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 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 una amplia gama 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.

Nombre de la región	Región	Punto de conexión	Protocolo	
Este de EE. UU. (Ohio)	us-east-2	dsql.us-east-2.api.aws dsql-fips.us-east-2.api.aws	HTTPS HTTPS	
Este de EE. UU. (Norte de Virginia)	us-east-1	dsql.us-east-1.api.aws dsql-fips.us-east-1.api.aws	HTTPS HTTPS	
Oeste de EE. UU. (Oregón)	us-west-2	dsql.us-west-2.api.aws dsql-fips.us-west-2.api.aws	HTTPS HTTPS	
Asia-Pacífico (Melbourne)	ap-southeast-4	dsql.ap-southeast-4.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
Asia-Pacífico (Seúl)	ap-northeast-2	dsql.ap-northeast-2.api.aws	HTTPS
Asia-Pacífico (Sídney)	ap-southeast-2	dsql.ap-southeast-2.api.aws	HTTPS
Asia-Pacífico (Tokio)	ap-northeast-1	dsql.ap-northeast-1.api.aws	HTTPS
Canadá (centro)	ca-central-1	dsql.ca-central-1.api.aws dsql-fips.ca-central-1.api.aws	HTTPS HTTPS
Oeste de Canadá (Calgary)	ca-west-1	dsql.ca-west-1.api.aws dsql-fips.ca-west-1.api.aws	HTTPS HTTPS
Europa (Fráncfort)	eu-central-1	dsql.eu-central-1.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

Nombre de la región	Región	Punto de conexión	Protocolo	
Europa (París)	eu-west-3	dsql.eu-west-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 (Fráncfort)
- 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 con clientes compatibles con PostgreSQL
- Aurora DSQL y PostgreSQL

Introducción a Aurora DSQL

Amazon Aurora DSQL es una base de datos relacional distribuida, completamente gestionada 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 consola de AWS e interactuará de forma opcional con la base de datos mediante otros clientes de PostgreSQL. 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 consola.
- Su identidad de IAM debe cumplir los siguientes criterios:
  - Acceso para realizar cualquier acción en cualquier recurso de la Cuenta de AWS
  - Se adjunta la política administrada de AWS `AmazonAuroraDSQLConsoleFullAccess`.

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 Consola de administración de AWS 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) Cambie el valor de la etiqueta de nombre predeterminada.
4. (Opcional) Agregue etiquetas adicionales para este clúster.
5. (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. Si utiliza una clave administrada por el cliente, asegúrese de que la política de claves conceda a Aurora DSQL los permisos necesarios. Para obtener más información, consulte Política de claves para una clave administrada por el cliente.
   - Seleccione Habilitar la protección contra la eliminación para evitar que se elimine el clúster. De forma predeterminada, la protección contra la eliminación está seleccionada.
   - Seleccione Política basada en recursos (avanzada) para especificar las políticas de control de acceso para este clúster.
6. Elija Create cluster.
7. La consola le devuelve a la página Clústeres. Aparece un banner de notificación que indica que se ha creado el clúster. Seleccione el ID del clúster para abrir la vista de detalles del clúster.

Paso 2: conexión al clúster de Aurora DSQL

Aurora SQL admite varias formas de conectarse al clúster, como el editor de consultas de DSQL, AWS CloudShell, el cliente psql local y otras herramientas compatibles con PostgreSQL. En este paso, se conecta mediante el editor de consultas de Aurora DSQL, que proporciona una forma rápida de empezar a interactuar con el nuevo clúster.

Conexión mediante el editor de consultas

1. En la consola de Aurora DSQL (https://console.aws.amazon.com/dsql), abra la página Clústeres y confirme que la creación del clúster se ha completado y que su estado es Activo.
2. Elija el clúster de la lista o elija el ID de clúster para abrir la página de detalles del clúster.
3. Elija Conectar con el editor de consultas.
4. Seleccione Conectar como administrador del clúster que se acaba de crear.

- Opcionalmente, puede conectarse con un rol personalizado, consulte Uso de roles de base de datos y autenticación 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. Tras abrir el clúster en el editor de consultas, seleccione y ejecute cada consulta de ejemplo paso a paso.

Ejecución de comandos SQL de ejemplo en Aurora DSQL

1. Cree un esquema denominado `test`.

```
CREATE SCHEMA IF NOT EXISTS test;
```

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

```
CREATE TABLE IF NOT EXISTS test.hello_world (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    message VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3. Inserte una fila de ejemplo.

```
INSERT INTO test.hello_world (message)
VALUES ('Hello, World!!');
```

4. Lea los valores insertados.

```
SELECT * FROM test.hello_world;
```

5. Limpieza opcional

```
DROP TABLE test.hello_world;
DROP SCHEMA test;
```

Paso 4 (opcional): creación de un clúster multirregional

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.

Use el siguiente procedimiento para crear un clúster inicial, crear un segundo clúster en una región diferente y, a continuación, emparejar los dos clústeres para crear un clúster multirregional. 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 consola de Aurora DSQL.
2. En el panel de navegación, seleccione Clusters (Clústeres).
3. Elija Crear clúster y, a continuación, Varias regiones.
4. (Opcional) Cambie el valor de la etiqueta de nombre predeterminada.
5. (Opcional) Agregue etiquetas adicionales para este clúster.
6. 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.

7. (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. Si utiliza una clave administrada por el cliente, asegúrese de que la política de claves conceda a Aurora DSQL los permisos necesarios. Para obtener más información, consulte Política de claves para una clave administrada por el cliente.
   - Seleccione Habilitar la protección contra la eliminación para evitar que se elimine el clúster. De forma predeterminada, la protección contra la eliminación está seleccionada.
   - Seleccione Política basada en recursos (avanzada) para especificar las políticas de control de acceso para este clúster.

8. 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.
9. 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.
10. 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.

11. 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.
12. (Opcional) Elija la configuración del clúster para el segundo clúster. Por ejemplo, puede elegir una AWS KMS key. Si utiliza una clave administrada por el cliente, asegúrese de que la política de claves conceda a Aurora DSQL los permisos necesarios. Para obtener más información, consulte Política de claves para una clave administrada por el cliente.
13. Elija Crear clúster para crear el segundo clúster.
14. Elija Emparejar en ***initial-cluster-region***, donde *initial-cluster-region* es la región que aloja el primer clúster que creó.
15. 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. Elija Conectar con el editor de consultas.
5. Elija Conectar como administrador.
6. Cree un esquema y tabla de ejemplo e inserte datos siguiendo los pasos de Paso 3: ejecución de comandos SQL de ejemplo en Aurora DSQL.

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. Elija Conectar con el editor de consultas.
5. Elija Conectar como administrador.
6. Consulte los datos que ha insertado en el segundo clúster.

   Example

   ```
   SELECT * FROM test.hello_world;
   ```

Solución de problemas

Consulte la sección Solución de problemas de la documentación de Aurora DSQL.

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 Consola de administración de AWS, 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 describir un clúster, asegúrese de que la identidad de IAM tiene permisos para la acción de IAM `dsql:GetCluster`, como en el siguiente ejemplo de acción de política.

```
{
  "Effect": "Allow",
  "Action": "dsql:GetCluster",
  "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, la AWS CLI o los SDK, el token caduca automáticamente en 15 minutos de forma predeterminada. 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 Consola de administración de AWS 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 (opcional): creación de un clúster multirregional.
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 que pueda generar un token de autenticación mediante AWS CloudShell, asegúrese de haber creado un clúster de Aurora DSQL.

Generación de un token de autenticación mediante AWS CloudShell

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 la parte inferior izquierda de la consola de AWS, elija AWS CloudShell.

3. 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?

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

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

5. 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:
```

6. 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:

> **Note**
>
> La generación de tokens es una operación local que firma la solicitud utilizando sus credenciales actuales de IAM. No contacta con AWS para validar las credenciales. Si sus credenciales han caducado o no son válidas, la generación del token se lleva a cabo con éxito, pero el intento de conexión falla. Asegúrese de que sus credenciales de IAM sean válidas antes de generar un token.

- 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

> **Tip**
>
> AWS recomienda usar el Conector de Aurora DSQL para Python, que gestiona la generación de tokens automáticamente.

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`.

```
import boto3

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) {
    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;
    return token;
}

int main() {
    Aws::SDKOptions options;
    Aws::InitAPI(options);
    // Replace with your cluster endpoint and region
    std::string token = generateToken("your_cluster_endpoint.dsql.us-east-1.on.aws",
 "us-east-1");
    Aws::ShutdownAPI(options);
    return 0;
}
```

JavaScript SDK

> **Tip**
>
> AWS recomienda usar el Conectores de Aurora DSQL para Node.js, que gestiona la generación de tokens automáticamente.

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

> **Tip**
>
> AWS recomienda usar el Conexión a clústeres de Aurora DSQL con un conector de JDBC, que gestiona la generación de tokens automáticamente.

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.builder().build())
                .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

> **Tip**
>
> AWS recomienda usar el Conexión a clústeres de Aurora DSQL con un conector de Ruby, que gestiona la generación de tokens automáticamente.

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::CredentialProviderChain.new.resolve

  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
  })
end
```

PHP 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`.

```
<?php
require 'vendor/autoload.php';

use Aws\DSQL\AuthTokenGenerator;
use Aws\Credentials\CredentialProvider;

function generateToken(string $yourClusterEndpoint, string $region): string {
    $provider = CredentialProvider::defaultProvider();
    $generator = new AuthTokenGenerator($provider);

    // Use generateDbConnectAuthToken if you are not connecting as admin
    $token = $generator->generateDbConnectAdminAuthToken($yourClusterEndpoint,
 $region);

    echo $token . PHP_EOL;
    return $token;
}
```

.NET

> **Tip**
>
> AWS recomienda usar el Conexión a clústeres de Aurora DSQL con un conector de .NET, que gestiona la generación de tokens automáticamente.

