IAM. Cuando accede a AWS mediante la federación, está asumiendo un rol de forma indirecta.

Según el tipo de usuario que sea, puede iniciar sesión en AWS Management Console o en el portal de acceso AWS. Para obtener más información sobre el inicio de sesión en AWS, consulta Cómo iniciar sesión en su Cuenta de AWS en la Guía del usuario de AWS Sign-In.

Si accede a AWS mediante programación, AWS proporciona un kit de desarrollo de software (SDK) y una interfaz de la línea de comandos (CLI) para firmar criptográficamente las solicitudes mediante el uso de las credenciales. Si no usa las herramientas de AWS, debe firmar las solicitudes. Para obtener más información sobre la firma de solicitudes, consulte AWSSignature Versión 4 para solicitudes API en la Guía del usuario de IAM.

Independientemente del método de autenticación que use, es posible que deba proporcionar información de seguridad adicional. Por ejemplo, AWS le recomienda el uso de la autenticación multifactor (MFA) para aumentar la seguridad de su cuenta. Para obtener más información, consulte Autenticación multifactor en la Guía del usuario de AWS IAM Identity Center y Autenticación multifactor AWSen IAM en la Guía del usuario de IAM.

Usuario raíz de Cuenta de AWS

Cuando se crea una Cuenta de AWS, se comienza con una identidad de inicio de sesión que tiene acceso completo a todos los recursos y Servicios de AWS de la cuenta. Esta identidad recibe el nombre de usuario raíz de la Cuenta de AWS y se accede a ella iniciando sesión con el email y la contraseña que utilizó para crear la cuenta. Recomendamos encarecidamente que no utiliza el usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de las tareas que requieren que inicie sesión como usuario raíz, consulta Tareas que requieren credenciales de usuario raíz en la Guía del usuario de IAM.

Identidad federada

Como práctica recomendada, solicite que los usuarios humanos, incluidos los que requieren acceso de administrador, utilizan la federación con un proveedor de identidades para acceder a los Servicios de AWS utilizando credenciales temporales.

Una identidad federada es un usuario del directorio de usuarios de su empresa, un proveedor de identidad web, el AWS Directory Service, el directorio del Identity Center, o cualquier usuario que acceda a Servicios de AWS utilizando credenciales proporcionadas a través de un origen de identidad. Cuando identidades federadas acceden a Cuentas de AWS, asumen roles y los roles proporcionan credenciales temporales.

Para una administración de acceso centralizada, le recomendamos que utiliza AWS IAM Identity Center. Puede crear usuarios y grupos en el IAM Identity Center o puede conectarse y sincronizar con un conjunto de usuarios y grupos de su propia fuente de identidad para usarlos en todas sus aplicaciones y Cuentas de AWS. Para obtener más información, consulta ¿Qué es el Centro de identidades de IAM? en la Guía del usuario de AWS IAM Identity Center.

Usuarios y grupos de IAM

Un usuario de IAM es una identidad de la Cuenta de AWSque dispone de permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos emplear credenciales temporales, en lugar de crear usuarios de IAM que tengan credenciales de larga duración como contraseñas y claves de acceso. No obstante, si tiene casos de uso específicos que requieran credenciales de larga duración con usuarios de IAM, recomendamos rotar las claves de acceso. Para más información, consulte Rotar las claves de acceso periódicamente para casos de uso que requieran credenciales de larga duración en la Guía del usuario de IAM.

Un grupo de IAM es una identidad que especifica un conjunto de usuarios de IAM. No puedes iniciar sesión como grupo. Puedes usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos de grandes conjuntos de usuarios. Por ejemplo, podría tener un grupo cuyo nombre fuese IAMAdmins y conceder permisos a dicho grupo para administrar los recursos de IAM.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales de larga duración permanentes; no obstante, los roles proporcionan credenciales temporales. Para obtener más información, consulte Casos de uso para usuarios de IAM en la Guía del usuario de IAM.

Roles de IAM

Un rol de IAM es una identidad de la Cuenta de AWS que dispone de permisos específicos. Es similar a un usuario de IAM, pero no está asociado a una persona determinada. Para asumir temporalmente un rol de IAM en la AWS Management Console, puede cambiar de una función de usuario a un rol de IAM (consola). Puedes asumir un rol llamando a una operación de la AWS CLI o de la API de AWS, o utilizando una URL personalizada. Para más información sobre los métodos para el uso de roles, consulta Métodos para asumir un rol en la Guía del usuario de IAM.

Los roles de IAM con credenciales temporales son útiles en las siguientes situaciones:

- Acceso de usuario federado: para asignar permisos a una identidad federada, puede crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información acerca de roles de federación, consulte Crear un rol para un proveedor de identidad de terceros (federación) en la Guía de usuario de IAM. Si utiliza el IAM Identity Center, debe configurar un conjunto de permisos. IAM Identity Center correlaciona el conjunto de permisos con un rol en IAM para controlar a qué pueden acceder las identidades después de autenticarse. Para obtener información acerca de los conjuntos de permisos, consulta Conjuntos de permisos en la Guía del usuario de AWS IAM Identity Center.
- Permisos de usuario de IAM temporales: un usuario de IAM puede asumir un rol de IAM para recibir temporalmente permisos distintos que le permitan realizar una tarea concreta.
- Acceso entre cuentas: puede utilizar un rol de IAM para permitir que alguien (una entidad principal de confianza) de otra cuenta acceda a los recursos de la cuenta. Los roles son la forma principal de conceder acceso entre cuentas. No obstante, con algunos Servicios de AWS se puede adjuntar una política directamente a un recurso (en lugar de utilizar un rol como representante). Para obtener información acerca de la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulta Acceso a recursos entre cuentas en IAM en la Guía del usuario de IAM.
- Acceso entre servicios: algunos Servicios de AWS utilizan características de otros Servicios de AWS. Por ejemplo, cuando realiza una llamada en un servicio, es común que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado a servicios.
  - Reenviar sesiones de acceso (FAS): cuando utiliza un rol o un usuario de IAM para llevar a cabo las acciones en AWS, se le considera una entidad principal. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. FAS utiliza los permisos de la entidad principal para llamar a un Servicio de AWS, combinados con el Servicio de AWS solicitante para realizar solicitudes a servicios posteriores. Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS o recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte Reenviar sesiones de acceso.
  - Rol de servicio: un rol de servicio es un rol de IAM que adopta un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio

desde IAM. Para obtener más información, consulte Creación de un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.

- Rol vinculado a los servicios: un rol vinculado a servicios es un tipo de rol de servicio que está vinculado a un Servicio de AWS. El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados a servicios aparecen en la Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.

- Aplicaciones que se ejecutan en Amazon EC2: puede utilizar un rol de IAM que le permita administrar credenciales temporales para las aplicaciones que se ejecutan en una instancia de EC2 y realizan solicitudes a la AWS CLI o a la API de AWS. Es preferible hacerlo de este modo a almacenar claves de acceso en la instancia de EC2. Para asignar un rol de AWS a una instancia de EC2 y ponerla a disposición de todas las aplicaciones, cree un perfil de instancia adjuntado a la instancia. Un perfil de instancia contiene el rol y permite a los programas que se ejecutan en la instancia de EC2 obtener credenciales temporales. Para obtener más información, consulte Uso de un rol de IAM para conceder permisos a aplicaciones que se ejecutan en instancias Amazon EC2 en la Guía del usuario de IAM.

Administración de acceso mediante políticas

Para controlar el acceso en AWS, se crean políticas y se adjuntan a identidades o recursos de AWS. Una política es un objeto de AWS que, cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando una entidad principal (sesión de rol, usuario o usuario raíz) realiza una solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega. La mayoría de las políticas se almacenan en AWS como documentos JSON. Para obtener más información sobre la estructura y el contenido de los documentos de política JSON, consulte Información general de políticas JSON en la Guía del usuario de IAM.

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Un administrador de IAM puede crear políticas de IAM para conceder permisos a los usuarios para realizar acciones en los recursos que necesitan. A continuación, el administrador puede añadir las políticas de IAM a roles y los usuarios pueden asumirlos.

Las políticas de IAM definen permisos para una acción independientemente del método que se utiliza para realizar la operación. Por ejemplo, suponga que dispone de una política que permite la acción

`iam:GetRole`. Un usuario con dicha política puede obtener información del usuario de la AWS Management Console, la AWS CLI o la API de AWS.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Las políticas basadas en identidades pueden clasificarse además como políticas insertadas o políticas administradas. Las políticas insertadas se integran directamente en un único usuario, grupo o rol. Las políticas administradas son políticas independientes que puedes adjuntar a varios usuarios, grupos y roles de su Cuenta de AWS. Las políticas gestionadas incluyen las políticas gestionadas de AWS y las políticas gestionadas por el cliente. Para obtener más información sobre cómo elegir una política administrada o una política insertada, consulte Elegir entre políticas administradas y políticas insertadas en la Guía del usuario de IAM.

Políticas basadas en recursos

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Los ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico. Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe especificar una entidad principal en una política en función de recursos. Las entidades principales pueden incluir cuentas, usuarios, roles, usuarios federados o Servicios de AWS.

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No se puede utilizar políticas de IAM administradas de AWS en una política basada en recursos.

Listas de control de acceso (ACL)

Las listas de control de acceso (ACL) controlan qué entidades principales (miembros de cuentas, usuarios o roles) tienen permisos para acceder a un recurso. Las ACL son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

Amazon S3, AWS WAF y Amazon VPC son ejemplos de servicios que admiten las ACL. Para obtener más información sobre las ACL, consulta Información general de Lista de control de acceso (ACL) en la Guía para desarrolladores de Amazon Simple Storage Service.

Otros tipos de políticas

AWS admite otros tipos de políticas adicionales menos frecuentes. Estos tipos de políticas pueden establecer el máximo de permisos que los tipos de políticas más frecuentes le conceden.

- Límites de permisos: un límite de permisos es una característica avanzada que le permite establecer los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM (usuario o rol de IAM). Puedes establecer un límite de permisos para una entidad. Los permisos resultantes son la intersección de las políticas basadas en la identidad de la entidad y los límites de permisos. Las políticas basadas en recursos que especifiquen el usuario o rol en el campo `Principal` no estarán restringidas por el límite de permisos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información sobre los límites de los permisos, consulta Límites de permisos para las entidades de IAM en la Guía del usuario de IAM.
- Políticas de control de servicio (SCP): las SCP son políticas de JSON que especifican los permisos máximos de una organización o una unidad organizativa en AWS Organizations. AWS Organizations es un servicio que le permite agrupar y administrar de manera centralizada varias Cuentas de AWS que posea su empresa. Si habilita todas las características en una empresa, entonces podrá aplicar políticas de control de servicio (SCP) a una o todas sus cuentas. Una SCP limita los permisos para las entidades de las cuentas de miembro, incluido cada Usuario raíz de la cuenta de AWS. Para obtener más información acerca de SCP y Organizations, consulta Políticas de control de servicios en la Guía del usuario de AWS Organizations.
- Políticas de control de recursos (RCP): las RCP son políticas JSON que permiten establecer los permisos máximos disponibles para los recursos de las cuentas sin actualizar las políticas de IAM asociadas a cada recurso que posea. La RCP limita los permisos de los recursos en las cuentas de miembros y puede afectar a los permisos efectivos de las identidades, incluidos los Usuario raíz de la cuenta de AWS, independientemente de si pertenecen a su organización. Para obtener más información sobre Organizations y RCP, incluida una lista de los Servicios de AWS que admiten RCP, consulte Políticas de control de recursos (RCP) en la Guía del usuario de AWS Organizations.
- Políticas de sesión: las políticas de sesión son políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal mediante programación para un rol o un usuario federado. Los permisos de la sesión resultantes son la intersección de las políticas basadas en identidades

del rol y las políticas de la sesión. Los permisos también pueden proceder de una política en función de recursos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para más información, consulte Políticas de sesión en la Guía del usuario de IAM.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para obtener información acerca de cómo AWS decide si permitir o no una solicitud cuando hay varios tipos de políticas implicados, consulte Lógica de evaluación de políticas en la Guía del usuario de IAM.

Cómo funciona Amazon Aurora DSQL con IAM

Antes de utilizar IAM para administrar el acceso a Aurora DSQL, obtenga más información sobre qué características de IAM se encuentran disponibles para utilizarlas con Aurora DSQL.

Características de IAM que puede utilizar con Amazon Aurora DSQL

Característica de IAM	Compatibilidad con Aurora DSQL
Políticas basadas en identidades	Sí
Políticas basadas en recursos	No
Acciones de políticas	Sí
Recursos de políticas	Sí
Claves de condición de política	Sí
ACL	No
ABAC (etiquetas en políticas)	Sí
Credenciales temporales	Sí
Permisos de entidades principales	Sí
Roles de servicio	Sí

Característica de IAM	Compatibilidad con Aurora DSQL
Roles vinculados al servicio	Sí

Para obtener información general sobre cómo funcionan Aurora DSQL y otros servicios de AWS con la mayoría de las características de IAM, consulte Servicios de AWS que funcionan con IAM en la Guía del usuario de IAM.

Políticas basadas en identidad para Aurora DSQL

Compatibilidad con las políticas basadas en identidad: sí

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Con las políticas basadas en identidades de IAM, puede especificar las acciones y los recursos permitidos o denegados, así como las condiciones en las que se permiten o deniegan las acciones. No es posible especificar la entidad principal en una política basada en identidad porque se aplica al usuario o rol al que está asociada. Para obtener más información sobre los elementos que puede utilizar en una política de JSON, consulte Referencia de los elementos de las políticas de JSON de IAM en la Guía del usuario de IAM.

Ejemplos de políticas basadas en identidad para Aurora DSQL

Para ver ejemplos de políticas basadas en identidad de Aurora DSQL, consulte Ejemplos de políticas basadas en identidad para Amazon Aurora DSQL.

Políticas basadas en recursos de Aurora DSQL

Admite políticas basadas en recursos: no

Las políticas basadas en recursos son documentos de política JSON que se asocian a un recurso. Los ejemplos de políticas basadas en recursos son las políticas de confianza de roles de IAM y las políticas de bucket de Amazon S3. En los servicios que admiten políticas basadas en recursos, los administradores de servicios pueden utilizarlos para controlar el acceso a un recurso específico.

Para el recurso al que se asocia la política, la política define qué acciones puede realizar una entidad principal especificada en ese recurso y en qué condiciones. Debe especificar una entidad principal en una política en función de recursos. Las entidades principales puedes incluir cuentas, usuarios, roles, usuarios federados o Servicios de AWS.

Para habilitar el acceso entre cuentas, puedes especificar toda una cuenta o entidades de IAM de otra cuenta como la entidad principal de una política en función de recursos. Añadir a una política en función de recursos una entidad principal entre cuentas es solo una parte del establecimiento de una relación de confianza. Cuando la entidad principal y el recurso se encuentran en Cuentas de AWS diferentes, un gestionador de IAM de la cuenta de confianza también debe conceder a la entidad principal (usuario o rol) permiso para acceder al recurso. Para conceder el permiso, adjunte la entidad a una política basada en identidad. Sin embargo, si la política basada en recursos concede acceso a una entidad principal de la misma cuenta, no es necesaria una política basada en identidad adicional. Para obtener más información, consulte Cross account resource access in IAM en la Guía del usuario de IAM.

Acciones de políticas para Aurora DSQL

Compatibilidad con las acciones de políticas: sí

Los administradores puedes utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Action` de una política JSON describe las acciones que puede utilizar para conceder o denegar el acceso en una política. Las acciones de la política generalmente tienen el mismo nombre que la operación de API de AWS asociada. Hay algunas excepciones, como acciones de solo permiso que no tienen una operación de API coincidente. También hay algunas operaciones que requieren varias acciones en una política. Estas acciones adicionales se denominan acciones dependientes.

Incluya acciones en una política para conceder permisos y así llevar a cabo la operación asociada.

Para ver una lista de las acciones de Aurora DSQL, consulte Acciones definidas por Amazon Aurora DSQL en la Referencia de autorizaciones de servicio.

Las acciones de políticas de Aurora DSQL utilizan el siguiente prefijo antes de la acción:

```
dsql
```

Para especificar varias acciones en una única instrucción, sepárelas con comas.

```
"Action": [
      "dsql:action1",
      "dsql:action2"
]
```

Para ver ejemplos de políticas basadas en identidad de Aurora DSQL, consulte Ejemplos de políticas basadas en identidad para Amazon Aurora DSQL.

Recursos de políticas para Aurora DSQL

Compatibilidad con los recursos de políticas: sí

Los administradores puedes utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puedes realizar acciones en qué recursos y en qué condiciones.

El elemento `Resource` de la política JSON especifica el objeto u objetos a los que se aplica la acción. Las instrucciones deben contener un elemento `Resource` o `NotResource`. Como práctica recomendada, especifique un recurso utilizando el Nombre de recurso de Amazon (ARN). Puedes hacerlo para acciones que admitan un tipo de recurso específico, conocido como permisos de nivel de recurso.

Para las acciones que no admiten permisos de nivel de recurso, como las operaciones de descripción, utiliza un carácter comodín (*) para indicar que la instrucción se aplica a todos los recursos.

```
"Resource": "*"
```

Para ver una lista de tipos de recursos de Aurora DSQL y los ARN, consulte Tipos de recurso definidos por Amazon Aurora DSQL en la Referencia de autorizaciones de servicio. Para obtener información acerca de las acciones con las que puede especificar el ARN de cada recurso, consulte Acciones definidas por Amazon Aurora DSQL.

Para ver ejemplos de políticas basadas en identidad de Aurora DSQL, consulte Ejemplos de políticas basadas en identidad para Amazon Aurora DSQL.

Claves de condición de políticas de Aurora DSQL

Compatibilidad con claves de condición de políticas específicas del servicio: sí

Los administradores puedes utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puedes realizar acciones en qué recursos y en qué condiciones.

El elemento `Condition` (o bloque de `Condition`) permite especificar condiciones en las que entra en vigor una instrucción. El elemento `Condition` es opcional. Puedes crear expresiones condicionales que utilizan operadores de condición, tales como igual o menor que, para que la condición de la política coincida con los valores de la solicitud.

Si especifica varios elementos de `Condition` en una instrucción o varias claves en un único elemento de `Condition`, AWS las evalúa mediante una operación `AND` lógica. Si especifica varios valores para una única clave de condición, AWS evalúa la condición con una operación lógica `OR`. Se deben cumplir todas las condiciones antes de que se concedan los permisos de la instrucción.

También puedes utilizar variables de marcador de posición al especificar condiciones. Por ejemplo, puedes conceder un permiso de usuario de IAM para acceder a un recurso solo si está etiquetado con su nombre de usuario de IAM. Para más información, consulta Elementos de la política de IAM: variables y etiquetas en la Guía del usuario de IAM.

AWS admite claves de condición globales y claves de condición específicas del servicio. Para ver todas las claves de condición globales de AWS, consulte Claves de contexto de condición globales de AWS en la Guía del usuario de IAM.

Para ver una lista de las claves de condición de Aurora DSQL, consulte Claves de condición para Amazon Aurora DSQL en la Referencia de autorizaciones de servicio. Para obtener información acerca de las acciones y los recursos con los que puede utilizar una clave de condición, consulte Acciones definidas por Amazon Aurora DSQL.

Para ver ejemplos de políticas basadas en identidad de Aurora DSQL, consulte Ejemplos de políticas basadas en identidad para Amazon Aurora DSQL.

ACL en Aurora DSQL

Compatibilidad con ACL: no

Las listas de control de acceso (ACL) controlan qué entidades principales (miembros de cuentas, usuarios o roles) tienen permisos para acceder a un recurso. Las ACL son similares a las políticas basadas en recursos, aunque no utilizan el formato de documento de políticas JSON.

ABAC con Aurora DSQL

Admite ABAC (etiquetas en las políticas): sí

El control de acceso basado en atributos (ABAC) es una estrategia de autorización que define permisos en función de atributos. En AWS, estos atributos se denominan etiquetas. Puedes adjuntar etiquetas a entidades de IAM (usuarios o roles) y a muchos recursos de AWS. El etiquetado de entidades y recursos es el primer paso de ABAC. A continuación, designa las políticas de ABAC para permitir operaciones cuando la etiqueta de la entidad principal coincida con la etiqueta del recurso al que se intenta acceder.

ABAC es útil en entornos que crecen con rapidez y ayuda en situaciones en las que la administración de las políticas resulta engorrosa.

Para controlar el acceso en función de etiquetas, debe proporcionar información de las etiquetas en el elemento de condición de una política utilizando las claves de condición `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` o `aws:TagKeys`.

Si un servicio admite las tres claves de condición para cada tipo de recurso, el valor es Sí para el servicio. Si un servicio admite las tres claves de condición solo para algunos tipos de recursos, el valor es Parcial.

Para obtener más información sobre ABAC, consulte Definición de permisos con la autorización de ABAC en la Guía del usuario de IAM. Para ver un tutorial con los pasos para configurar ABAC, consulte Uso del control de acceso basado en atributos (ABAC) en la Guía del usuario de IAM.

Uso de credenciales temporales con Aurora DSQL

Compatibilidad con credenciales temporales: sí

Algunos Servicios de AWS no funcionan cuando inicia sesión con credenciales temporales. Para obtener información adicional, incluida la información sobre qué Servicios de AWS funcionan con credenciales temporales, consulta Servicios de AWS que funcionan con IAM en la Guía del usuario de IAM.

Utiliza credenciales temporales si inicia sesión en la AWS Management Console con cualquier método, excepto un nombre de usuario y una contraseña. Por ejemplo, cuando accede a AWS utilizando el enlace de inicio de sesión único (SSO) de la empresa, ese proceso crea automáticamente credenciales temporales. También crea automáticamente credenciales temporales cuando inicia sesión en la consola como usuario y luego cambia de rol. Para obtener más

información sobre el cambio de roles, consulte Cambio de un usuario a un rol de IAM (consola) en la Guía del usuario de IAM.