> **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;

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

// Use `DSQLAuthTokenGenerator.GenerateDbConnectAuthToken` if you are _not_ logging
 in as `admin` user
var token =
 DSQLAuthTokenGenerator.GenerateDbConnectAdminAuthToken(RegionEndpoint.USEast1,
 yourClusterEndpoint);

Console.WriteLine(token);
```

Go

> **Tip**
>
> AWS recomienda usar el Conexión a clústeres de Aurora DSQL con un conector de Go, que gestiona la generación de tokens automáticamente.

El SDK de AWS para Go v2 proporciona un método integrado para generar tokens de autenticación en el paquete `github.com/aws/aws-sdk-go-v2/feature/dsql/auth`.

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

```
package main

import (
 "context"
 "fmt"

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

func main() {
 ctx := context.Background()

 cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion("region"))
 if err != nil {
  panic(err)
 }

 // Use auth.GenerateDbConnectAuthToken for non-admin users
 token, err := auth.GenerateDBConnectAdminAuthToken(ctx, "yourClusterEndpoint",
 "region", cfg.Credentials)
 if err != nil {
  panic(err)
 }

 fmt.Println(token)
}
```

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 Consola de administración de AWS

- 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.

Visualización de las asignaciones de roles de IAM a bases de datos

Para ver las asignaciones entre los roles de IAM y los roles de base de datos, consulte la tabla del sistema `sys.iam_pg_role_mappings`.

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

Ejemplo de código de salida:

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

En esta tabla, se muestran todas las asignaciones entre los roles de IAM (identificados por su ARN) y los roles de base de datos 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 de 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 funciones más destacadas 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`.

Sintaxis 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.

Ventajas de la arquitectura distribuida

El diseño distribuido y sin recursos compartidos de Aurora DSQL ofrece ventajas en cuanto a rendimiento y escalabilidad que superan a las bases de datos tradicionales de un solo nodo. Las funciones principales 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 sobre el uso del comando EXPLAIN en PostgreSQL, consulte DDL y las transacciones distribuidas en Aurora DSQL.

Compatibilidad con características SQL en Aurora DSQL

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.	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 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.

Funciones de tiempo de ejecución de JSON

Aurora DSQL admite JSON y JSONB como tipos de datos en tiempo de ejecución para el procesamiento de consultas. Almacene los datos JSON como columnas de `text` y conviértalos a JSON durante la ejecución de la consulta para utilizar las funciones y operadores JSON de PostgreSQL.

Aurora DSQL admite la mayoría de las funciones de PostgreSQL JSON de la sección 9.1.6 Funciones y operadores JSON con un comportamiento idéntico.

Las funciones que devuelven tipos JSON o JSONB pueden requerir una conversión adicional a `text` para que se muestren correctamente.

```
SELECT json_build_array(1, 2, 'foo', 4, 5)::text;
```

La función devuelve una respuesta similar a lo siguiente:

```
    json_build_array
---------------------
  [1, 2, "foo", 4, 5]
(1 row)
```

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` Comando de la

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`	
`USING`	
`WITH` (expresiones de tabla comunes)	
`INNER JOIN`	`ON`
`OUTER JOIN`	`LEFT`, `RIGHT`, `FULL`, `ON`
`CROSS JOIN`	`ON`
`UNION`	`ALL`
`INTERSECT`	`ALL`
`EXCEPT`	`ALL`

Cláusula principal	Cláusulas admitidas
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	
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.

Comando	Cláusula principal	Cláusulas admitidas
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	SEQUENCE	Para obtener información sobre la sintaxis admitida del comando CREATE SEQUENCE, consulte CREATE SEQUENCE.
ALTER	SEQUENCE	Para obtener información sobre la sintaxis admitida del comando ALTER SEQUENCE, consulte ALTER SEQUENCE.
DROP	SEQUENCE	Para obtener información sobre la sintaxis admitida del comando DROP SEQUENCE, consulte DROP SEQUENCE.
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 SELECT
UPDATE	SET	WHERE (SELECT)

Comando	Cláusula principal	Cláusulas admitidas
		`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	Alias
`COMMIT`	`[WORK | TRANSACTION ]` `[AND NO CHAIN]`	`END`

`BEGIN`	`[WORK | TRANSACTION ]`
	`[ISOLATION LEVEL REPEATABLE READ]`
	`[READ WRITE | READ ONLY]`	

`START TRANSACTION`	`[ISOLATION LEVEL REPEATABLE READ]` `[READ WRITE | READ ONLY]`	
`ROLLBACK`	`[WORK | TRANSACTION ]`	`ABORT`

Comando	Cláusulas admitidas	Alias
	[AND NO CHAIN]	

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 proporciona información detallada sobre los comandos SQL compatibles, centrándose en los comandos con conjuntos de parámetros y subcomandos extensos. Por ejemplo, CREATE TABLE en PostgreSQL ofrece muchas cláusulas y parámetros, un subconjunto de los cuales es compatible con Aurora DSQL. En esta sección se describen todos los elementos de la sintaxis de PostgreSQL que Aurora DSQL admite para estos comandos.

Temas

- CREATE TABLE
- ALTER TABLE
- CREATE SEQUENCE
- ALTER SEQUENCE
- DROP SEQUENCE
- CREATE VIEW
- ALTER VIEW
- DROP VIEW

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 |
  GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY ( sequence_options ) |
  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 [, ... ] ) ]
```

Columnas de identidad

> **Note**
>
> Cuando se utilizan columnas de identidad, se debe considerar con cuidado el valor de la caché. Para obtener más información, consulte el aviso Importante de la página CREATE SEQUENCE.
>
> Para obtener orientación sobre la mejor manera de utilizar las columnas de identidad en función de los patrones de carga de trabajo, consulte Trabajar con secuencias y columnas de identidad.

La cláusula GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY (*sequence_options*) crea la columna como una columna de identidad. Tendrá una secuencia implícita asociada y, en las filas recién insertadas, la columna tendrá automáticamente los valores de la secuencia que se le haya asignado. Dicha columna es implícitamente NOT NULL.

Las cláusulas ALWAYS y BY DEFAULT determinar cómo se gestionan explícitamente los valores especificados por el usuario en los comandos INSERT y UPDATE.

En un comando INSERT, si se selecciona ALWAYS, solo se acepta un valor especificado por el usuario si la instrucción INSERT especifica OVERRIDING SYSTEM VALUE. Si se selecciona BY DEFAULT, prevalece el valor especificado por el usuario.

En un comando UPDATE, si se selecciona ALWAYS, cualquier actualización de la columna a un valor distinto de DEFAULT se rechazará. Si se selecciona BY DEFAULT, la columna se puede actualizar de forma normal. (No hay una cláusula OVERRIDING para el comando UPDATE).

La cláusula *sequence_options* se puede utilizar para anular los parámetros de la secuencia. Las opciones disponibles incluyen las que se muestran para CREATE SEQUENCE más SEQUENCE NAME *name*. Sin SEQUENCE NAME, el sistema elige un nombre no utilizado para la secuencia.

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
    ALTER [ COLUMN ] column_name { SET GENERATED { ALWAYS | BY DEFAULT } | SET
 sequence_option | RESTART [ [ WITH ] restart ] } [...]
    ALTER [ COLUMN ] column_name DROP IDENTITY [ IF EXISTS ]
```

```
    OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER | SESSION_USER }
```

Acciones de la columna de identidad

`SET GENERATED { ALWAYS | BY DEFAULT }` / `SET` *`sequence_option`* / `RESTART`

Estos formularios cambian si una columna es una columna de identidad o cambian el atributo de generación de una columna de identidad existente. Para obtener más información, consulte `CREATE TABLE`. Al igual que `SET DEFAULT`, estos formularios solo afectan al comportamiento de los comandos `INSERT` y `UPDATE` posteriores; no provocan cambios en las filas que ya se encuentran en la tabla.

La opción *`sequence_option`* es una opción compatible con `ALTER SEQUENCE` como `INCREMENT BY`. Estos formularios modifican la secuencia que subyace a una columna de identidad existente.

`DROP IDENTITY [ IF EXISTS ]`

Este formulario elimina la propiedad de identidad de una columna. Si se especifica `DROP IDENTITY IF EXISTS` y la columna no es una columna de identidad, no se genera ningún error. En este caso, se emite un aviso en su lugar.

CREATE SEQUENCE

`CREATE SEQUENCE`: definir un nuevo generador de secuencias.

> **Important**
>
> En PostgreSQL, especificar `CACHE` es opcional y el valor predeterminado es 1. En un sistema distribuido como Amazon Aurora DSQL, las operaciones de secuencia implican coordinación, y un tamaño de caché de 1 puede aumentar la sobrecarga de coordinación en condiciones de alta simultaneidad. Si bien los valores de caché más grandes permiten que los números de secuencia se sirvan desde rangos preasignados localmente, lo que mejora el rendimiento, los valores reservados no utilizados pueden perderse, lo que hace que las brechas y los efectos de ordenación sean más visibles. Dado que las aplicaciones difieren en su sensibilidad al orden de asignación frente al rendimiento, Amazon Aurora DSQL requiere que se especifique explícitamente `CACHE` y, actualmente, admite `CACHE = 1` o `CACHE >= 65536`, lo que proporciona una clara distinción entre el comportamiento de asignación más cercano a la generación estrictamente secuencial y la asignación optimizada para cargas de trabajo altamente simultáneas.

Cuando `CACHE >= 65536`, los valores de secuencia siguen garantizando su unicidad, pero es posible que no se generen en estricto orden ascendente entre sesiones, y pueden producirse brechas, especialmente cuando los valores en caché no se consumen por completo. Estas características son coherentes con la semántica de PostgreSQL para secuencias en caché en uso simultáneo, donde ambos sistemas garantizan valores distintos, pero no garantizan un orden estrictamente secuencial entre sesiones.
Dentro de una misma sesión de cliente, es posible que los valores de secuencia no siempre aparezcan en estricto orden ascendente, especialmente fuera de transacciones explícitas. Este comportamiento es similar al de las implementaciones de PostgreSQL que utilizan agrupación de conexiones. Se puede lograr un comportamiento de asignación más cercano al de un entorno PostgreSQL de sesión única utilizando `CACHE = 1` u obteniendo valores de secuencia dentro de transacciones explícitas.
Con `CACHE = 1`, la asignación de secuencias sigue el comportamiento de secuencias no en caché de PostgreSQL.
Para obtener orientación sobre cómo utilizar mejor las secuencias basadas en patrones de carga de trabajo, consulte Trabajar con secuencias y columnas de identidad.

Sintaxis admitida

```
CREATE SEQUENCE [ IF NOT EXISTS ] name CACHE cache
    [ AS data_type ]
    [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]
    [ [ NO ] CYCLE ]
    [ START [ WITH ] start ]
    [ OWNED BY { table_name.column_name | NONE } ]

where data_type is BIGINT
      and cache = 1 or cache >= 65536
```

Descripción

`CREATE SEQUENCE` crea un nuevo generador de números de secuencia. Esto implica crear e inicializar una nueva tabla especial de una sola fila con el nombre *`name`*. El generador será propiedad del usuario que emita el comando.

Si se proporciona un nombre de esquema, la secuencia se crea en el esquema especificado. En caso contrario, se crea en el esquema actual. El nombre de la secuencia debe ser distinto del

nombre de cualquier otra relación (tabla, secuencia, índice, vista, vista materializada o tabla externa) en el mismo esquema.

Una vez creada una secuencia, se utilizan las funciones `nextval` `currval` y `setval` para operar en la secuencia. Estas funciones están documentadas en Funciones de manipulación de secuencias.

Aunque no puede actualizar una secuencia directamente, puede utilizar una consulta como:

```
SELECT * FROM name;
```

para examinar algunos de los parámetros y el estado actual de una secuencia. En particular, el campo `last_value` de la secuencia muestra el último valor asignado por cualquier sesión. (Por supuesto, este valor podría estar obsoleto en el momento de su impresión, si otras sesiones están realizando llamadas de `nextval` de forma activa). En la vista `pg_sequences` se pueden observar otros parámetros, como *`increment`* y *`maxvalue`*.

Parameters

`IF NOT EXISTS`

No se genera un error si ya existe una relación con el mismo nombre. En este caso, se emite un aviso. Tenga en cuenta que no hay garantía de que la relación existente sea similar a la secuencia que se habría creado; es posible que ni siquiera sea una secuencia.

`name`

El nombre (opcionalmente calificado por el esquema) de la secuencia que se va a crear.

`data_type`

La cláusula opcional `AS` *`data_type`* especifica el tipo de datos de la secuencia. Los valores válidos son `bigint`. `bigint` es el valor predeterminado. El tipo de datos determina los valores mínimo y máximo predeterminados de la secuencia.

`incremento`

La cláusula opcional `INCREMENT BY` *`increment`* especifica qué valor se añade al valor de secuencia actual para crear un valor nuevo. Un valor positivo formará una secuencia ascendente y uno negativo una secuencia descendente. El valor predeterminado es 1.

***`minvalue`* / `NO MINVALUE`**

La cláusula opcional `MINVALUE` *`minvalue`* determina el valor mínimo que puede generar una secuencia. Si esta cláusula no se proporciona o se especifica `NO MINVALUE`, se utilizarán los

valores predeterminados. El valor predeterminado para una secuencia ascendente es 1. El valor predeterminado para una secuencia descendente es el valor mínimo del tipo de datos.

maxvalue / **NO MAXVALUE**

La cláusula opcional MAXVALUE *maxvalue* determina el valor máximo para la secuencia. Si esta cláusula no se proporciona o se especifica NO MAXVALUE, se utilizarán los valores predeterminados. El valor predeterminado para una secuencia ascendente es el valor máximo del tipo de datos. El valor predeterminado para una secuencia descendente es -1.

CYCLE / **NO CYCLE**

La opción CYCLE permite que la secuencia se repita cuando los valores *maxvalue* o *minvalue* se han alcanzado mediante una secuencia ascendente o descendente, respectivamente. Si se alcanza el límite, el siguiente número generado será el valor *minvalue* o *maxvalue*, respectivamente.

Si NO CYCLE se especifica, cualquier llamada realizada a nextval después de que la secuencia haya alcanzado su valor máximo devolverá un error. Si CYCLE o NO CYCLE no se especifican, NO CYCLE es el valor predeterminado.

iniciar

La cláusula opcional START WITH *start* permite que la secuencia comience en cualquier lugar. El valor inicial predeterminado es *minvalue* para las secuencias ascendentes y *maxvalue* para las descendentes.

cache

La cláusula CACHE *cache* especifica cuántos números de secuencia se deben preasignar y almacenar en la memoria para un acceso más rápido. Los valores aceptables para CACHE en Aurora DSQL son 1 o cualquier número >= 65536. El valor mínimo es 1 (solo se puede generar un valor a la vez, lo que significa que no caché).

OWNED BY *table_name*.*column_name* / **OWNED BY NONE**

La opción OWNED BY hace que la secuencia se asocie a una columna de tabla específica, de modo que si se elimina esa columna (o su tabla completa), la secuencia también se eliminará automáticamente. La tabla especificada debe tener el mismo propietario y estar en el mismo esquema que la secuencia OWNED BY NONE, el valor predeterminado, especifica que no existe tal asociación.

Notas

Use DROP SEQUENCE para eliminar una secuencia.

Las secuencias se basan en la aritmética bigint, por lo que el rango no puede superar el rango de un entero de ocho bytes (-9223372036854775808 a 9223372036854775807).

Dado que las llamadas de nextval y setval nunca se revierten, los objetos de secuencia no se pueden utilizar si se necesita una asignación "sin brechas" de números de secuencia.

Cada sesión asignará y almacenará en caché valores de secuencia sucesivos durante un acceso al objeto de secuencia y aumentará el last_value del objeto de secuencia en consecuencia. A continuación, los siguientes usos de nextval en la *caché*-1 dentro de esa sesión simplemente devuelven los valores preasignados sin tocar el objeto de secuencia. De este modo, cualquier número asignado pero no utilizado dentro de una sesión se perderá cuando esta finalice, lo que provocará "huecos" en la secuencia.

Además, aunque se garantiza que las sesiones múltiples asignarán valores de secuencia distintos, los valores podrían generarse fuera de secuencia cuando se tienen en cuenta todas las sesiones. Por ejemplo, con una configuración de *caché* de 10, la sesión A podría reservar los valores 1..10 y devolver nextval=1; a continuación, la sesión B podría reservar los valores 11..20 y devolver nextval=11 antes de que la sesión A genere nextval=2. Por lo tanto, con una configuración de *caché* de uno, es seguro suponer que los valores nextval se generan de forma secuencial; con una configuración de *caché* superior a uno, solo se debe suponer que los valores nextval son todos distintos, no que se generan de forma puramente secuencial. Además, last_value reflejará el último valor reservado por cualquier sesión, independientemente de si nextval ya lo ha devuelto.

Otra consideración es que un setval ejecutado en dicha secuencia no será detectado por otras sesiones hasta que hayan agotado los valores preasignados que tienen almacenados en caché.

Ejemplos

Cree una secuencia ascendente llamada de serial, empezando por 101:

```
CREATE SEQUENCE serial CACHE 65536 START 101;
```

Seleccione el siguiente número de esta secuencia:

```
SELECT nextval('serial');

 nextval
```

```
---------
     101
```

Seleccione el siguiente número de esta secuencia:

```
SELECT nextval('serial');

 nextval
---------
     102
```

Utilice esta secuencia en un comando `INSERT`:

```
INSERT INTO distributors VALUES (nextval('serial'), 'nothing');
```

Restablezca la secuencia a un valor específico usando `setval`:

```
SELECT setval('serial', 200);
SELECT nextval('serial');

 nextval
---------
     201
```

Compatibilidad

`CREATE SEQUENCE` cumple con el estándar SQL con las siguientes excepciones:

- La obtención del siguiente valor se realiza mediante la función `nextval()` en lugar de la expresión `NEXT VALUE FOR` del estándar.
- La cláusula `OWNED BY` es una extensión de PostgreSQL.

`ALTER SEQUENCE`

`ALTER SEQUENCE`: cambiar la definición de un generador de secuencias.

> **Important**
>
> Al utilizar secuencias, se debe tener muy en cuenta el valor de la caché. Para obtener más información, consulte el aviso Importante de la página `CREATE SEQUENCE`.

Para obtener orientación sobre cómo utilizar mejor las secuencias basadas en patrones de carga de trabajo, consulte Trabajar con secuencias y columnas de identidad.

Sintaxis admitida

```
ALTER SEQUENCE [ IF EXISTS ] name
    [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE maxvalue | NO MAXVALUE ]
    [ [ NO ] CYCLE ]
    [ START [ WITH ] start ]
    [ RESTART [ [ WITH ] restart ] ]
    [ CACHE cache ]
    [ OWNED BY { table_name.column_name | NONE } ]
ALTER SEQUENCE [ IF EXISTS ] name OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER |
 SESSION_USER }
ALTER SEQUENCE [ IF EXISTS ] name RENAME TO new_name
ALTER SEQUENCE [ IF EXISTS ] name SET SCHEMA new_schema

where cache is 1 or cache >= 65536
```

Descripción

`ALTER SEQUENCE` cambia los parámetros de un generador de secuencias existente. Todos los parámetros que no estén establecidos específicamente en el comando `ALTER SEQUENCE` conservan su configuración anterior.

Debe ser el propietario de la secuencia para utilizar `ALTER SEQUENCE`. Para modificar el esquema de una secuencia, 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 secuencia. (Estas restricciones garantizan que modificar el propietario no haga nada que no se pueda hacer eliminando y volviendo a crear la secuencia. Sin embargo, un superusuario puede cambiar la propiedad de cualquier secuencia de todos modos).

Parameters

name

El nombre (opcionalmente calificado por el esquema) de una secuencia que se va a modificar.

`IF EXISTS`

No se genera un error si la secuencia no existe. En este caso, se emite un aviso.

`incremento`

La cláusula `INCREMENT BY` *`increment`* es opcional. Un valor positivo formará una secuencia ascendente y uno negativo una secuencia descendente. Si no se especifica, se mantendrá el valor de incremento anterior.

`minvalue` / **`NO MINVALUE`**

La cláusula opcional `MINVALUE` *`minvalue`* determina el valor mínimo que puede generar una secuencia. Si se especifica `NO MINVALUE`, se utilizarán los valores predeterminados de 1 y el valor mínimo del tipo de datos para secuencias ascendentes y descendentes, respectivamente. Si no se especifica ninguna opción, se mantendrá el valor mínimo actual.

`maxvalue` / **`NO MAXVALUE`**

La cláusula opcional `MAXVALUE` *`maxvalue`* determina el valor máximo para la secuencia. Si se especifica `NO MAXVALUE`, se utilizarán los valores predeterminados del valor máximo del tipo de datos y -1 para secuencias ascendentes y descendentes, respectivamente. Si no se especifica ninguna opción, se mantendrá el valor máximo actual.

`CYCLE`

La palabra clave `CYCLE` opcional se puede utilizar para permitir que la secuencia se repita cuando los valores *`maxvalue`* o *`minvalue`* se han alcanzado mediante una secuencia ascendente o descendente, respectivamente. Si se alcanza el límite, el siguiente número generado será el valor *`minvalue`* o *`maxvalue`*, respectivamente.

`NO CYCLE`

Si se especifica la palabra clave `NO CYCLE` opcional, cualquier llamada a `nextval` después de que la secuencia haya alcanzado su máximo valor devolverá un error. Si no se especifica `CYCLE` ni `NO CYCLE`, se mantendrá el comportamiento del ciclo antiguo.

`iniciar`

La cláusula opcional `START WITH` *`start`* cambia el valor inicial registrado de la secuencia. Esto no afecta al valor de secuencia actual; simplemente establece el valor que utilizarán los futuros comandos `ALTER SEQUENCE RESTART`.

restart

La cláusula opcional `RESTART [ WITH` *`restart`* `]` cambia el valor actual de la secuencia. Esto es similar a llamar a la función `setval` con `is_called = false`: el valor especificado se devolverá en la siguiente llamada de `nextval`. Escribir `RESTART` sin un valor de *`reinicio`* equivale a proporcionar el valor inicial registrado por `CREATE SEQUENCE` o establecido por última vez por `ALTER SEQUENCE START WITH`.

A diferencia de una llamada de `setval`, una operación `RESTART` en una secuencia es transaccional e impide que las transacciones simultáneas obtengan números de la misma secuencia. Si ese no es el modo de operación deseado, debería usarse `setval`.

cache

La cláusula `CACHE` *`cache`* permite preasignar números de secuencia y almacenarlos en la memoria para un acceso más rápido. El valor debe ser 1 o algún valor >= 65536. Si no se especifica, se mantendrá el valor de caché antiguo. Para obtener más información sobre el comportamiento de la caché, consulte las instrucciones que se incluyen en `CREATE SEQUENCE`.

`OWNED BY` ***`table_name.column_name`*** **/ `OWNED BY NONE`**

La opción `OWNED BY` hace que la secuencia se asocie a una columna de tabla específica, de modo que si se elimina esa columna (o su tabla completa), la secuencia también se eliminará automáticamente. Si se especifica, esta asociación reemplaza cualquier asociación previamente especificada para la secuencia. La tabla especificada debe tener el mismo propietario y estar en el mismo esquema que la secuencia. Especificar `OWNED BY NONE` elimina cualquier asociación existente, haciendo que la secuencia sea “independiente”.

new_owner

El nombre de usuario del nuevo propietario de la secuencia.

new_name

El nuevo nombre de la secuencia.

new_schema

El nuevo esquema de la secuencia.

Notas

`ALTER SEQUENCE` no afectará inmediatamente a los resultados de `nextval` en los backends, salvo en el actual, que tienen valores de secuencia preasignados (almacenados en caché). Utilizarán todos

los valores en caché antes de detectar los parámetros de generación de secuencias modificados. El backend actual se verá afectado de forma inmediata.

`ALTER SEQUENCE` no afecta al estado de `currval` para la secuencia.

`ALTER SEQUENCE` puede provocar otras transacciones a OCC.

Por razones históricas, `ALTER TABLE` también se puede utilizar con secuencias; pero las variantes de `ALTER TABLE` que se permiten con secuencias son solo las equivalentes a las formas mostradas anteriormente.

Ejemplos

Reinicie una secuencia llamada `serial` en 105:

```
ALTER SEQUENCE serial RESTART WITH 105;
```

Compatibilidad

`ALTER SEQUENCE` cumple con el estándar SQL, excepto por las cláusulas `AS`, `START WITH`, `OWNED BY`, `OWNER TO`, `RENAME TO` y `SET SCHEMA`, que son extensiones de PostgreSQL.

DROP SEQUENCE

`DROP SEQUENCE`: eliminar una secuencia.

Sintaxis admitida

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

Descripción

`DROP SEQUENCE` elimina los generadores de números de secuencia. Solo su propietario o un superusuario pueden eliminar una secuencia.

Parameters

IF EXISTS

No se genera un error si la secuencia no existe. En este caso, se emite un aviso.

name

El nombre (opcionalmente calificado por el esquema) de una secuencia.

`CASCADE`

Elimine automáticamente los objetos que dependen de la secuencia y, a su vez, todos los objetos que dependen de esos objetos.

`RESTRICT`

Rechace eliminar la secuencia si hay objetos que dependen de ella. Esta es la opción predeterminada.

Ejemplos

Para eliminar la secuencia `seq`:

```
DROP SEQUENCE seq;
```

Compatibilidad

`DROP SEQUENCE` cumple con el estándar SQL, excepto que el estándar solo permite eliminar una secuencia por comando, y aparte de la opción `IF EXISTS`, que es una extensión de PostgreSQL.

`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.

Parameters

`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 a nivel de 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.

`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`.
- 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.
- 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.

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';
```

Cree una vista con `LOCAL CHECK OPTION`.

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

Cree una vista recursiva.

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

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`). Para usar`ALTER VIEW`, debe ser propietario de

la vista. 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.

Parameters

`name`

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

`column_name`

Nombre de una columna existente o nombre nuevo 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.

`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] [, ... ] )`

Establece una opción de vista. Las siguientes son 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.
- `security_invoker (boolean)`: cambia la propiedad de invocador de seguridad de la vista.

`RESET ( view_option_name [, ... ] )`

Restablece una opción de vista a su valor predeterminado.

Ejemplos

Cambiar el nombre de la vista `foo` a `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.

Parameters

`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.

Migración de PostgreSQL a Aurora DSQL

Aurora DSQL se ha diseñado para ser compatible con PostgreSQL y admite características relacionales básicas como las transacciones ACID, los índices secundarios, las uniones y las operaciones DML estándar. La mayoría de las aplicaciones PostgreSQL existentes pueden migrar a Aurora DSQL con cambios mínimos.

En esta sección se proporcionan orientaciones prácticas para migrar la aplicación a Aurora DSQL, incluidos la compatibilidad con marcos, los patrones de migración y consideraciones de la arquitectura.

Compatibilidad con marcos y ORM

Aurora DSQL usa el protocolo de conexión estándar de PostgreSQL, lo que garantiza la compatibilidad con controladores y marcos de PostgreSQL. Las asignaciones relacionales de objetos (ORM) más populares funcionan con Aurora DSQL con cambios mínimos o sin cambios.

Consulte the section called “Adaptadores de Aurora DSQL” para ver implementaciones de referencia e integraciones de ORM disponibles.

Patrones de migración comunes

Al migrar de PostgreSQL a Aurora DSQL, algunas características funcionan de forma diferente o tienen una sintaxis alternativa. En esta sección se proporciona orientación sobre escenarios de migración comunes.

Alternativas a las operaciones DDL

Aurora DSQL ofrece alternativas modernas a las operaciones de lenguaje de definición de datos (DDL) tradicionales de PostgreSQL:

Creación de índices

Utilice `CREATE INDEX ASYNC` en lugar de `CREATE INDEX` para la creación de índices sin bloqueo.

Ventaja: Creación de índices sin tiempo de inactividad en tablas grandes.

Eliminación de datos

Use `DELETE FROM table_name` en lugar de `TRUNCATE`.

Alternativa: Para una recreación completa de las tablas, utilice `DROP TABLE` y, a continuación, `CREATE TABLE`.

Configuración del sistema

Aurora DSQL está totalmente administrado, por lo que la configuración se gestiona automáticamente en función de los patrones de carga de trabajo. Use la consola de administración de AWS o la API para administrar la configuración del clúster.

Ventaja: No es necesario ajustar la base de datos ni administrar los parámetros.

Patrones de diseño de esquema

Adapte estos patrones comunes de PostgreSQL para que sean compatibles con Aurora DSQL:

Patrones de integridad referencial

Aurora DSQL admite relaciones de tabla y operaciones de `JOIN`. Para garantizar la integridad referencial, implemente la validación en la capa de aplicación. Este diseño se ajusta a los

patrones modernos de bases de datos distribuidas, en los que la validación de la capa de aplicaciones proporciona más flexibilidad y evita cuellos de botella en el rendimiento derivados de las operaciones en cascada.

Patrón: Implemente comprobaciones de integridad referencial en la capa de aplicaciones mediante convenciones de nomenclatura, lógica de validación y límites de transacción coherentes. Muchas aplicaciones a gran escala prefieren este enfoque para controlar mejor la gestión de errores y el rendimiento.

Gestión de datos temporales

Utilice CTE, subconsultas o tablas normales con lógica de limpieza en lugar de tablas temporales.

Alternativa: Cree tablas con nombres específicos de cada sesión y límpielas en su aplicación.

Descripción de las diferencias arquitectónicas

La arquitectura distribuida y sin servidor de Aurora DSQL se diferencia de manera intencionada de PostgreSQL tradicional en varias áreas. Estas diferencias hacen posibles los beneficios clave de simplicidad y escala de Aurora DSQL.

Modelo simplificado de la base de datos

Base de datos única por clúster

Aurora DSQL proporciona una base de datos integrada denominada `postgres` por clúster.

Consejo de migración: Si la aplicación utiliza varias bases de datos, cree clústeres de Aurora DSQL independientes para una separación lógica o utilice esquemas dentro de un único clúster.

Ausencia de tablas temporales

Para la gestión temporal de datos, DEBERÍA utilizar expresiones de tabla comunes (CTE) y subconsultas, que ofrecen alternativas flexibles para consultas complejas.

Alternativa: Utilice CTE con cláusulas `WITH` para conjuntos de resultados temporales o tablas normales con nombres únicos para datos específicos de la sesión.

Administración de almacenamiento automática

Aurora DSQL elimina los espacios de tabla y la administración de almacenamiento manual. El almacenamiento se escala y se optimiza automáticamente en función de sus patrones de datos.

Ventaja: No es necesario supervisar el espacio en disco, planificar la asignación de almacenamiento ni administrar las configuraciones de los espacios de tabla.

Patrones de aplicación modernos

Aurora DSQL fomenta los patrones de desarrollo de aplicaciones modernos que mejoran la capacidad de mantenimiento y el rendimiento:

Lógica de nivel de aplicación en lugar de desencadenadores de bases de datos

Para obtener una funcionalidad similar a la de un activador, implemente una lógica basada en eventos en la capa de aplicación.

Estrategia de migración: Traslade la lógica de los desencadenadores al código de la aplicación, utilice arquitecturas basadas en eventos con servicios de AWS como EventBridge o implemente registros de auditoría mediante el registro de aplicaciones.

Funciones SQL para el procesamiento de datos

Aurora DSQL admite funciones basadas en SQL, pero no lenguajes procedurales como PL/pgSQL.

Alternativa: Utilice funciones SQL para las transformaciones de datos o traslade la lógica compleja a la capa de aplicación o a funciones de AWS Lambda.

Control de simultaneidad optimista en lugar de bloqueo pesimista

Aurora DSQL utiliza el control de simultaneidad optimista (OCC), un enfoque sin bloqueos diferente a los mecanismos de bloqueo de bases de datos tradicionales. En lugar de adquirir bloqueos que afecten a otras transacciones, Aurora DSQL permite que las transacciones continúen sin bloquearse y detecta los conflictos en el momento de la confirmación. De este modo, se eliminan los interbloqueos y se impide que las transacciones lentas bloqueen otras operaciones.

Diferencia clave: Cuando se producen conflictos, Aurora DSQL devuelve un error de serialización en lugar de hacer que las transacciones esperen bloqueos. Esto requiere que las aplicaciones implementen una lógica de reintento, similar a la gestión de tiempos de espera de bloqueo en las bases de datos tradicionales, pero los conflictos se resuelven de forma inmediata en lugar de provocar esperas de bloqueo.

Patrón de diseño: Implemente lógica de transacción idempotente con mecanismos de reintento. Diseñe esquemas para minimizar la contención mediante claves principales aleatorias y la

distribución de actualizaciones en todo su intervalo de claves. Para obtener más información, consulte Control de simultaneidad en Aurora DSQL.

Relaciones e integridad referencial

Aurora DSQL admite relaciones de claves externas entre tablas, incluidas las operaciones `JOIN`. Para garantizar la integridad referencial, implemente la validación en la capa de aplicación. Si bien aplicar la integridad referencial puede resultar útil, las operaciones en cascada (como las eliminaciones en cascada) pueden provocar problemas de rendimiento inesperados; por ejemplo, eliminar un pedido con mil partidas se convierte en una transacción de mil y una filas. Por este motivo, muchos clientes evitan las restricciones de clave externa.

Patrón de diseño: Implemente comprobaciones de integridad referencial en la capa de aplicaciones, utilice patrones de coherencia final o aproveche los servicios de AWS para la validación de datos.

Simplificaciones operativas

Aurora DSQL elimina muchas de las tareas tradicionales de mantenimiento de bases de datos, lo que reduce la sobrecarga operativa:

No se requiere mantenimiento manual

Aurora DSQL administra automáticamente la optimización del almacenamiento, la recopilación de estadísticas y el ajuste del rendimiento. Los comandos de mantenimiento tradicionales como `VACUUM` son gestionados por el sistema.

Ventaja: Elimina la necesidad de ventanas de mantenimiento de base de datos, la programación del vaciado y la afinación de los parámetros del sistema.

Particionamiento y escalado automáticos

Aurora DSQL particiona y distribuye sus datos de forma automática en función de los patrones de acceso. Utilice UUID o ID generados por la aplicación para una distribución óptima.

Consejo de migración: Elimine la lógica de particionamiento manual y deje que Aurora DSQL se encargue de la distribución de datos. Utilice UUID o ID generados por la aplicación para una distribución óptima. Si su aplicación requiere identificadores secuenciales, consulte Secuencias y columnas de identidad.

Migración agéntica con herramientas de IA

Los agentes de codificación de IA pueden acelerar su migración a Aurora DSQL mediante el análisis de esquemas, la transformación de código y la ejecución de migraciones DDL con comprobaciones de seguridad integradas.

Uso de Kiro para la migración

Los agentes de codificación como Kiro pueden ayudarle a analizar y migrar su código PostgreSQL a Aurora DSQL:

- Análisis del esquema: Cargue sus archivos de esquema existentes y pida a Kiro que identifique los posibles problemas de compatibilidad y que sugiera alternativas
- Transformación de código: Proporcione el código de su aplicación y pida a Kiro que ayude a refactorizar la lógica de desencadenador, reemplazar secuencias con UUID o modificar patrones de transacción
- Planificación de la migración: Pida a Kiro que cree un plan de migración paso a paso basado en la arquitectura específica de su aplicación
- Migraciones DDL: ejecute modificaciones del esquema utilizando el patrón de recreación de tablas con comprobaciones de seguridad integradas y verificación de usuarios.

Ejemplos de peticiones:

```
"Analyze this PostgreSQL schema for DSQL compatibility and suggest alternatives for any
 unsupported features"

"Help me refactor this trigger function into application-level logic for DSQL
 migration"

"Create a migration checklist for moving my Django application from PostgreSQL to DSQL"

"Drop the legacy_status column from the orders table"

"Change the price column from VARCHAR to DECIMAL in the products table"
```

Migración DDL con recreación de tablas.

Al utilizar agentes de IA con el servidor MCP de Aurora DSQL, ciertas operaciones ALTER TABLE utilizan un patrón de recreación de tablas que migra sus datos de forma segura. El agente gestiona la complejidad y, al mismo tiempo, le mantiene informado en cada paso.

Las siguientes operaciones utilizan el patrón de recreación de tablas:

Operación	Enfoque
`DROP COLUMN`	Excluir columna de la nueva tabla
`ALTER COLUMN TYPE`	Tipo de datos de conversión durante la migración
`ALTER COLUMN SET/DROP NOT NULL`	Cambiar restricción en la definición de la nueva tabla
`ALTER COLUMN SET/DROP DEFAULT`	Definir el valor predeterminado en la definición de la nueva tabla
`ADD/DROP CONSTRAINT`	Incluir o eliminar una restricción en la nueva tabla
`MODIFY PRIMARY KEY`	Definir una nueva clave principal con validación de unicidad
Dividir/fusionar columnas	Usar SPLIT_PART, SUBSTRING o CONCAT

Las siguientes operaciones ALTER TABLE se admiten directamente sin necesidad de recrear la tabla:

- `ALTER TABLE ... RENAME COLUMN`: cambiar el nombre de una columna
- `ALTER TABLE ... RENAME TO`: cambiar el nombre de una tabla
- `ALTER TABLE ... ADD COLUMN`: añadir una nueva columna

Características de seguridad: al ejecutar migraciones DDL, los agentes de IA presentan el plan de migración, verifican la compatibilidad de los datos, confirman el recuento de filas y solicitan una aprobación explícita antes de realizar cualquier operación destructiva, como DROP TABLE.

Migraciones por lotes: para tablas que superan las 3000 filas, el agente divide automáticamente la migración en lotes de entre 500 y 1000 filas para mantenerse dentro de los límites de transacción.

Servidor MCP de Aurora DSQL

El servidor de protocolo de contexto para modelos (MCP) de Aurora DSQL permite a los asistentes de IA conectarse directamente a su clúster de Aurora DSQL y consultar la documentación de Aurora DSQL. Esto permite a la IA:

- Analizar el esquema existente y sugerir cambios de migración
- Ejecutar migraciones DDL con el patrón de recreación de tablas
- Probar consultas y comprobar la compatibilidad durante la migración
- Proporcionar directrices precisas y actualizadas basadas en la documentación más reciente de Aurora DSQL

Para utilizar el servidor MCP de Aurora DSQL con asistentes de IA, consulte las instrucciones de configuración para el servidor MCP de Aurora DSQL.

Consideraciones de Aurora DSQL para la compatibilidad de PostgreSQL

Aurora DSQL presenta diferencias en la compatibilidad de las características con respecto a la instancia de PostgreSQL autoadministrada, las cuales permiten su arquitectura distribuida, funcionamiento sin servidor y escalado automático. La mayoría de las aplicaciones funcionan dentro de estas diferencias sin hacer ninguna modificación.

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` por clúster. Para una separación lógica, cree clústeres de Aurora DSQL independientes o utilice esquemas dentro de un único clúster.
- La base de datos `postgres` utiliza la codificación de caracteres UTF-8, que ofrece una amplia compatibilidad con caracteres internacionales.
- La base de datos usa solo la intercalación de `C`.
- Aurora DSQL utiliza `UTC` como la zona horaria del sistema. Postgres almacena internamente todas las fechas y horas que tienen en cuenta las zonas horarias en UTC. Puede configurar el parámetro de configuración `TimeZone` para convertir la forma en que se muestra en el cliente y que sirva como valor predeterminado para las entradas del cliente que el servidor utilizará para convertir internamente a UTC.
- El nivel de aislamiento de las transacciones se fija en PostgreSQL `Repeatable Read`.
- Las transacciones tienen las siguientes restricciones:
  - Las operaciones DDL y DML requieren transacciones separadas.
  - 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.
- Aurora DSQL administra los permisos a través de concesiones a nivel de esquema. Los usuarios administradores crean esquemas utilizando `CREATE SCHEMA` y conceden acceso utilizando `GRANT USAGE ON SCHEMA`. Los usuarios administradores gestionan objetos en el esquema público, mientras que los usuarios no administradores crean objetos en esquemas creados por usuarios para establecer límites claros de propiedad. Para obtener más información, consulte Autorización de roles de base de datos para utilizar SQL en la base de datos.

¿Necesita ayuda con la migración?

Si encuentra características que son fundamentales para la migración pero que actualmente no son compatibles con Aurora DSQL, consulte Aportación de comentarios sobre Amazon Aurora DSQL para obtener información sobre cómo compartir comentarios con AWS.

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.

Secuencias y columnas de identidad

Las secuencias y las columnas de identidad generan valores enteros y son útiles cuando se necesitan identificadores compactos o legibles por humanos. Estos valores implican el comportamiento de asignación y almacenamiento en caché descrito en la documentación `CREATE SEQUENCE`.

Temas

- Funciones de manipulación de secuencias
- Columnas de identidad
- Trabajar con secuencias y columnas de identidad

Funciones de manipulación de secuencias

En esta sección se describen las funciones para operar con objetos de secuencia, también denominados generadores de secuencias o simplemente secuencias. Los objetos de secuencia son tablas especiales de una sola fila creadas con `CREATE SEQUENCE`. Los objetos de secuencia se utilizan habitualmente para generar identificadores únicos para las filas de una tabla. Las funciones de secuencia proporcionan métodos sencillos y seguros para varios usuarios que permiten obtener valores de secuencia sucesivos a partir de objetos de secuencia.

> **Important**
>
> Al utilizar secuencias, se debe tener muy en cuenta el valor de la caché. Para obtener más información, consulte el aviso Importante de la página `CREATE SEQUENCE`.
> Para obtener orientación sobre cómo utilizar mejor las secuencias basadas en patrones de carga de trabajo, consulte Trabajar con secuencias y columnas de identidad.

Función	Descripción
`nextval ( regclass ) # bigint`	Hace avanzar el objeto secuencial a su siguiente valor y devuelve ese valor. Esto se hace de forma atómica: incluso si se ejecutan varias sesiones `nextval` simultáne amente, cada una recibirá de forma segura un valor de secuencia distinto. Si el objeto de secuencia se ha creado con los parámetros predeterminados, las llamadas de `nextval` sucesivas devolverán valores crecientes que comiencen por 1. Se pueden obtener otros comportam ientos utilizando los parámetros adecuados en el comando `CREATE SEQUENCE`. Esta función requiere un privilegio USAGE o UPDATE en la secuencia.
`setval ( regclass, bigint [, boolean ] ) # bigint`	Establece el valor actual del objeto de secuencia y, opcionalmente, su indicador `is_called` . La forma de dos parámetros establece el campo `last_value` de la secuencia en el valor especificado y establece su campo `is_called` en `true`, lo que significa que el siguiente `nextval` hará avanzar la secuencia antes de devolver un valor. El valor que indicará `currval` también se establece en el valor especificado. En el formulario de tres parámetros, `is_called` se puede establece r en `true` o `false`. `true` tiene el mismo efecto que el formulario de dos parámetros. Si se establece en `false`, el `nextval` siguiente devolverá exactamente el valor especificado y el avance de la secuencia comenzará

Función	Descripción

	con el `nextval` siguiente. Además, el valor indicado por `currval` no cambia aquí. Por ejemplo:
	
	`SELECT setval('myseq', 42);  -- Next nextval will return 43`
	`SELECT setval('myseq', 42, true);  -- Same as above`
	`SELECT setval('myseq', 42, false);  -- Next nextval will return 42`
	
	El resultado devuelto por `setval` es solo el valor de su segundo argumento. Esta función requiere el privilegio `UPDATE` en la secuencia.

`currval ( regclass ) # bigint`	Devuelve el valor obtenido más recientemente por `nextval` para esta secuencia en la sesión actual. (Se informa de un error si `nextval` nunca se ha llamado para esta secuencia en esta sesión). Dado que esto devuelve un valor local de la sesión, ofrece una respuesta predecible sobre si otras sesiones han ejecutado `nextval` o no, ya que la sesión actual lo ha hecho. Esta función requiere un privilegio `USAGE` o `SELECT` en la secuencia.
`lastval () # bigint`	Devuelve el valor devuelto más recientemente por `nextval` en la transacción actual. Esta función es idéntica a `currval`, excepto que, en lugar de tomar el nombre de la secuencia como argumento, se refiere a la secuencia a la que `nextval` se haya aplicado más recientemente en la transacción actual. Es un error llamar a `lastval` si aún no se ha llamado a `nextval` en la transacción actual. Esta función requiere un privilegio `USAGE` o `SELECT` en la última secuencia utilizada.