Puedes crear credenciales temporales de forma manual mediante la AWS CLI o la API de AWS. A continuación, puedes usar esas credenciales temporales para acceder a AWS. AWS recomienda generar credenciales temporales de forma dinámica en lugar de usar claves de acceso a largo plazo. Para obtener más información, consulte Credenciales de seguridad temporales en IAM.

Permisos de entidades principales entre servicios de Aurora DSQL

Admite sesiones de acceso directo (FAS): sí

Cuando utiliza un usuario o un rol de IAM para llevar a cabo acciones en AWS, se le considera una entidad principal. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. FAS utiliza los permisos de la entidad principal para llamar a un Servicio de AWS, combinados con el Servicio de AWS solicitante para realizar solicitudes a servicios posteriores. Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS o recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte Reenviar sesiones de acceso.

Roles de servicio para Aurora DSQL

Compatibilidad con roles de servicio: sí

Un rol de servicio es un rol de IAM que asume un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para obtener más información, consulte Creación de un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.

> **Warning**
>
> Cambiar los permisos para un rol de servicio podría interrumpir la funcionalidad de Aurora DSQL. Edite los roles de servicio solo cuando Aurora DSQL proporcione orientación para hacerlo.

Roles vinculados al servicio para Aurora DSQL

Admite roles vinculados a servicios: sí

Un rol vinculado a servicios es un tipo de rol de servicio que está vinculado a un Servicio de AWS. El servicio puedes asumir el rol para realizar una acción en su nombre. Los roles vinculados a servicios aparecen en la Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.

Para obtener información acerca de cómo crear o administrar roles vinculados a servicios para Aurora SQL, consulte Uso de los roles vinculados al servicio en Aurora DSQL.

Ejemplos de políticas basadas en identidad para Amazon Aurora DSQL

De forma predeterminada, los usuarios y roles no tienen permiso para crear, ver ni modificar recursos de Aurora DSQL. Tampoco puedes realizar tareas mediante la AWS Management Console, la AWS Command Line Interface (AWS CLI) o la API de AWS. Un administrador de IAM puedes crear políticas de IAM para conceder permisos a los usuarios para realizar acciones en los recursos que necesitan. A continuación, el administrador puedes añadir las políticas de IAM a roles y los usuarios puedes asumirlos.

Para obtener información acerca de cómo crear una política basada en identidades de IAM mediante el uso de estos documentos de políticas JSON de ejemplo, consulte Creación de políticas de IAM (consola) en la Guía del usuario de IAM.

Para obtener detalles sobre las acciones y los tipos de recursos definidos por Aurora DSQL, incluido el formato de los ARN para cada uno de los tipos de recursos, consulte Acciones, recursos y claves de condición para Amazon Aurora DSQL en la Referencia de autorización de servicios.

Temas

- Prácticas recomendadas sobre las políticas
- Uso de la consola de Aurora DSQL
- Cómo permitir a los usuarios consultar sus propios permisos

Prácticas recomendadas sobre las políticas

Las políticas basadas en identidad determinan si alguien puede crear recursos de Aurora DSQL de la cuenta, acceder a ellos o eliminarlos. Estas acciones pueden generar costos adicionales para su Cuenta de AWS. Siga estas directrices y recomendaciones al crear o editar políticas basadas en identidades:

- Comienza con las políticas administradas por AWSy continúa con los permisos de privilegio mínimo: a fin de comenzar a conceder permisos a los usuarios y las cargas de tarea, utiliza las

políticas administradas por AWS, que conceden permisos para muchos casos de uso comunes. Están disponibles en su Cuenta de AWS. Se recomienda definir políticas gestionadas por el cliente de AWS específicas para sus casos de uso a fin de reducir aún más los permisos. Con el fin de obtener más información, consulta las políticas administradas por AWS o las políticas administradas por AWS para funciones de tarea en la Guía de usuario de IAM.

- Aplique permisos de privilegio mínimo: cuando establezca permisos con políticas de IAM, conceda solo los permisos necesarios para realizar una tarea. Para ello, debe definir las acciones que se pueden llevar a cabo en determinados recursos en condiciones específicas, también conocidos como permisos de privilegios mínimos. Con el fin de obtener más información sobre el uso de IAM para aplicar permisos, consulta Políticas y permisos en IAM en la Guía del usuario de IAM.
- Utilice condiciones en las políticas de IAM para restringir aún más el acceso: puede agregar una condición a sus políticas para limitar el acceso a las acciones y los recursos. Por ejemplo, puedes escribir una condición de políticas para especificar que todas las solicitudes deben enviarse utilizando SSL. También puedes usar condiciones para conceder acceso a acciones de servicios si se emplean a través de un Servicio de AWS determinado como, por ejemplo, AWS CloudFormation. Para obtener más información, consulta Elementos de la política de JSON de IAM: Condición en la Guía del usuario de IAM.
- Utiliza el analizador de acceso de IAM para validar las políticas de IAM con el fin de garantizar la seguridad y funcionalidad de los permisos: el analizador de acceso de IAM valida políticas nuevas y existentes para que respeten el lenguaje (JSON) de las políticas de IAM y las prácticas recomendadas de IAM. El analizador de acceso de IAM proporciona más de 100 verificaciones de políticas y recomendaciones procesables para ayudar a crear políticas seguras y funcionales. Para más información, consulte Validación de políticas con el Analizador de acceso de IAM en la Guía del usuario de IAM.
- Solicite la autenticación multifactor (MFA): si se encuentra en una situación en la que necesite usuarios raíz o de IAM en su Cuenta de AWS, active la MFA para obtener una mayor seguridad. Para exigir la MFA cuando se invoquen las operaciones de la API, añada condiciones de MFA a sus políticas. Para más información, consulte Acceso seguro a la API con MFA en la Guía del usuario de IAM.

Para obtener más información sobre las prácticas recomendadas de IAM, consulte Prácticas recomendadas de seguridad en IAM en la Guía del usuario de IAM.

Uso de la consola de Aurora DSQL

Para acceder a la consola de Amazon Aurora DSQL, debe tener un conjunto mínimo de permisos. Estos permisos deben permitirle mostrar y consultar los detalles sobre los recursos de Aurora DSQL en la Cuenta de AWS. Si crea una política basada en identidades que sea más restrictiva que el mínimo de permisos necesarios, la consola no funcionará del modo esperado para las entidades (usuarios o roles) que tengan esa política.

No es necesario que conceda permisos mínimos para la consola a los usuarios que solo realizan llamadas a la AWS CLI o a la API de AWS. En su lugar, permite el acceso únicamente a las acciones que coincidan con la operación de API que intentan realizar.

Para asegurarse de que los usuarios y los roles puedan seguir utilizando la consola de Aurora DSQL, asocie también a las entidades la política administrada de AWS `AmazonAuroraDSQLConsoleFullAccess` o `AmazonAuroraDSQLReadOnlyAccess` de Aurora DSQL. Para obtener más información, consulte Adición de permisos a un usuario en la Guía del usuario de IAM:

Cómo permitir a los usuarios consultar sus propios permisos

En este ejemplo, se muestra cómo podría crear una política que permita a los usuarios de IAM ver las políticas gestionadas e insertadas que se asocian a la identidad de sus usuarios. Esta política incluye permisos para llevar a cabo esta acción en la consola o mediante programación con la AWS CLI o la API de 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": "*"
        }
    ]
}
```

Solución de problemas de identidades y accesos en Amazon Aurora DSQL

Utilice la siguiente información para diagnosticar y solucionar los problemas comunes que puedan surgir cuando trabaje con Aurora DSQL e IAM.

Temas

- No tengo autorización para realizar una acción en Aurora DSQL
- No tengo autorización para realizar la operación iam:PassRole
- Quiero permitir a personas externas a mi Cuenta de AWS el acceso a mis recursos en Aurora DSQL

No tengo autorización para realizar una acción en Aurora DSQL

Si recibe un error que indica que no tiene autorización para realizar una acción, las políticas se deben actualizar para permitirle realizar la acción.

En el siguiente ejemplo, el error se produce cuando `mateojackson` intenta utilizar la consola para consultar los detalles acerca de un recurso *`my-dsql-cluster`*, pero no tiene los permisos *`GetCluster`*.

```
User: iam:::user/mateojackson is not authorized to perform: GetCluster on resource: my-dsql-cluster
```

En este caso, la política del usuario `mateojackson` debe actualizarse para permitir el acceso al recurso *`my-dsql-cluster`* mediante la acción *`GetCluster`*.

Si necesita ayuda, póngase en contacto con su gestionador de . El gestionador es la persona que le proporcionó las credenciales de inicio de sesión.

No tengo autorización para realizar la operación iam:PassRole

Si recibe un error que indica que no tiene autorización para realizar la acción `iam:PassRole`, se deben actualizar las políticas a fin de permitirle pasar un rol a Aurora DSQL.

Algunos Servicios de AWS le permiten transferir un rol existente a dicho servicio en lugar de crear un nuevo rol de servicio o uno vinculado al servicio. Para ello, debe tener permisos para transferir el rol al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado `marymajor` intenta utilizar la consola para realizar una acción en Aurora DSQL. Sin embargo, la acción requiere que el servicio cuente con permisos que otorguen un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su gestionador de AWS. El gestionador es la persona que le proporcionó las credenciales de inicio de sesión.

Quiero permitir a personas externas a mi Cuenta de AWS el acceso a mis recursos en Aurora DSQL

Puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Puede especificar una persona de confianza para que asuma el rol. En el caso de los servicios que admitan las políticas basadas en recursos o las listas de

control de acceso (ACL), puede utilizar dichas políticas para conceder a las personas acceso a sus recursos.

Para obtener más información, consulte lo siguiente:

- Para obtener información acerca de si Aurora DSQL admite estas características, consulte Cómo funciona Amazon Aurora DSQL con IAM.
- Para obtener información acerca de cómo proporcionar acceso a los recursos de las Cuentas de AWS de su propiedad, consulte Acceso para un usuario de IAM en otra Cuenta de AWS propia en la Guía del usuario de IAM.
- Para obtener información acerca de cómo proporcionar acceso a tus recursos a Cuentas de AWS de terceros, consulta Proporcionar acceso a Cuentas de AWS que son propiedad de terceros en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso mediante una federación de identidades, consulta Proporcionar acceso a usuarios autenticados externamente (identidad federada) en la Guía del usuario de IAM.
- Para conocer sobre la diferencia entre las políticas basadas en roles y en recursos para el acceso entre cuentas, consulte Acceso a recursos entre cuentas en IAM en la Guía del usuario de IAM.

Uso de los roles vinculados al servicio en Aurora DSQL

Aurora DSQL utiliza roles vinculados al servicio de AWS Identity and Access Management (IAM). Un rol vinculado a un servicio es un tipo único de rol de IAM que está vinculado directamente con Aurora DSQL. Los roles vinculados al servicio están predefinidos en Aurora DSQL e incluyen todos los permisos que el servicio requiere para llamar a Servicios de AWS en nombre del clúster de Aurora DSQL.

Los roles vinculados al servicio facilitan el proceso de configuración porque no tiene que agregar manualmente los permisos necesarios para utilizar Aurora DSQL. Cuando crea un clúster, Aurora DSQL crea automáticamente un rol vinculado al servicio para usted. Puede eliminar el rol vinculado al servicio solo después de eliminar todos los clústeres. De esta forma, se protegen los recursos de Aurora DSQL, ya que evita que se puedan eliminar accidentalmente los permisos necesarios para acceder a los recursos.

Para obtener información sobre otros servicios que admiten roles vinculados al servicio, consulte Servicios de AWS que funcionan con IAM y busque los servicios que muestran Sí en la columna

Rol vinculado al servicio. Elija una opción Sí con un enlace para ver la documentación acerca del rol vinculado a servicios en cuestión.

Los roles vinculados al servicio están disponibles en todas las regiones de Aurora DSQL admitidas.

Permisos de rol vinculado al servicio para Aurora DSQL

Aurora DSQL utiliza el rol vinculado al servicio denominado `AWSServiceRoleForAuroraDsql`. Permite a Amazon Aurora DSQL crear y administrar recursos de AWS en su nombre.
Este rol vinculado al servicio está asociado a la siguiente política administrada: AuroraDsqlServiceLinkedRolePolicy.

> **Note**
>
> Debe configurar permisos para permitir a una entidad de IAM (como un usuario, grupo o rol) crear, editar o eliminar un rol vinculado a servicios. Podría encontrarse con el siguiente mensaje de error: `You don't have the permissions to create an Amazon Aurora DSQL service-linked role`. Si aparece este mensaje, asegúrese de que tiene los siguientes permisos habilitados:
>
> ```
> {
> "Sid" : "CreateDsqlServiceLinkedRole",
>     "Effect" : "Allow",
>     "Action" : "iam:CreateServiceLinkedRole",
>     "Resource" : "*",
>     "Condition" : {
>       "StringEquals" : {
>         "iam:AWSServiceName" : "dsql.amazonaws.com"
>       }
>     }
>   }
> ```
>
> Para obtener más información, consulte Permisos de rol vinculado al servicio.

Creación de un rol vinculado al servicio

No necesita crear manualmente un rol vinculado al servicio AuroraDSQLServiceLinkedRolePolicy. Aurora DSQL crea el rol vinculado al servicio por usted. Si el rol vinculado al servicio

AuroraDSQLServiceLinkedRolePolicy se ha eliminado de la cuenta, Aurora DSQL crea el rol cuando usted crea un nuevo clúster de Aurora DSQL.

Edición de un rol vinculado a servicios

Aurora DSQL no le permite editar el rol vinculado al servicio AuroraDSQLServiceLinkedRolePolicy. Después de crear un rol vinculado a un servicio, no puede cambiarle el nombre, ya que varias entidades pueden hacer referencia a él. No obstante, puede editar la descripción del rol con la consola de IAM, la AWS Command Line Interface (AWS CLI) o la API de IAM.

Eliminar un rol vinculado a un servicio

Si ya no necesita usar una característica o servicio que requieran un rol vinculado a un servicio, le recomendamos que elimine dicho rol. De esta forma, no tiene una entidad no utilizada que no se supervise ni mantenga de forma activa.

Antes de poder eliminar un rol vinculado al servicio de una cuenta, debe eliminar cualquier clúster de la cuenta.

Puede utilizar la consola de IAM, la AWS CLI o la API de IAM para eliminar un rol vinculado a un servicio. Para obtener más información, consulte Creación de un rol vinculado al servicio en la Guía del usuario de IAM.

Regiones admitidas para los roles vinculados al servicio de Aurora DSQL

Aurora DSQL admite el uso de roles vinculados al servicio en todas las regiones en las que el servicio esté disponible. Para obtener más información, consulte Puntos de conexión y Regiones de AWS.

Uso de claves de condición de IAM con Amazon Aurora DSQL

Al conceder permisos en Aurora DSQL, puede especificar condiciones que determinan cómo se aplica una política de permisos. Los siguientes ejemplos muestran cómo puede usar claves de condición en las políticas de permisos de Aurora DSQL.

Ejemplo 1: concesión de permiso para crear un clúster en una Región de AWS específica

La siguiente política concede permiso para crear clústeres en las regiones Este de EE. UU. (Norte de Virginia) y Este de EE. UU. (Ohio). Esta política utiliza el recurso ARN para limitar las regiones

permitidas, por lo que Aurora DSQL solo puede crear clústeres si ese ARN está especificado en la sección `Resource` de la política.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            # Control where clusters can be created
            "Action": ["CreateCluster"],
            "Resource": [
                "arn:aws:dsql:us-east-1:*:cluster/*",
                "arn:aws:dsql:us-east-2:*:cluster/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Ejemplo 2: concesión de permiso para crear un clúster multirregional en una Región de AWS específica o en varias

La siguiente política concede permiso para crear clústeres multirregionales en las regiones Este de EE. UU. (Norte de Virginia) y Este de EE. UU. (Ohio). Esta política utiliza el ARN de recurso para limitar las regiones permitidas, por lo que Aurora DSQL solo puede crear clústeres de varias regiones si ese ARN está especificado en la sección `Resource` de la política. Tenga en cuenta que la creación de clústeres de varias regiones también requiere los permisos `PutMultiRegionProperties`, `PutWitnessRegion` y `AddPeerCluster` en cada región especificada.

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/*"
          ]
        }
    ]
}
```

Ejemplo 3: concesión de permiso para crear un clúster multirregional con una región testigo específica

La siguiente política utiliza una clave de condición `dsql:WitnessRegion` de Aurora DSQL y permite a un usuario crear clústeres multirregionales con una región testigo en Oeste de EE. UU. (Oregón). Si no especifica la condición `dsql:WitnessRegion`, puede usar cualquier región como testigo.

```
{
    "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"
                     ]
```

```
                }

              }
            }
        ]
}
```

Respuesta a incidentes en Amazon Aurora DSQL

La seguridad de AWS es nuestra mayor prioridad. Como parte del modelo de responsabilidad compartida de la nube de AWS, AWS administra una arquitectura de red, software y centro de datos que cumple los requisitos de seguridad de las organizaciones más exigentes. AWS se encarga de responder a cualquier incidente relacionado con el propio servicio de Amazon Aurora DSQL. Como cliente de AWS, usted comparte la responsabilidad de mantener la seguridad en la nube. Esto significa que usted controla la seguridad que decide implementar desde las herramientas y características de AWS a las que tiene acceso. Además, usted es responsable de la respuesta a los incidentes en su parte del modelo de responsabilidad compartida.

Al establecer una base de seguridad que cumpla con los objetivos de las aplicaciones que se ejecutan en la nube, puede detectar las desviaciones a las que puede responder. Para ayudarle a entender el impacto que la respuesta a los incidentes y sus decisiones tienen en sus objetivos empresariales, le recomendamos que consulte los siguientes recursos:

- AWS Security Incident Response Guide
- Prácticas recomendadas para la seguridad, la identidad y el cumplimiento de AWS
- Documento técnico Perspectiva de seguridad de AWS Cloud Adoption Framework (CAF)

Amazon GuardDuty es un servicio administrado de detección de amenazas que supervisa de forma continua los comportamientos malintencionados o no autorizados para ayudar a los clientes a proteger las cargas de trabajo y las Cuentas de AWS e identificar posibles actividades sospechosas antes de que se conviertan en un incidente. Supervisa actividades como las llamadas inusuales a la API o las implementaciones potencialmente no autorizadas, lo que indica la posibilidad de que las cuentas o los recursos se vean comprometidos o el reconocimiento por parte de personas malintencionadas. Por ejemplo, Amazon GuardDuty es capaz de detectar actividades sospechosas en las API de Amazon Aurora DSQL, como el inicio de sesión de un usuario desde una nueva ubicación y la creación de un nuevo clúster.

Validación del cumplimiento para Amazon Aurora DSQL

Para saber si un Servicio de AWS está incluido en el ámbito de programas de conformidad específicos, consulte Servicios de AWS en el ámbito del programa de conformidad y elija el programa de conformidad que le interese. Para obtener información general, consulta Programas de conformidad de AWS.

Puedes descargar los informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulta Descarga de informes en AWS Artifact.

Su responsabilidad de conformidad al utilizar Servicios de AWS se determina en función de la sensibilidad de los datos, los objetivos de cumplimiento de su empresa y la legislación y los reglamentos correspondientes. AWS proporciona los siguientes recursos para ayudar con la conformidad:

- Cumplimiento de seguridad y gobernanza: en estas guías se explican las consideraciones de arquitectura y se proporcionan pasos para implementar las características de seguridad y cumplimiento.
- Referencia de servicios válidos de HIPAA: muestra una lista con los servicios válidos de HIPAA. No todos los Servicios de AWS son aptos para HIPAA.
- Recursos de conformidad de AWS: este conjunto de manuales y guías podría aplicarse a su sector y ubicación.
- Guías de cumplimiento para clientes de AWS: comprenda el modelo de responsabilidad compartida desde el punto de vista del cumplimiento. Las guías resumen las mejores prácticas para garantizar la seguridad de los Servicios de AWS y orientan los controles de seguridad en varios marcos (incluidos el Instituto Nacional de Estándares y Tecnología (NIST, por sus siglas en inglés), el Consejo de Estándares de Seguridad de la Industria de Tarjetas de Pago (PCI, por sus siglas en inglés) y la Organización Internacional de Normalización (ISO, por sus siglas en inglés)).
- Evaluación de recursos con reglas en la Guía para desarrolladores de AWS Config: el servicio AWS Config evalúa en qué medida las configuraciones de sus recursos cumplen las prácticas internas, las directrices del sector y las normativas.
- AWS Security Hub: este Servicio de AWS proporciona una visión completa de su estado de seguridad en AWS. Security Hub utiliza controles de seguridad para evaluar sus recursos de AWS y comprobar su cumplimiento con los estándares y las prácticas recomendadas del sector de la seguridad. Para obtener una lista de los servicios y controles compatibles, consulta la Referencia de controles de Security Hub.

- Amazon GuardDuty: este Servicio de AWS detecta posibles amenazas para sus Cuentas de AWS, cargas de tarea, contenedores y datos mediante la supervisión de su entorno para detectar actividades sospechosas y maliciosas. GuardDuty puedes ayudarlo a satisfacer varios requisitos de conformidad, como PCI DSS, cumpliendo los requisitos de detección de intrusos que exigen determinados marcos de conformidad.
- AWS Audit Manager: este servicio de Servicio de AWS le ayuda a auditar continuamente el uso de AWS con el fin de simplificar la forma en que administra el riesgo y la conformidad con las normativas y los estándares del sector.

Resiliencia en Amazon Aurora DSQL

La infraestructura global de AWS se divide en Regiones de AWS y zonas de disponibilidad (AZ). Las Regiones de AWS proporcionan varias zonas de disponibilidad físicamente independientes y aisladas que se encuentran conectadas mediante redes con un alto nivel de rendimiento y redundancia, además de baja latencia. Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre las zonas sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de uno o varios centros de datos. Aurora DSQL se ha diseñado para que pueda aprovechar la infraestructura regional de AWS y, al mismo tiempo, proporcionar la máxima disponibilidad de base de datos. De forma predeterminada, los clústeres de una sola región en Aurora DSQL tienen disponibilidad Multi-AZ, lo que proporciona tolerancia a los principales errores de los componentes y a las interrupciones de la infraestructura que podrían afectar el acceso a una AZ completa. Los clústeres multirregionales proporcionan todos los beneficios de la resiliencia Multi-AZ a la vez que siguen proporcionando la disponibilidad de base de datos de alta coherencia, incluso en los casos en los que Región de AWS es inaccesible para los clientes de la aplicación.