> **Warning**
>
> El valor obtenido por `nextval` no se recupera para su reutilización si la transacción de llamada se cancela posteriormente. Esto significa que las interrupciones de transacciones o los fallos de la base de datos pueden provocar brechas en la secuencia de valores asignados. Eso también puede ocurrir sin que se cancele la transacción. Por ejemplo, un `INSERT` con una cláusula `ON CONFLICT` calculará la tupla que se va a insertar, incluida la realización de las llamadas de `nextval` necesarias, antes de detectar cualquier conflicto que pueda provocar que siga la regla `ON CONFLICT`. Por lo tanto, los objetos de secuencia de Aurora DSQL no se pueden utilizar para obtener secuencias "sin brechas".
> Del mismo modo, los cambios de estado de secuencia realizados por `setval` son inmediatamente visibles para otras transacciones y no se deshacen si la transacción que realiza la llamada se revierte.

La secuencia sobre la que va a operar una función de secuencia se especifica mediante un argumento `regclass`, que es simplemente el OID de la secuencia en el catálogo del sistema `pg_class`. Sin embargo, no tiene que buscar el OID manualmente, ya que el convertidor de entrada del tipo de datos `regclass` hará el trabajo por usted. Consulte la documentación de PostgreSQL sobre los tipos de identificadores de objetos para obtener más información.

Columnas de identidad

> **Important**
>
> Cuando se utilizan columnas de identidad, se debe considerar con cuidado el valor de la caché. Para obtener más información, consulte el aviso Importante de la página `CREATE SEQUENCE`.
> Para obtener orientación sobre la mejor manera de utilizar las columnas de identidad en función de los patrones de carga de trabajo, consulte Trabajar con secuencias y columnas de identidad.

Una columna de identidad es una columna especial que se genera automáticamente a partir de una secuencia implícita. Se puede usar para generar valores clave. Para crear una columna de identidad, utilice la cláusula `GENERATED ... AS IDENTITY` en `CREATE TABLE`, por ejemplo:

```
CREATE TABLE people (
```

```
    id bigint GENERATED ALWAYS AS IDENTITY (CACHE 70000),
    ...
);
```

o, de forma alternativa:

```
CREATE TABLE people (
    id bigint GENERATED BY DEFAULT AS IDENTITY (CACHE 70000),
    ...
);
```

Consulte CREATE TABLE para obtener más detalles.

Si se ejecuta un comando INSERT en la tabla con la columna de identidad y no se especifica ningún valor explícito para la columna de identidad, se inserta un valor generado por la secuencia implícita. Por ejemplo, con las definiciones anteriores y suponiendo que haya columnas adicionales adecuadas, escribir:

```
INSERT INTO people (name, address) VALUES ('A', 'foo');
INSERT INTO people (name, address) VALUES ('B', 'bar');
```

generaría valores para la columna id empezando por 1 y daría como resultado los siguientes datos de la tabla:

```
 id | name | address
----+------+---------
  1 | A    | foo
  2 | B    | bar
```

Como alternativa, la clave DEFAULT se puede especificar en lugar de un valor para solicitar explícitamente el valor generado por la secuencia:

```
INSERT INTO people (id, name, address) VALUES (DEFAULT, 'C', 'baz');
```

Del mismo modo, la palabra clave DEFAULT se puede utilizar en los comandos UPDATE.

Por lo tanto, en muchos aspectos, una columna de identidad se comporta como una columna con un valor predeterminado.

Las cláusulas `ALWAYS` y `BY DEFAULT` en la definición de la columna determinan cómo se gestionan de forma explícita los valores especificados por el usuario en los comandos `INSERT` y `UPDATE`. En un comando `INSERT`, si se selecciona `ALWAYS`, solo se acepta un valor especificado por el usuario si la instrucción `INSERT` especifica `OVERRIDING SYSTEM VALUE`. Si se selecciona `BY DEFAULT`, prevalece el valor especificado por el usuario. Por lo tanto, el uso de `BY DEFAULT` da como resultado un comportamiento más similar a los valores predeterminados, donde el valor predeterminado puede ser anulado por un valor explícito, mientras que `ALWAYS` proporciona una mayor protección contra la inserción accidental de un valor explícito.

El tipo de datos de una columna de identidad debe ser uno de los tipos de datos compatibles con las secuencias. (Consulte `CREATE SEQUENCE`.) Las propiedades de la secuencia asociada se pueden especificar al crear una columna de identidad (consulte `CREATE TABLE`) o modificarse posteriormente (consulte `ALTER TABLE`).

Una columna de identidad se marca automáticamente como `NOT NULL`. Sin embargo, una columna de identidad no garantiza la unicidad. (Una secuencia normalmente devuelve valores únicos, pero una secuencia podría restablecerse, o los valores podrían insertarse manualmente en la columna de identidad, como se ha comentado anteriormente). La unicidad tendría que imponerse mediante una restricción `UNIQUE` o `PRIMARY KEY`.

Trabajar con secuencias y columnas de identidad

Esta sección le ayudará a comprender cómo utilizar mejor las secuencias y las columnas de identidad en función de los patrones de carga de trabajo.

> **Important**
>
> Consulte el aviso Importante de la página `CREATE SEQUENCE` para obtener más información sobre el comportamiento de asignación y almacenamiento en caché.

Elección de los tipos de identificadores

Amazon Aurora DSQL admite tanto identificadores basados en UUID como valores enteros generados mediante secuencias o columnas de identidad. Estas opciones difieren en la forma en que se asignan los valores y en la forma en que se escalan bajo carga.

Los valores UUID se pueden generar sin coordinación y son adecuados para cargas de trabajo en las que se crean identificadores con frecuencia o en muchas sesiones. Dado que Amazon

Aurora DSQL está diseñado para funcionar de forma distribuida, a menudo resulta beneficioso evitar la coordinación. Por este motivo, se recomienda utilizar UUID como tipo de identificador predeterminado, especialmente para claves principales en cargas de trabajo en las que la escalabilidad es importante y no se requiere un orden estricto de los identificadores.

Las secuencias y las columnas de identidad generan valores enteros compactos que resultan muy prácticos para los identificadores legibles por el ser humano, la generación de informes y las interfaces externas. Cuando se prefieran los identificadores numéricos por motivos de usabilidad o integración, considere la posibilidad de utilizar una columna de secuencia o identidad en combinación con identificadores basados en UUID. Cuando se requieren secuencias de números enteros o valores de identidad, elegir un tamaño de caché adecuado se convierte en una parte importante del diseño de la carga de trabajo. Consulte la siguiente sección para obtener instrucciones sobre cómo elegir un tamaño de caché.

Elección de un tamaño de caché

Seleccionar un valor de caché adecuado es una parte importante del uso eficaz de secuencias y columnas de identidad. La configuración de la caché determina cómo se comporta la asignación de identificadores bajo carga, lo que influye tanto en el rendimiento del sistema como en la precisión con la que los valores reflejan el orden de asignación.

Un tamaño de caché mayor de **`CACHE >= 65536`** es adecuado cuando:

- Los identificadores se generan a alta frecuencia.
- Muchas sesiones se insertan de forma simultánea.
- La carga de trabajo puede tolerar las brechas y los efectos de ordenamiento visibles.

Por ejemplo, las cargas de trabajo de ingestión de eventos de gran volumen (como IoT o telemetría), así como los identificadores operativos como los ID de ejecución de trabajos, las referencias de casos de soporte o los números de pedido internos, suelen beneficiarse de tamaños de caché más grandes, donde los identificadores se generan con frecuencia y no se requiere un orden estricto.

Un tamaño de caché de 1 se alinea mejor cuando:

- Las tasas de asignación son relativamente bajas.
- Se espera que los identificadores sigan el orden de asignación más de cerca con el tiempo.
- Minimizar las brechas es más importante que el máximo rendimiento.

Las cargas de trabajo como la asignación de números de cuenta o de referencia, en las que los identificadores se generan con menos frecuencia y es deseable un orden más cercano, se ajustan mejor a un tamaño de caché de 1.

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 ]
```

Parameters

`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 where job_id = 'wqhu6ewifze5xitg3umt24h5ua';
```

Aurora DSQL devuelve una respuesta similar a la siguiente.

```
           job_id           |  status   | details |  job_type   | class_id | object_id
 |    object_name    |       start_time       |      update_time
----------------------------+-----------+---------+-------------+----------+-----------
+------------------+-----------------------+-----------------------
 wqhu6ewifze5xitg3umt24h5ua | completed |         | INDEX_BUILD |     1259 |     26433
 | public.nt2_c1_idx | 2025-09-25 22:07:31+00 | 2025-09-25 22:07:46+00
```

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

Estado	Descripción
`submitted`	La tarea se ha enviado, pero Aurora DSQL aún no ha empezado a procesarla.
`processing`	Aurora DSQL está procesando la tarea.
`failed`	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`.
`completed`	Aurora DSQL ha completado la tarea correctamente.

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 secciones siguientes para obtener información sobre las tablas y los catálogos del sistema compatibles con Aurora DSQL, así como sobre consultas útiles para obtener información sobre el sistema, como la versión.

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

Nombre	Descripción
`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
`pg_am`	Sí
`pg_amop`	No
`pg_amproc`	No
`pg_attrdef`	Sí
`pg_attribute`	Sí
`pg_authid`	No (utilice `pg_roles`)

Nombre	Aplicable a Aurora DSQL
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
pg_foreign_data_wrapper	No
pg_foreign_server	No
pg_foreign_table	No
pg_index	Sí
pg_inherits	Sí
pg_init_privs	No

Nombre	Aplicable a Aurora DSQL
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
pg_range	Sí
pg_replication_origin	No
pg_rewrite	No
pg_seclabel	No
pg_sequence	No
pg_shdepend	Sí

Nombre	Aplicable a Aurora DSQL
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
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
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

Nombre	Aplicable a Aurora DSQL
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
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

Nombre	Aplicable a Aurora DSQL
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
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

Nombre	Aplicable a Aurora DSQL
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
pg_statio_user_indexes	No

La vista 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;
```

Aurora DSQL devuelve una respuesta similar a la siguiente.