Para obtener más información sobre las Regiones de AWS y las zonas de disponibilidad, consulte Infraestructura global de AWS.

Además de la infraestructura global de AWS, Aurora DSQL ofrece varias características que lo ayudan con las necesidades de resiliencia y copia de seguridad de los datos.

Copia de seguridad y restauración

Aurora DSQL admite copias de seguridad y restauración con Consola de AWS Backup. Puede realizar una copia de seguridad completa y restaurar los clústeres de una sola región y de varias

regiones. Para obtener más información, consulte Copia de seguridad y restauración para Amazon Aurora DSQL.

Replicación

Por diseño, Aurora DSQL confirma todas las transacciones de escritura en un registro de transacciones distribuido y replica de forma síncrona todos los datos de registro confirmados en réplicas de almacenamiento de usuario en tres AZ. Los clústeres multirregionales proporcionan capacidades completas de replicación entre regiones de lectura y escritura.

Una región testigo designada admite escrituras solo de registro de transacciones y no consume almacenamiento. Las regiones testigo no tienen punto de conexión. Esto significa que las regiones testigo solo almacenan registros de transacciones cifrados, no requieren administración ni configuración y no son accesibles para los usuarios.

Los registros de transacciones de Aurora DSQL y el almacenamiento del usuario se distribuyen con todos los datos presentados a los procesadores de consultas de Aurora DSQL como un único volumen lógico. Aurora DSQL divide, combina y replica automáticamente los datos basándose en el intervalo de clave principal de la base de datos y en los patrones de acceso. Aurora DSQL escala y reduce verticalmente las réplicas de lectura de forma automática basándose en la frecuencia de acceso de lectura.

Las réplicas de almacenamiento de clúster se distribuyen a través de una flota de almacenamiento de varios inquilinos. Si un componente o AZ se deteriora, Aurora DSQL redirige automáticamente el acceso a los componentes supervivientes y repara de forma asíncrona las réplicas que faltan. Una vez que Aurora DSQL repara las réplicas deterioradas, las vuelve a agregar automáticamente al quórum de almacenamiento y las pone a disposición del clúster.

Alta disponibilidad

De forma predeterminada, los clústeres de una sola región y multirregionales en Aurora DSQL son activo-activo, y no necesita aprovisionar, configurar ni reconfigurar manualmente ningún clúster. Aurora DSQL automatiza completamente la recuperación del clúster, lo que elimina la necesidad de las tradicionales operaciones de conmutación por error principal-secundario. La replicación es siempre síncrona y se realiza en múltiples AZ, por lo que no hay riesgo de pérdida de datos debido al retardo en la replicación o a la conmutación por error a una base de datos secundaria asíncrona durante la recuperación por error.

Los clústeres de una sola región proporcionan un punto de conexión redundante Multi-AZ que habilita automáticamente el acceso simultáneo con una gran coherencia de datos en tres AZ.

Esto significa que las réplicas de almacenamiento de usuario en cualquiera de estas tres AZ siempre devuelven el mismo resultado a uno o más lectores y siempre están disponibles para recibir escrituras. Esta sólida coherencia y resiliencia Multi-AZ está disponible en todas las regiones para los clústeres multirregionales de Aurora DSQL. Esto significa que los clústeres multirregionales proporcionan dos puntos de conexión regionales de alta coherencia, por lo que los clientes pueden leer o escribir indistintamente en cualquiera de las regiones con un retardo de replicación cero en la confirmación.

Aurora DSQL proporciona una disponibilidad del 99,99 % para clústeres de una sola región y del 99,999 % para clústeres multirregionales.

Seguridad de infraestructuras en Amazon Aurora DSQL

Como servicio administrado, Amazon Aurora DSQL está protegido por los procedimientos de seguridad de red globales de AWS que se describen en Prácticas recomendadas para seguridad, identidad y conformidad.

Utilice las llamadas a la API publicadas por AWS para acceder a Aurora DSQL a través de la red. Los clientes deben ser compatibles con Transport Layer Security (TLS) 1.2 o una versión posterior. Los clientes también deben ser compatibles con conjuntos de cifrado con confidencialidad directa total (PFS) tales como DHE (Ephemeral Diffie-Hellman) o ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad principal de IAM. También puedes utilizar AWS Security Token Service (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

Administración y conexión a clústeres de Amazon Aurora DSQL mediante AWS PrivateLink

Con AWS PrivateLink para Amazon Aurora DSQL, puede aprovisionar puntos de conexión de Amazon VPC de la interfaz (puntos de conexión de interfaz) en Amazon Virtual Private Cloud. A estos puntos de conexión se puede acceder directamente desde las aplicaciones que se encuentran en las instalaciones a través de Amazon VPC y AWS Direct Connect, o bien, en una Región de AWS diferente mediante el emparejamiento de Amazon VPC. Al usar AWS PrivateLink y puntos de

conexión de interfaz, puede simplificar la conectividad de la red privada desde las aplicaciones a Aurora DSQL.

Las aplicaciones en la Amazon VPC pueden acceder a Aurora DSQL mediante los puntos de conexión de interfaz de Amazon VPC sin necesidad de direcciones IP públicas.

Los puntos de conexión de la interfaz se representan mediante una o más interfaces de red elásticas (ENI) a las que se asignan direcciones IP privadas desde subredes de la Amazon VPC. Las solicitudes a Aurora DSQL a través de puntos de conexión de interfaz permanecen en la red de AWS. Para obtener más información sobre cómo conectar Amazon VPC a la red en las instalaciones, consulte la Guía del usuario de AWS Direct Connect y la Guía del usuario de AWS Site-to-Site VPN.

Para obtener información general sobre los puntos de conexión de interfaz, consulte Acceso a un servicio de AWS mediante un punto de conexión de Amazon VPC en la Guía del usuario de AWS PrivateLink.

Tipos de puntos de conexión de Amazon VPC para Aurora DSQL

Aurora DSQL requiere dos tipos diferentes de puntos de conexión de AWS PrivateLink.

1. Punto de conexión de administración: este punto de conexión se utiliza para operaciones de administración, como `get`, `create`, `update`, `delete` y `list` en clústeres de Aurora DSQL. Consulte Administración de clústeres de Aurora DSQL mediante AWS PrivateLink.
2. Punto de conexión: este punto de conexión se utiliza para conectarse a los clústeres de Aurora DSQL a través de clientes de PostgreSQL. Consulte Conexión a clústeres de Aurora DSQL mediante AWS PrivateLink.

Consideraciones al utilizar AWS PrivateLink para Aurora DSQL

Las consideraciones de Amazon VPC se aplican a AWS PrivateLink para Aurora DSQL. Para obtener más información, consulte Acceso a un servicio de AWS con un punto de conexión de VPC de interfaz y Cuotas de AWS PrivateLink en la Guía de AWS PrivateLink.

Administración de clústeres de Aurora DSQL mediante AWS PrivateLink

Puede utilizar la AWS Command Line Interface o los kits de desarrollo de software (SDK) de AWS para administrar clústeres de Aurora DSQL a través de puntos de conexión de interfaz de Aurora DSQL.

Creación de un punto de conexión de VPC de Amazon

Para crear un punto de conexión de interfaz de Amazon VPC, consulte Creación de un punto de conexión de Amazon VPC en la Guía de 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 \
```

Para utilizar el nombre DNS regional predeterminado para las solicitudes de la API de Aurora DSQL, no desactive el DNS privado cuando cree el punto de conexión de interfaz de Aurora DSQL. Cuando el DNS privado esté habilitado, las solicitudes al servicio Aurora DSQL realizadas desde dentro de Amazon VPC se resolverán automáticamente en la dirección IP privada del punto de conexión de VPC de Amazon, en lugar del nombre DNS público. Si el DNS privado está habilitado, las solicitudes de Aurora DSQL realizadas dentro de Amazon VPC se resolverán automáticamente en el punto de conexión de VPC de Amazon.

Si el DNS privado no está habilitado, utilice los parámetros `--region` y `--endpoint-url` con comandos de la AWS CLI para administrar los clústeres de Aurora DSQL a través de los puntos de conexión de interfaz de Aurora DSQL.

Enumeración de clústeres mediante una URL de punto de conexión

En el siguiente ejemplo, reemplace la Región de AWS `us-east-1` y el nombre DNS del ID de punto de conexión de Amazon VPC `vpce-1a2b3c4d-5e6f.dynamodb.us-east-1.vpce.amazonaws.com` con información propia.

```
aws dsql --region us-east-1 --endpoint-url https://vpce-1a2b3c4d-5e6f.dsql.us-east-1.vpce.amazonaws.com list-clusters
```

Operaciones de API

Consulte la referencia de la API de Aurora DSQL para obtener documentación sobre la administración de recursos en Aurora DSQL.

Administración de políticas de punto de conexión

Al probar y configurar detenidamente las políticas de punto de conexión de Amazon VPC, puede ayudar a garantizar que el clúster de Aurora DSQL sea seguro, cumpla las normas y se ajuste a los requisitos específicos de control de acceso y gobernanza de la organización.

Ejemplo: política de acceso completo a Aurora DSQL

La siguiente política concede acceso completo a todas las acciones y recursos de Aurora DSQL a través del punto de conexión de Amazon VPC especificado.

```
aws ec2 modify-vpc-endpoint \
    --vpc-endpoint-id vpce-xxxxxxxxxxxxxxxx \
    --region region \
    --policy-document '{
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": "*",
          "Action": "dsql:*",
          "Resource": "*"
        }
      ]
    }'
```

Ejemplo: política de acceso restringido a Aurora DSQL

La siguiente política solo permite estas acciones de Aurora DSQL.

- `CreateCluster`
- `GetCluster`
- `ListClusters`

Se deniegan todas las demás acciones de Aurora DSQL.

JSON

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "dsql:CreateCluster",
        "dsql:GetCluster",
        "dsql:ListClusters"
      ],
      "Resource": "*"
    }
  ]
}
```

Conexión a clústeres de Aurora DSQL mediante AWS PrivateLink

Una vez que el punto de conexión de AWS PrivateLink esté configurado y activo, podrá conectarse al clúster de Aurora DSQL mediante un cliente de PostgreSQL. Las instrucciones de conexión que aparecen a continuación describen los pasos para construir el nombre de host adecuado para conectarse a través del punto de conexión de AWS PrivateLink.

Configuración de un punto de conexión de AWS PrivateLink

Paso 1: obtención del nombre del servicio del clúster

Cuando cree un punto de conexión de AWS PrivateLink para conectarse al clúster, primero deberá obtener el nombre de servicio específico del clúster.