```
          job_id           |  status   | details |  job_type   | class_id | object_id
 |    object_name    |       start_time       |      update_time
---------------------------+-----------+---------+-------------+----------+-----------
+------------------+------------------------+------------------------
 wqhu6ewifze5xitg3umt24h5ua | completed |         | INDEX_BUILD |     1259 |     26433
 | public.nt2_c1_idx | 2025-09-25 22:07:31+00 | 2025-09-25 22:07:46+00
 kkngzf33dndl3daacxehpx5eba | completed |         | ANALYZE     |     1259 |     26419
 | public.nt         | 2025-09-25 21:57:05+00 | 2025-09-25 21:57:27+00
 fyopxjb6ovdn7po6lrkj63cyea | completed |         | DROP        |     1259 |     26422
 |                   | 2025-09-25 22:05:57+00 | 2025-09-25 22:06:03+00
```

En la siguiente tabla se describen las columnas de la vista `sys.jobs`.

Columnas de la vista sys.jobs

Columna	Tipo	Descripción
`job_id`	`text`	Un UUID de base 32 que representa el trabajo.
`status`	`text`	El estado actual del trabajo. Los valores posibles son `submitted` , `processing` , `completed` y `failed`. Para obtener más información, consulte Valores de estado de sys.jobs.
`details`	`text`	Cualquier dato relevante sobre el trabajo. Si el trabajo falla, se proporciona un motivo detallado.
`job_type`	`text`	El tipo de trabajo asíncrono. Los valores posibles son: `INDEX_BUILD` : una creación de índices asíncrona. `ANALYZE`: un trabajo de análisis automático enviado por el sistema. `DROP`: elimina los datos físicos tras una operación de `DROP TABLE` o `DROP INDEX`.
`class_id`	`oid`	El OID de la tabla de catálogo que contiene el objeto.
`object_id`	`oid`	El OID del objeto.
`object_name`	`text`	El nombre completo del objeto. Los trabajos de `DROP` no pueden hacer referencia a objetos que

Columna	Tipo	Descripción
		ya se han eliminado. Si el objeto al que se hace referencia ya se ha eliminado, el valor `object_name` puede ser NULL.
`start_time`	`timestamp with time zone`	La marca de tiempo en la que se envió el trabajo.
`update_time`	`timestamp with time zone`	La marca de tiempo en la que se actualizó por última vez la fila del trabajo.

Valores de estado de sys.jobs

Estado	Descripción
`submitted`	La tarea se ha enviado, pero Aurora DSQL aún no ha empezado a procesarla.
`processing`	Aurora DSQL está procesando la tarea.
`failed`	La tarea ha fallado. Consulte la columna `details` para obtener más información.
`completed`	Aurora DSQL ha completado la tarea correctamente.

La vista 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;
```

Consultas de metadatos de sistemas útiles

Utilice estas consultas para obtener estadísticas de tabla y metadatos de sistema sin realizar operaciones costosas como escanear tablas completas.

Obtención del recuento estimado de filas de una tabla

Para obtener el recuento aproximado de filas en una tabla sin realizar un escaneo de tabla completo, use la consulta siguiente:

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

El comando devuelve un resultado similar al siguiente:

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

Este enfoque es más eficaz que en el caso de `SELECT COUNT(*)` para las tablas grandes en Aurora DSQL.

Obtención de la versión principal actual de Aurora DSQL

Para obtener la versión principal actual del clúster de Aurora DSQL, utilice la siguiente consulta:

```
SELECT * FROM sys.dsql_major_version();
```

El comando devuelve un resultado similar al siguiente:

```
 dsql_major_version
--------------------
                  1
```

Esto devuelve la versión principal en la que se encuentra la conexión SQL en Aurora DSQL.

Obtención de la versión actual de la plataforma

Para obtener la versión actual de PostgreSQL del clúster de Aurora DSQL, utilice la siguiente consulta:

```
SHOW server_version;
```

El comando devuelve un resultado similar al siguiente:

```
 server_version
----------------
 16.13
```

Esto devuelve la versión en la que se encuentra la conexión SQL en Aurora DSQL.

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.

Trabajo con planes EXPLAIN de Aurora DSQL

Aurora DSQL utiliza una estructura de plan EXPLAIN similar a la de PostgreSQL, pero con adiciones clave que reflejan su arquitectura distribuida y su modelo de ejecución.

En esta documentación, proporcionaremos información general de los planes EXPLAIN de Aurora DSQL y destacaremos las similitudes y diferencias en comparación con PostgreSQL. Cubriremos los distintos tipos de operaciones de escaneo disponibles en Aurora DSQL y lo ayudaremos a comprender el costo de ejecutar las consultas.

Planes EXPLAIN de PostgreSQL frente a Aurora DSQL

Aurora SQL se basa en la base de datos de PostgreSQL y comparte la mayoría de las estructuras planificadas con PostgreSQL, pero presenta diferencias arquitectónicas clave que afectan a la ejecución y optimización de las consultas:

Característica	PostgreSQL	Aurora DSQL
Almacenamiento de datos	Almacenamiento de montón	Sin montón, todas las filas están indexadas por un identificador único
Clave principal	El índice de clave principal está separado de los datos de la tabla	El índice de clave principal es la tabla con todas las columnas adicionales como columnas INCLUDE
Índices secundarios	Índices secundarios estándar	Funciona igual que PostgreSQL, con la posibilidad de incluir columnas no clave
Capacidades de filtrado	Condición de índice, filtro de montón	Condición de índice, filtro de almacenamiento, filtro de procesador de consultas

Característica	PostgreSQL	Aurora DSQL
Tipos de exámenes	Escaneo secuencial, escaneo de índice, escaneo solo de índice	Escaneo completo, escaneo solo de índice, escaneo de índice
Ejecución de consulta	Local en la base de datos	Distribuido (el procesamiento y el almacenamiento son independi entes)

Aurora DSQL almacena los datos de la tabla directamente en orden de clave principal en lugar de en un montón independiente. Cada fila se identifica mediante una clave única, normalmente la clave principal, que permite a la base de datos optimizar las búsquedas de manera más eficiente. La diferencia arquitectónica explica por qué Aurora DSQL suele utilizar escaneos solo de índice en los casos en que PostgreSQL puede elegir un escaneo secuencial.

Otra distinción clave es que Aurora DSQL separa la informática del almacenamiento, lo que permite aplicar filtros en una fase más temprana de la ruta de ejecución para reducir el movimiento de datos y mejorar el rendimiento.

Para obtener más información sobre el uso de los planes EXPLAIN con PostgreSQL, consulte la documentación EXPLAIN de PostgreSQL.

Elementos clave de los planes EXPLAIN de Aurora DSQL

Los planes EXPLAIN de Aurora DSQL proporcionan información detallada sobre cómo se ejecutan las consultas, incluido dónde se produce el filtrado y qué columnas se recuperan del almacenamiento. La comprensión de este resultado ayuda a optimizar el rendimiento de las consultas.

Condición de índice

Condiciones utilizadas para navegar por el índice. El filtrado más eficiente que reduce los datos escaneados. En Aurora DSQL, las condiciones de índice se pueden aplicar en varios niveles del plan de ejecución.

Proyecciones

Columnas recuperadas del almacenamiento. Menos proyecciones significan un mejor rendimiento.

Filtro de almacenamiento

Condiciones aplicadas por almacenamiento. Más eficiente que los filtros del procesador de consultas.

Filtro de procesador de consultas

Condiciones aplicadas por procesador de consultas. Requiere transferir todos los datos antes de filtrarlos, lo que se traduce en una mayor sobrecarga de procesamiento y movimiento de datos.

Filtros en Aurora DSQL

Aurora DSQL separa la informática del almacenamiento, lo que significa que el punto en el que se aplican los filtros durante la ejecución de la consulta tiene un impacto significativo en el rendimiento. Los filtros que se aplican antes de transferir grandes volúmenes de datos reducen la latencia y mejoran la eficiencia. Cuanto antes se aplique un filtro, menos datos deberán procesarse, moverse y escanearse, lo que se traducirá en consultas más rápidas.

Aurora DSQL puede aplicar filtros en varias etapas de la ruta de consulta. la comprensión de estas etapas es clave para interpretar los planes de consultas y optimizar el rendimiento.

Nivel	Tipo de filtro	Descripción
1	Condición de índice	Se aplica al escanear el índice. Limita la cantidad de datos que se leen del almacenamiento y reduce los datos que se envían a la capa de procesami ento.
2	Filtro de almacenam iento	Se aplica después de leer los datos del almacenam iento, pero antes de enviarlos a informática. Un ejemplo de esto es un filtro en una columna de inclusión de un índice. Reduce la transferencia de datos, pero no la cantidad leída.
3	Filtro de procesador de consultas	Se aplica después de que los datos lleguen a la capa de informática. Todos los datos se deben transferir primero, lo que aumenta la latencia y el costo. Actualmente, Aurora DSQL no puede realizar todas las operaciones de filtrado y proyección en el

Nivel	Tipo de filtro	Descripción
		almacenamiento, por lo que es posible que algunas consultas se vean obligadas a recurrir a este tipo de filtrado.

Lectura de los planes EXPLAIN de Aurora DSQL

Comprender cómo leer los planes EXPLAIN es clave para optimizar el rendimiento de las consultas. En esta sección, analizaremos ejemplos reales de planes de consultas de Aurora DSQL, mostraremos cómo se comportan los diferentes tipos de escaneo, explicaremos dónde se aplican los filtros y destacaremos las oportunidades de optimización.

Tablas de muestra utilizadas en estos ejemplos

Los ejemplos siguientes hacen referencia a dos tablas: `transaction` y `account`.

La tabla `transaction` o dispone de clave principal, lo que provoca que Aurora DSQL escanee toda la tabla al consultarla.

La tabla `account` tiene un índice en `customer_id`. Este índice incluye `balance` y `status` como columnas de cobertura, lo que permite responder a determinadas consultas directamente desde el índice sin necesidad de leer la tabla base. Sin embargo, el índice no incluye `created_at`, por lo que las consultas que hacen referencia a esta columna requieren un acceso adicional a la tabla.

```
CREATE TABLE transaction (
    account_id uuid,
    transaction_date timestamp,
    description text
);

CREATE TABLE account (
    customer_id uuid,
    balance numeric,
    status varchar,
    created_at timestamp
);

CREATE INDEX ASYNC idx1 ON account (customer_id) INCLUDE (balance, status);
```

Ejemplo de escaneo completo

Aurora DSQL tiene escaneos secuenciales, que funcionan de manera idéntica a PostgreSQL, así como escaneos completos. La única diferencia entre estos dos es que los escaneos completos pueden utilizar filtros adicionales en el almacenamiento. Debido a esto, casi siempre se selecciona por encima de los escaneos secuenciales. Debido a la similitud, solo trataremos ejemplos de los escaneos completos más interesantes.

Los escaneos completos se utilizarán principalmente en tablas sin clave principal. Como las claves principales de Aurora DSQL son índices de cobertura completa de forma predeterminada, lo más probable es que Aurora DSQL utilice escaneos solo indexados en la clave principal en muchas situaciones en las que PostgreSQL utilizaría un escaneo secuencial. Como ocurre con la mayoría de las demás bases de datos, una tabla sin índices se escalará mal.

```
EXPLAIN SELECT account_id FROM transaction WHERE transaction_date > '2025-01-01' AND
 description LIKE '%external%';
```

```
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------
 Full Scan (btree-table) on transaction  (cost=125100.05..177933.38 rows=33333
 width=16)
   Filter: (description ~~ '%external%'::text)
   -> Storage Scan on transaction (cost=12510.05..17793.38 rows=66666 width=16)
        Projections: account_id, description
        Filters: (transaction_date > '2025-01-01 00:00:00'::timestamp without time
 zone)
        -> B-Tree Scan on transaction (cost=12510.05..17793.38 rows=100000 width=30)
```

Este plan muestra dos filtros aplicados en diferentes etapas. La condición `transaction_date > '2025-01-01'` se aplica en la capa de almacenamiento, lo que reduce la cantidad de datos que se devuelven. La condición `description LIKE '%external%'` se aplica más adelante en el procesador de consultas, una vez transferidos los datos, lo que hace que sea menos eficiente. Insertar filtros más selectivos en las capas de almacenamiento o índice generalmente mejora el rendimiento.