AWS CLI

```
aws dsql get-vpc-endpoint-service-name \
--region us-east-1 \
--identifier your-cluster-id
```

Ejemplo de respuesta

```
{
    "serviceName": "com.amazonaws.us-east-1.dsql-fnh4"
}
```

El nombre del servicio incluye un identificador, como `dsql-fnh4` en el ejemplo. Este identificador también es necesario al construir el nombre de host para conectarse al clúster.

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);
```

Paso 2: creación del punto de conexión de Amazon VPC

Mediante el nombre de servicio obtenido en el paso anterior, cree un punto de conexión de Amazon VPC.

> **Important**
>
> Las instrucciones de conexión que aparecen a continuación solo funcionan para conectarse a clústeres cuando está habilitado el DNS privado. No utilice la marca `--no-private-dns-enabled` al crear el punto de conexión, ya que esto impedirá que las instrucciones de conexión que figuran a continuación funcionen correctamente. Si desactiva el DNS privado, tendrá que crear un registro DNS privado comodín propio que apunte al punto de conexión creado.

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
```

Ejemplo de respuesta

```
{
    "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

Uso de una URL de punto de conexión para las API de Aurora DSQL

```
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);
```

Conexión a un clúster de Aurora DSQL mediante un punto de conexión de AWS PrivateLink

Una vez que el punto de conexión de AWS PrivateLink esté configurado y activo (compruebe que `State` es `available`), puede conectarse al clúster de Aurora DSQL mediante un cliente de PostgreSQL. Para obtener instrucciones sobre cómo utilizar los AWS SDK, puede seguir las guías de Programación con Aurora DSQL. Debe cambiar el punto de conexión del clúster para que coincida con el formato del nombre de host.

Construcción del nombre de host

El nombre de host para conectarse a través de AWS PrivateLink difiere del nombre de host DNS público. Debe construirlo con los siguientes componentes.

1. `Your-cluster-id`
2. El identificador del servicio a partir del nombre del servicio. Por ejemplo: `dsql-fnh4`

3. la Región de AWS,

Use el siguiente formato *cluster-id*.*service-identifier*.*region*.on.aws

Ejemplo: conexión mediante 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
```

Solución de problemas con AWS PrivateLink

Problemas y soluciones comunes

En la siguiente tabla se muestran los problemas y las soluciones más comunes relacionados con AWS PrivateLink con Aurora DSQL.

Problema	Causa posible	Solución
Tiempo de espera de la conexión	El grupo de seguridad no está configurado correctamente	Utilice Analizador de accesibilidad de Amazon VPC para asegurarse de que la configuración de red permite el tráfico en el puerto 5432.
Error de resolución de DNS	DNS privado no habilitado	Compruebe que el punto de conexión de VPC de Amazon se ha creado con el DNS privado habilitado.
Error de autenticación	Credenciales incorrectas o token caducado	Genere un nuevo token de autenticación y verifique el nombre de usuario.

Problema	Causa posible	Solución
No se ha encontrado el nombre de servicio	ID de clúster incorrecto	Compruebe el ID del clúster y Región de AWS al obtener el nombre del servicio.

Activos relacionados

Para obtener más información, consulte los siguientes recursos:

- Guía del usuario de Amazon Aurora DSQL
- Documentación de AWS PrivateLink
- Acceso a los servicios de AWS a través de AWS PrivateLink

Configuración y análisis de vulnerabilidades en Amazon Aurora DSQL

AWS gestiona las tareas de seguridad básicas, como la aplicación de parches en la base de datos y el sistema operativo (SO) de invitado, la configuración del firewall y la recuperación de desastres. Estos procedimientos han sido revisados y certificados por los terceros pertinentes. Para obtener más detalles, consulte los siguientes recursos de :

- Modelo de responsabilidad compartida
- Amazon Web Services: Información general de procesos de seguridad (documento técnico)

Prevención de la sustitución confusa entre servicios

El problema de la sustitución confusa es un problema de seguridad en el que una entidad que no tiene permiso para realizar una acción puede obligar a una entidad con más privilegios a realizar la acción. En AWS, la suplantación entre servicios puede dar lugar al problema de la sustitución confusa. La suplantación entre servicios puedes producirse cuando un servicio (el servicio que lleva a cabo las llamadas) llama a otro servicio (el servicio al que se llama). El servicio que lleva a cabo las llamadas se puedes manipular para utilizar sus permisos a fin de actuar en función de los recursos de otro cliente de una manera en la que no debe tener permiso para acceder. Para evitarlo, AWS proporciona herramientas que lo ayudan a proteger sus datos para todos los servicios con entidades principales de servicio a las que se les ha dado acceso a los recursos de su cuenta.

Se recomienda utilizar las claves de contexto de condición global `aws:SourceArn` y `aws:SourceAccount` en las políticas de recursos para limitar los permisos que Amazon Aurora DSQL concede a otro servicio para el recurso. Utiliza `aws:SourceArn` si desea que solo se asocie un recurso al acceso entre servicios. Utiliza `aws:SourceAccount` si quiere permitir que cualquier recurso de esa cuenta se asocie al uso entre servicios.

La forma más eficaz de protegerse contra el problema de la sustitución confusa es utilizar la clave de contexto de condición global de `aws:SourceArn` con el ARN completo del recurso. Si no conoce el ARN completo del recurso o si está especificando varios recursos, utilice la clave de condición de contexto global `aws:SourceArn` con caracteres comodines (*) para las partes desconocidas del ARN. Por ejemplo, `arn:aws:servicename:*:123456789012:*`.

Si el valor de `aws:SourceArn` no contiene el ID de cuenta, como un ARN de bucket de Amazon S3, debe utilizar ambas claves de contexto de condición global para limitar los permisos.

El valor de `aws:SourceArn` debe ser ResourceDescription.

El siguiente ejemplo muestra cómo se pueden utilizar las claves de contexto de condición global `aws:SourceArn` y `aws:SourceAccount` en Aurora DSQL para prevenir el error de la sustitución confusa.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "ConfusedDeputyPreventionExamplePolicy",
    "Effect": "Allow",
    "Principal": {
      "Service": "servicename.amazonaws.com"
    },
    "Action": "servicename:ActionName",
    "Resource": [
      "arn:aws:servicename:::ResourceName/*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:servicename:*:123456789012:*"
      },
      "StringEquals": {
        "aws:SourceAccount": "123456789012"
```

```
        }
      }
    }
  }
```

Prácticas recomendadas de seguridad para Aurora DSQL

Aurora DSQL proporciona una serie de características de seguridad que debe tener en cuenta a la hora de desarrollar e implementar las políticas de seguridad propias. Las siguientes prácticas recomendadas son directrices generales y no constituyen una solución de seguridad completa. Puesto que es posible que estas prácticas recomendadas no sean adecuadas o suficientes para el entorno, considérelas como consideraciones útiles en lugar de como normas.

Temas

Prácticas recomendadas de detección de seguridad para Aurora DSQL

Además de las siguientes formas de utilizar Aurora DSQL de forma segura, consulte Seguridad en AWS Well-Architected Tool para obtener información sobre cómo las tecnologías en la nube mejoran la seguridad.

Alarmas de Amazon CloudWatch

Con las alarmas de Amazon CloudWatch, puede ver una métrica determinada durante el periodo especificado. Si la métrica supera un límite determinado, se envía una notificación a un tema de Amazon SNS o a una política de AWS Auto Scaling. Las alarmas de CloudWatch no invocan acciones simplemente porque se encuentren en determinado estado. En su lugar, el estado debe haber cambiado y debe mantenerse durante el número de periodos especificado.

Etiquetado de los recursos de Aurora DSQL para la identificación y la automatización

Puede asignar metadatos a los recursos de AWS en forma de etiquetas. Cada etiqueta es una marca que consta de una clave definida por el cliente y un valor opcional que puede hacer que sea más fácil administrar, buscar y filtrar recursos.

El etiquetado permite implementar controles agrupados. Aunque no hay tipos inherentes de etiquetas, lo habilitan a clasificar los recursos según su finalidad, propietario, entorno u otros criterios. A continuación se muestran algunos ejemplos:

- Seguridad: se utiliza para determinar requisitos tales como el cifrado.
- Confidencialidad: un identificador para el nivel concreto de confidencialidad de los datos que admite un recurso.
- Entorno: utilizado para distinguir entre el desarrollo, la prueba y la infraestructura de producción.

Puede asignar metadatos a los recursos de AWS en forma de etiquetas. Cada etiqueta es una marca que consta de una clave definida por el cliente y un valor opcional que puede hacer que sea más fácil administrar, buscar y filtrar recursos.

El etiquetado permite implementar controles agrupados. Aunque no hay tipos inherentes de etiquetas, le permiten clasificar recursos de según su finalidad, propietario, entorno u otro criterio. A continuación se muestran algunos ejemplos.

- Seguridad: se utiliza para determinar requisitos tales como el cifrado.
- Confidencialidad: un identificador para el nivel concreto de confidencialidad de los datos que admite un recurso.
- Entorno: utilizado para distinguir entre el desarrollo, la prueba y la infraestructura de producción.

Para obtener información, consulte Prácticas recomendadas para el etiquetado de los recursos de AWS.

Prácticas recomendadas de seguridad preventiva para Aurora DSQL

Además de las siguientes formas de utilizar Aurora DSQL de forma segura, consulte Seguridad en AWS Well-Architected Tool para obtener información sobre cómo las tecnologías en la nube mejoran la seguridad.

Use roles de IAM para autenticar el acceso a Aurora DSQL.

Los usuarios, las aplicaciones y demás Servicios de AWS que accedan a Aurora DSQL deben incluir credenciales de AWS válidas en la API de AWS y en las solicitudes de la AWS CLI. No debe almacenar las credenciales de AWS de forma directa en la aplicación ni en las instancias EC2. Se trata de credenciales a largo plazo que no se rotan automáticamente. Existe un impacto

empresarial significativo si estas credenciales se ven comprometidas. Un rol de IAM le permite obtener claves de acceso temporal que puede utilizar para acceder a los recursos y los Servicios de AWS.

Para obtener más información, consulte Autenticación y autorización para Aurora DSQL.

Use políticas de IAM para la autorización de la base de Aurora DSQL.

Cuando concede permisos, usted decide quién los obtiene, para qué operaciones de la API de Aurora DSQL obtiene permisos y las acciones específicas que desea permitir en esos recursos. La implementación de privilegios mínimos es la clave a la hora de reducir los riesgos de seguridad y el impacto que podrían causar los errores o los intentos malintencionados.

Asocie políticas de permisos a los roles de IAM y conceda permisos para realizar operaciones en los recursos de Aurora DSQL. También están disponibles los límites de permisos para entidades de IAM, que le permiten establecer los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM.

De forma similar a las prácticas recomendadas para el usuario raíz de la Cuenta de AWS, no utilice el rol de `admin` en Aurora DSQL para realizar operaciones cotidianas. En su lugar, le recomendamos que cree roles de base de datos personalizados para administrar y conectarse al clúster. Para obtener más información, consulte Acceso a Aurora DSQL y Descripción de la autenticación y autorización para Aurora DSQL.

Use **`verify-full`** en entornos de producción.

Esta configuración comprueba que el certificado del servidor esté firmado por una autoridad de certificación de confianza y que el nombre de host del servidor coincida con el certificado.

Actualización del cliente de PostgreSQL

Actualice periódicamente el cliente de PostgreSQL a la versión más reciente para beneficiarse de las mejoras de seguridad. Recomendamos utilizar la versión 17 de PostgreSQL.

Etiquetado de recursos en Aurora DSQL

En AWS, las etiquetas son pares clave-valor definidos por el usuario que usted define y asocia a recursos de Aurora DSQL como los clústeres. Las etiquetas son opcionales. Si proporciona una clave, el valor es opcional.

Puede utilizar la AWS Management Console, la AWS CLI o los AWS SDK para agregar, enumerar y eliminar etiquetas en los clústeres de Aurora DSQL. Puede agregar etiquetas durante y después de la creación del clúster mediante la consola de AWS. Para etiquetar un clúster después de la creación con la AWS CLI, utilice la operación `TagResource`.

Etiquetado de clústeres con un nombre

Aurora DSQL crea clústeres con un identificador único global asignado como nombre de recurso de Amazon (ARN). Si desea asignar un nombre descriptivo al clúster, le recomendamos que utilice una etiqueta.

Si crea un clúster con la consola de Aurora DSQL, Aurora DSQL crea automáticamente una etiqueta. Esta etiqueta tiene una clave de Nombre y un valor generado automáticamente que representa el nombre del clúster. Este valor es configurable, por lo que puede asignar un nombre más descriptivo al clúster. Si un clúster tiene una etiqueta Nombre con un valor asociado, podrá ver el valor en la consola de Aurora DSQL.

Requisitos de etiquetado

Las etiquetas tienen los siguientes requisitos:

- Las claves no pueden tener el prefijo `aws:`.
- Las claves deben ser únicas dentro de un conjunto de etiquetas.
- Una clave debe tener entre 1 y 128 caracteres permitidos.
- Un valor debe tener entre 0 y 256 caracteres permitidos.
- No es necesario que los valores sean únicos dentro de un conjunto de etiquetas.
- Los caracteres permitidos para las claves y los valores son letras, dígitos, espacios en blanco y cualquiera de los siguientes símbolos: _ . : / = + - @.
- Las claves y los valores distinguen entre mayúsculas y minúsculas.

Notas sobre el uso de etiquetas

Cuando utilice etiquetas en Aurora DSQL, tenga en cuenta lo siguiente.

- Cuando utilice la AWS CLI o las operaciones de la API de Aurora DSQL, asegúrese de proporcionar el nombre de recurso de Amazon (ARN) del recurso de Aurora DSQL con el que va a trabajar. Para obtener más información, consulte Formato de nombres de recurso de Amazon (ARN) para recursos de Aurora DSQL.
- Cada recurso tiene un conjunto de etiquetas, que es una colección de una o varias etiquetas asignadas al recurso.
- Cada recurso puede tener hasta 50 etiquetas por cada conjunto de etiquetas.
- Si elimina un recurso, se eliminarán todas las etiquetas asociadas.
- Puede agregar etiquetas cuando cree un recurso, puede ver y modificar etiquetas con las siguientes operaciones de la API: `TagResource`, `UntagResource` y `ListTagsForResource`.
- Puede utilizar etiquetas con las políticas de IAM. Puede utilizarlas para administrar el acceso a los clústeres de Aurora DSQL y para controlar qué acciones se pueden aplicar a esos recursos. Para obtener más información, consulte Control del acceso a los recursos de AWS mediante etiquetas.
- Puede utilizar etiquetas para otras actividades en AWS. Para obtener más información, consulte Estrategias de etiquetado comunes.

Consideraciones para trabajar con Amazon Aurora DSQL

Tenga en cuenta los siguientes comportamientos cuando trabaje con Amazon Aurora DSQL. Para obtener más información acerca de la compatibilidad y el soporte de PostgreSQL, consulte Compatibilidad con características SQL en Aurora DSQL. Para obtener información acerca de las cuotas y los límites, consulte Cuotas de clúster y límites de base de datos en Amazon Aurora DSQL.

- Aurora DSQL no completa las operaciones `COUNT(*)` antes del tiempo de espera de la transacción en el caso de tablas grandes. Para recuperar el recuento de filas de tabla del catálogo del sistema, consulte Uso de tablas y comandos del sistema en Aurora DSQL.
- Los controladores que llaman a `PG_PREPARED_STATEMENTS` podrían proporcionar una vista incoherente de las instrucciones preparadas almacenadas en caché para el clúster. Podría ver más del número esperado de instrucciones preparadas por conexión para el mismo clúster y rol de IAM. Aurora DSQL no conserva los nombres de instrucción que prepare.
- En escenarios poco frecuentes de deterioro del clúster vinculado multirregional, es posible que la disponibilidad de la confirmación de transacciones tarde más de lo esperado en reanudarse. En general, las operaciones automatizadas de recuperación de clústeres pueden dar lugar a errores temporales de control de simultaneidad o de conexión. En la mayoría de los casos, solo verá los efectos para un porcentaje de la carga de trabajo. Cuando vea estos errores de tránsito, vuelva a intentar la transacción o vuelva a conectarse con el cliente.
- Algunos clientes de SQL, como Datagrip, realizan llamadas expansivas a los metadatos del sistema para rellenar la información del esquema. Aurora DSQL no admite toda esta información y devuelve errores. Este problema no afecta la funcionalidad de las consultas SQL, pero podría afectar la visualización del esquema.
- El rol de administrador tiene un conjunto de permisos relacionados con las tareas de administración de bases de datos. De forma predeterminada, estos permisos no se extienden a los objetos que crean otros usuarios. El rol de administrador no puede conceder ni revocar permisos en estos objetos creados por el usuario a otros usuarios. El usuario administrador puede concederse a sí mismo cualquier otro rol para obtener los permisos necesarios en estos objetos.

Cuotas de clúster y límites de base de datos en Amazon Aurora DSQL

Las siguientes secciones describen las cuotas de clúster y los límites de base de datos para Aurora DSQL.

Cuotas de clúster

La Cuenta de AWS tiene las siguientes cuotas de clúster en Aurora DSQL. Para solicitar un aumento de las cuotas de servicio para clústeres de una región o multirregionales en una Región de AWS específica, utilice la página de la consola de Service Quotas. Para otros aumentos de cuotas, póngase en contacto con AWS Support.

Descripción	Límite predeterminado	¿Configurable?	Código de error de Aurora DSQL
Máximo de clústeres de región única por Cuenta de AWS	20 clústeres	Sí	Código de error de la API `ServiceQuotaExceededE` `402`
Número máximo de clústeres de varias regiones por Cuenta de AWS	5 clústeres	Sí	Código de error de la API `ServiceQuotaExceededE` `402`
Almacenamiento máximo por clúster	Límite predeterminado de 10 TiB, hasta 128 TiB con aumento	Sí	`DISK_FULL(53100)`

Descripción	Límite predeterminado	¿Configurable?	Código de error de Aurora DSQL
	de límite aprobado		
Conexiones máximas por clúster	10 000 conexiones	Sí	`TOO_MANY_CONNECTIONS(53300)`
Tasa máxima de conexiones por clúster	100 conexiones por segundo	No	`CONFIGURED_LIMIT_EXCEEDED(53400)`
Capacidad máxima de capacidad de ampliación por clúster	1000 conexiones	No	Sin código de error
Máximo de trabajos de restauración simultáneos	4	No	Sin código de error
Tasa de reposición de conexiones	100 conexiones por segundo	No	Sin código de error

Límites de base de datos en Aurora DSQL

En la siguiente tabla se describen los límites de la base de datos en Aurora DSQL.

Descripción	Límite predeterminado	¿Configurable?	Código de error de Aurora DSQL	Mensaje de error
Tamaño máximo combinado de las columnas utilizadas en una clave principal	1 KiB	No	54000	ERROR: key size too large
Tamaño máximo combinado de las columnas de un índice secundario	1 KiB	No	54000	ERROR: key size too large
Tamaño máximo de una fila en una tabla	2 MiB	No	54000	ERROR: maximum row size excee
Tamaño máximo de una columna que no forma parte de un índice	1 MiB	No	54000	ERROR: maximum column size ex
Número máximo de columnas en una clave principal o un índice secundario	8	No	54011	ERROR: more than 8 column key are not supported

Descripción	Límite predeterminado	¿Configurable?	Código de error de Aurora DSQL	Mensaje de error
Número máximo de columnas en una tabla	255	No	`54011`	`ERROR: tables can have at mos`
Número máximo de índices en una tabla	24	No	`54000`	`ERROR: more than 24 indexes p allowed`
Tamaño máximo de todos los datos modificados en una transacción de escritura	10 MiB	No	`54000`	`ERROR: transaction size limit DETAIL: Current transaction s 10mb`
Número máximo de filas de tablas e índices que se pueden mutar en un bloque de transacción	3000 filas por transacción. Consulte Consideraciones de Aurora DSQL para la compatibilidad de PostgreSQL.	No	`54000`	`ERROR: transaction row limit`

Descripción	Límite predeterminado	¿Configurable?	Código de error de Aurora DSQL	Mensaje de error
Cantidad máxima de memoria base que puede utilizar una operación de consulta	128 MiB por transacción	No	53200	ERROR: query requires too muc out of memory.
Número máximo de esquemas definidos en una base de datos	10	No	54000	ERROR: more than 10 schemas n
Número máximo de tablas en una base de datos	1000 tablas	No	54000	ERROR: creating more than 100 allowed
Número máximo de bases de datos en un clúster	1	No	Sin código de error	ERROR: unsupported statement
Tiempo máximo de transacción	5 minutos	No	54000	ERROR: transaction age limit exceeded
Duración máxima de la conexión	60 minutos	No	Sin código de error	Sin mensaje de error
Número máximo de vistas en una base de datos	5 000	No	54000	ERROR: creating more than 500 allowed

Descripción	Límite predeterminado	¿Configurable?	Código de error de Aurora DSQL	Mensaje de error
Tamaño máximo de definición de vista	2 MiB	No	`54000`	`ERROR: view definition too la`

Para conocer los límites de tipos de datos específicos de Aurora DSQL, consulte Tipos de datos admitidos en Aurora DSQL.

Referencia de la API de Aurora DSQL

Además de la AWS Management Console y la AWS Command Line Interface (AWS CLI), Aurora DSQL también proporciona una interfaz API. Puede utilizar las operaciones de la API para administrar los recursos en Aurora DSQL.

Para ver una lista de acciones de la API ordenada alfabéticamente, consulte el tema relacionado con las acciones.

Para ver una lista de tipos de datos ordenada alfabéticamente, consulte el tema relacionado con los Tipos de datos.

Para ver una lista de parámetros de consulta comunes, consulte el tema relacionado con los Parámetros comunes.

Para ver las descripciones de los códigos de error, consulte el tema relacionado con los Errores comunes.

Para obtener más información sobre la AWS CLI, consulte la referencia de la AWS Command Line Interface para Aurora DSQL.

Solución de problemas en Aurora DSQL

> **Note**
>
> En los siguientes temas se proporcionan consejos para la solución de problemas y errores que puede encontrar al utilizar Aurora DSQL. Si encuentra un problema que no figura en esta lista, póngase en contacto con el servicio de asistencia de AWS

Temas

Solución de problemas de errores de conexión

error: código de error SSL no reconocido: 6

Causa: puede que esté utilizando una versión de psql anterior a la versión 14, que no admite la indicación de nombre de servidor (SNI). La SNI es necesaria cuando se conecta a Aurora DSQL.

Puede comprobar la versión de cliente con `psql --version`.

error: NetworkUnreachable

Es posible que un error `NetworkUnreachable` durante los intentos de conexión indique que el cliente no admite conexiones IPv6, en lugar de señalar un problema real de red. Este error suele producirse en instancias que solo utilizan IPv4 debido a la forma en que los clientes de PostgreSQL gestionan las conexiones de doble pila. Cuando un servidor admite el modo de doble pila, estos clientes primero resuelven los nombres de host a direcciones IPv4 e IPv6. Primero intentan establecer una conexión IPv4 y, a continuación, prueban con IPv6 si la conexión inicial produce un

error. Si el sistema no es compatible con IPv6, verá un error general `NetworkUnreachable` en lugar de un mensaje claro que diga “IPv6 no es compatible”.

Solución de problemas de errores de autenticación

Error de autenticación de IAM para el usuario “…”

Cuando genera un token de autenticación de IAM de Aurora DSQL, la duración máxima que puede establecer es de una semana. Después de una semana, no podrá autenticarse con ese token.

Además, Aurora DSQL rechaza la solicitud de conexión si el rol asumido ha caducado. Por ejemplo, si intenta conectarse con un rol de IAM temporal aunque el token de autenticación no haya caducado, Aurora DSQL rechazará la solicitud de conexión.

Para obtener más información sobre cómo funciona IAM con Aurora DSQL, consulte Descripción de la autenticación y autorización para Aurora DSQL y AWS Identity and Access Management en Aurora DSQL.

An error occurred (InvalidAccessKeyId) when calling the GetObject operation: The AWS Access Key ID you provided does not exist in our records

IAM ha rechazado la solicitud. Para obtener más información, consulte Por qué se firman las solicitudes.

IAM role <role> does not exist

Aurora DSQL no ha podido encontrar el rol de IAM. Para obtener más información, consulte Roles de IAM.

IAM role must look like an IAM ARN

Consulte Identificadores IAM: ARN de IAM para obtener más información.

Solución de problemas de errores de autorización

Role <role> not supported

Aurora DSQL no admite la operación `GRANT`. Consulte Subconjuntos de comandos PostgreSQL admitidos en Aurora DSQL.

Cannot establish trust with role <role>

Aurora DSQL no admite la operación `GRANT`. Consulte Subconjuntos de comandos PostgreSQL admitidos en Aurora DSQL.

El rol <rol> no existe

Aurora DSQL no ha podido encontrar el usuario de base de datos especificado. Consulte Autorización de roles personalizados de base de datos para conectarse a un clúster.

ERROR: permission denied to grant IAM trust with role <role>

Para conceder acceso a un rol de base de datos, debe estar conectado al clúster con el rol de administrador. Para obtener más información, consulte Autorización de roles de base de datos para utilizar SQL en una base de datos.

ERROR: role <role> must have the LOGIN attribute

Todos los roles de base de datos que cree deben tener el permiso `LOGIN`.

Para solucionar este error, asegúrese de que ha creado el rol de PostgreSQL con el permiso `LOGIN`. Para obtener más información, consulte CREATE ROLE y ALTER ROLE en la documentación de PostgreSQL.

ERROR: role <role> cannot be dropped because some objects depend on it

Aurora DSQL devuelve un error si descarta un rol de base de datos con una relación de IAM hasta que revoque la relación mediante `AWS IAM REVOKE`. Para obtener más información, consulte Revocación de la autorización.

Solución de errores de SQL

Error: Not supported

Aurora DSQL no admite todos los dialectos basados en PostgreSQL. Para saber lo que se admite, consulte Características de PostgreSQL admitidas en Aurora DSQL.

Error: SELECT FOR UPDATE in a read-only transaction is a no-op

Está intentando una operación que no está permitida en una transacción de solo lectura. Para obtener más información, consulte Descripción del control de simultaneidad en Aurora DSQL.

Error: use **`CREATE INDEX ASYNC`** instead

Para crear un índice en una tabla con filas existentes, debe utilizar el comando `CREATE INDEX ASYNC`. Para obtener más información, consulte Creación de índices de forma asíncrona en Aurora DSQL.

Solución de errores de OCC

OC000 “ERROR: mutation conflicts with another transaction, retry as needed”

OC001 “ERROR: schema has been updated by another transaction, retry as needed”

La sesión de PostgreSQL tenía una copia en caché del catálogo de esquemas. Esa copia en caché era válida en el momento en que se cargó. Llamemos al tiempo T1 y a la versión V1.

Otra transacción actualiza el catálogo en el momento T2. Llamémosla V2.

Cuando la sesión original intenta leer del almacenamiento en el tiempo T2 todavía está utilizando la versión V1 del catálogo. La capa de almacenamiento de Aurora DSQL rechaza la solicitud porque la última versión del catálogo en T2 es V2.

Cuando se reintenta en el tiempo T3 desde la sesión original, Aurora DSQL actualiza la memoria caché del catálogo. La transacción en T3 utiliza el catálogo V2. Aurora DSQL finalizará la transacción siempre y cuando no se hayan producido otros cambios en el catálogo desde el tiempo T2.

Solución de problemas de las conexiones SSL/TLS

Error de SSL: no se pudo verificar el certificado

Este error indica que el cliente no puede verificar el certificado del servidor. Asegúrese de que:

1. El certificado Amazon Root CA 1 está instalado correctamente. Consulte Configuración de certificados SSL/TLS para conexiones de Aurora DSQL para obtener instrucciones sobre cómo validar e instalar este certificado.
2. La variable de entorno `PGSSLROOTCERT` apunta al archivo de certificado correcto.
3. El archivo de certificado tiene los permisos correctos.

Código de error SSL no reconocido: 6

Este error se produce con los clientes de PostgreSQL anteriores a la versión 14. Actualice el cliente de PostgreSQL a la versión 17 para resolver este problema.

Error de SSL: esquema no registrado (Windows)

Este es un problema conocido con el cliente psql de Windows cuando se utilizan certificados del sistema. Utilice el método del archivo de certificado descargado que se describe en las instrucciones de Conexión desde Windows.

Historial de documentos para la Guía del usuario de Amazon Aurora DSQL

En la siguiente tabla se describen las versiones de la documentación de Aurora DSQL.

Cambio	Descripción	Fecha
Disponibilidad general (GA) de Amazon Aurora DSQL	Amazon Aurora DSQL ya está disponible de forma general con soporte adicional para la supervisión de CloudWatch, características de protección de datos mejoradas e integración AWS Backup. Para obtener más información, consulte Supervisión de Aurora DSQL con CloudWatch, Copia de seguridad y restauración para Amazon Aurora DSQL, y Cifrado de datos para Amazon Aurora DSQL.	27 de mayo de 2025
Actualización de AmazonAuroraDSQLFullAccess	Agrega la capacidad de realizar operaciones de copia de seguridad y restauración para los clústeres de Aurora DSQL, incluidos los trabajos de inicio, detención y supervisión. También agrega la capacidad de utilizar claves de KMS administradas por el cliente para el cifrado de clústeres. Para obtener más información, consulte AmazonAuroraDSQLFullAccess y Uso de roles	21 de mayo de 2025

	vinculados a servicios en Aurora DSQL.	
Actualización de AmazonAuroraDSQLConsoleFullAccess	Agrega la capacidad de realizar operaciones de copia de seguridad y restauración para los clústeres de Aurora DSQL a través de AWS Console Home. Esto incluye iniciar, detener y supervisar los trabajos. También admite el uso de claves de KMS administradas por el cliente para el cifrado de clústeres y el lanzamiento de AWS CloudShell. Para obtener más información, consulte AmazonAuroraDSQLConsoleFullAccess y Uso de roles vinculados al servicio en Aurora DSQL.	21 de mayo de 2025

Actualización de AmazonAuroraDSQLReadOnlyAccess	Incluye la capacidad de determinar el nombre de servicio de punto de conexión de VPC correcto al conectarse a los clústeres de Aurora DSQL a través de AWS PrivateLink Aurora DSQL crea puntos de conexión únicos por celda, por lo que esta API lo ayuda a asegurarse de que puede identificar el punto de conexión correcto para el clúster y evitar errores de conexión. Para obtener más información, consulte AmazonAuroraDSQLReadOnlyAccess y Uso de roles vinculados al servicio en Aurora DSQL.	13 de mayo de 2025

Actualización de AmazonAuroraDSQLFullAccess	La política agrega cuatro nuevos permisos para crear y administrar clústeres de bases de datos en múltiples Regiones de AWS: `PutMultiRegionProperties` , `PutWitnessRegion` , `AddPeerCluster` y `RemovePeerCluster` . Estos permisos incluyen controles de recursos y claves de condición para que pueda controlar qué usuarios de clústeres puede modificar. La política también agrega el permiso `GetVpcEndpointServiceName` para ayudarlo a conectarse a los clústeres de Aurora DSQL a través de AWS PrivateLink. Para obtener más información, consulte AmazonAuroraDSQLConsoleFullAccess y Uso de roles vinculados al servicio en Aurora DSQL.	13 de mayo de 2025

Actualización de AmazonAuroraDSQLConsoleFullAccess	Agrega nuevos permisos a Aurora DSQL para admitir la administración de clústeres multirregionales y la conexión de puntos de conexión de VPC. Los nuevos permisos incluyen `PutMultiRegionProperties`, `PutWitnessRegion`, `AddPeerCluster`, `RemovePeerCluster` y `GetVpcEndpointServiceName`. Consulte AmazonAuroraDSQLConsoleFullAccess y Uso de roles vinculados a servicios en Aurora DSQL.	13 de mayo de 2025
Actualización de AuroraDsqlServiceLinkedRolePolicy	Agrega a la política la capacidad de publicar métricas en `AWS/AuroraDSQL` y los espacios de nombres de `AWS/Usage` `CloudWatch`. Esto permite que el servicio o el rol asociado emita datos de uso y rendimiento más completos al entorno de CloudWatch. Para obtener más información, consulte AuroraDsqlServiceLinkedRolePolicy y Uso de roles vinculados al servicio en Aurora DSQL.	8 de mayo de 2025

AWS PrivateLink para Amazon Aurora DSQL	Aurora DSQL ahora admite AWS PrivateLink. Con AWS PrivateLink, puede simplific ar la conectividad de red privada entre nubes privadas virtuales (VPC), Aurora DSQL y los centros de datos en las instalaciones mediante puntos de conexión de VPC de Amazon de interfaz y direccion es IP privadas. Para obtener más información, consulte Administración y conexión a clústeres de Amazon Aurora DSQL mediante AWS PrivateLink.	8 de mayo de 2025
Versión inicial	Versión inicial de la Guía del usuario de Amazon Aurora DSQL.	3 de diciembre de 2024