Ejemplo de escaneo solo de índices

Los escaneos solo de índices son los tipos de escaneo más óptimos en Aurora DSQL, ya que producen el menor número de viajes de ida y vuelta a la capa de almacenamiento y son los que más

filtran. Sin embargo, el hecho de que vea escaneo solo de índice no significa que tenga el mejor plan. Debido a los diferentes niveles de filtrado que se pueden producir, es esencial seguir prestando atención a los diferentes lugares en los que se puede producir el filtrado.

```
EXPLAIN SELECT balance FROM account
WHERE customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'
AND balance > 100
AND status = 'pending';
```

```
                                  QUERY PLAN
------------------------------------------------------------------------------
 Index Only Scan using idx1 on account  (cost=725.05..1025.08 rows=8 width=18)
   Index Cond: (customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'::uuid)
   Filter: (balance > '100'::numeric)
   -> Storage Scan on idx1 (cost=12510.05..17793.38 rows=9 width=16)
        Projections: balance
        Filters: ((status)::text = 'pending'::text)
        -> B-Tree Scan on idx1 (cost=12510.05..17793.38 rows=10 width=30)
            Index Cond: (customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'::uuid)
```

En este plan, la condición de indexación (`customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'`), se evalúa primero durante el escaneo del índice, que es la etapa más eficiente porque limita la cantidad de datos que se leen del almacenamiento. El filtro de almacenamiento, `status = 'pending'`, se aplica después de leer los datos, pero antes de enviarlos a la capa de procesamiento, lo que reduce la cantidad de datos transferidos. Por último, el filtro del procesador de consultas, `balance > 100`, se ejecuta en último lugar, después de mover los datos, por lo que es el menos eficiente. De estas, la condición de índice es la que ofrece el mejor rendimiento, ya que controla directamente la cantidad de datos que se escanean.

Ejemplo de escaneo de índices

Los escaneos de índices son similares a los escaneos solo de índices, excepto que tienen el paso adicional de tener que llamar a la tabla base. Como Aurora DSQL puede especificar filtros de almacenamiento, puede hacerlo tanto en la llamada de índice como en la llamada de búsqueda.

Para que quede claro, Aurora DSQL presenta el plan como dos nodos. De esta forma, puede ver claramente en qué medida puede ayudar agregar una columna de inclusión en términos de filas devueltas desde el almacenamiento.

```
EXPLAIN SELECT balance FROM account
WHERE customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'
AND balance > 100
AND status = 'pending'
AND created_at > '2025-01-01';
```

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Index Scan using idx1 on account  (cost=728.18..1132.20 rows=3 width=18)
   Filter: (balance > '100'::numeric)
   Index Cond: (customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'::uuid)
   -> Storage Scan on idx1 (cost=12510.05..17793.38 rows=8 width=16)
        Projections: balance
        Filters: ((status)::text = 'pending'::text)
        -> B-Tree Scan on account (cost=12510.05..17793.38 rows=10 width=30)
            Index Cond: (customer_id = '4b18a761-5870-4d7c-95ce-0a48eca3fceb'::uuid)
   -> Storage Lookup on account (cost=12510.05..17793.38 rows=4 width=16)
        Filters: (created_at > '2025-01-01 00:00:00'::timestamp without time zone)
        -> B-Tree Lookup on transaction (cost=12510.05..17793.38 rows=8 width=30)
```

Este plan muestra cómo se filtra en varias etapas:

- La condición de indexación en `customer_id` filtra los datos de forma temprana.
- El filtro de almacenamiento en `status` reduce aún más los resultados antes de que se envíen a informática.
- El filtro del procesador de consultas en `balance` se aplica más adelante, después de la transferencia.
- El filtro de búsqueda en `created_at` se evalúa al buscar columnas adicionales de la tabla base.

Agregar columnas de uso frecuente como campos de `INCLUDE` suele eliminar esta búsqueda y mejorar el rendimiento.

Prácticas recomendadas

- Alinee los filtros con las columnas indexadas para acelerar el filtrado.
- Utilice las columnas INCLUDE para permitir escaneos solo de índices y evitar búsquedas.
- Valide las estimaciones de filas al investigar los problemas de rendimiento. Aurora DSQL administra las estadísticas automáticamente ejecutando `ANALYZE` en segundo plano en función de

las tasas de cambio de los datos. Si las estimaciones parecen inexactas, puede ejecutar `ANALYZE` manualmente para actualizar las estadísticas inmediatamente.

- Evite las consultas no indexadas en tablas grandes para evitar costosos escaneos completos.

Descripción de las DPU en EXPLAIN ANALYZE

Aurora DSQL proporciona información sobre la unidad de procesamiento distribuido (DPU) por instrucción en la producción del plan de `EXPLAIN ANALYZE VERBOSE`, lo que le brinda una mayor visibilidad del costo de las consultas durante el desarrollo. Esta sección explica qué son las DPU y cómo interpretarlas en el resultado de `EXPLAIN ANALYZE VERBOSE`.

¿Qué es una DPU?

Una unidad de procesamiento distribuido (DPU) es la medida normalizada del trabajo realizado por Aurora DSQL. Está compuesta por:

- ComputeDPU: tiempo dedicado a ejecutar consultas SQL
- ReadDPU: recursos utilizados para leer datos del almacenamiento
- WriteDPU: recursos que se utilizan para escribir datos en el almacenamiento
- MultiRegionWriteDPU: recursos que se utilizan para replicar las escrituras en clústeres interconectados en configuraciones multirregionales.

Uso de la DPU en EXPLAIN ANALYZE VERBOSE

Aurora DSQL amplía `EXPLAIN ANALYZE VERBOSE` para incluir una estimación del uso de la DPU por instrucción al final del resultado. Esto proporciona una visibilidad inmediata del costo de las consultas, lo que le ayuda a identificar los factores de costo de la carga de trabajo, ajustar el rendimiento de las consultas y pronosticar mejor el uso de los recursos.

Los siguientes ejemplos muestran cómo interpretar las estimaciones de la DPU por instrucción incluidas en el resultado de EXPLAIN ANALYZE VERBOSE.

Ejemplo 1: consulta SELECT

```
EXPLAIN ANALYZE VERBOSE SELECT * FROM test_table;
```

```
QUERY PLAN
```

```
-----------------------------------------------------
Index Only Scan using test_table_pkey on public.test_table  (cost=125100.05..171100.05
 rows=1000000 width=36) (actual time=2.973..4.482 rows=120 loops=1)
  Output: id, context
  -> Storage Scan on test_table_pkey (cost=125100.05..171100.05 rows=1000000 width=36)
 (actual rows=120 loops=1)
       Projections: id, context
       -> B-Tree Scan on test_table_pkey (cost=125100.05..171100.05 rows=1000000
 width=36) (actual rows=120 loops=1)
Query Identifier: qymgw1m77maoe
Planning Time: 11.415 ms
Execution Time: 4.528 ms
Statement DPU Estimate:
  Compute: 0.01607 DPU
  Read: 0.04312 DPU
  Write: 0.00000 DPU
  Total: 0.05919 DPU
```

En este ejemplo, la instrucción SELECT realiza un análisis solo de índices, por lo que la mayor parte del costo proviene de la DPU de lectura (0.04312), que representa los datos recuperados del almacenamiento y Compute DPU (0.01607), que refleja los recursos informáticos utilizados para procesar y devolver los resultados. No hay ninguna DPU de escritura, ya que la consulta no modifica los datos. La DPU total (0.05919) es la suma de Informática + Lectura + Escritura.

Ejemplo 2: consulta INSERT

```
EXPLAIN ANALYZE VERBOSE INSERT INTO test_table VALUES (1, 'name1'), (2, 'name2'), (3,
 'name3');
```

```
QUERY PLAN
-----------------------------------------------------
Insert on public.test_table  (cost=0.00..0.04 rows=0 width=0) (actual time=0.055..0.056
 rows=0 loops=1)
  ->  Values Scan on "*VALUES*"  (cost=0.00..0.04 rows=3 width=122) (actual
 time=0.003..0.008 rows=3 loops=1)
       Output: "*VALUES*".column1, "*VALUES*".column2
Query Identifier: jtkjkexhjotbo
Planning Time: 0.068 ms
Execution Time: 0.543 ms
Statement DPU Estimate:
  Compute: 0.01550 DPU
  Read: 0.00307 DPU (Transaction minimum: 0.00375)
```

```
  Write: 0.01875 DPU (Transaction minimum: 0.05000)
  Total: 0.03732 DPU
```

Esta instrucción realiza principalmente escrituras, por lo que la mayor parte del costo está asociado a una DPU de escritura. La DPU de informática (0.01550) representa el trabajo realizado para procesar e insertar los valores. La DPU de lectura (0.00307) refleja las lecturas menores del sistema (para búsquedas en catálogos o comprobaciones de índices).

Observe los mínimos de transacciones que se muestran junto a las DPU de lectura y escritura. Indican los costos básicos por transacción que se aplican solo cuando la operación incluye lecturas o escrituras. No significan que cada transacción incurra automáticamente en un cargo de DPU de lectura de 0.00375 o DPU de escritura de 0.05. En cambio, estos mínimos se aplican por transacción durante la agregación de costos y solo si se realizan lecturas o escrituras dentro de esa transacción. Debido a esta diferencia de alcance, es posible que las estimaciones por instrucción en `EXPLAIN ANALYZE VERBOSE` no coincidan exactamente con las métricas por transacción de las que se informa en CloudWatch o en los datos de facturación.

Uso de la información de la DPU para la optimización

Las estimaciones de la DPU por instrucción ofrecen una forma eficaz de optimizar las consultas más allá del tiempo de ejecución. Los casos de uso comunes incluyen:

- Conocimiento de los costos: comprenda lo caro que es una consulta en relación con otras.
- Optimización del esquema: compare el impacto de los índices o los cambios en el esquema tanto en el rendimiento como en la eficiencia de los recursos.
- Planificación presupuestaria: calcule el costo de la carga de trabajo en función del uso observado de la DPU.
- Comparación de consultas: evalúe los enfoques de consulta alternativos según su consumo relativo de DPU.

Interpretación de la información de la DPU

Tenga en cuenta las siguientes prácticas recomendadas al utilizar datos de DPU de `EXPLAIN ANALYZE VERBOSE`:

- Úselo de forma direccional: trate la DPU de la que se ha informado como una forma de entender el costo relativo de una consulta, en lugar de una coincidencia exacta con las métricas o los datos de facturación de CloudWatch. Se esperan diferencias porque `EXPLAIN ANALYZE`

`VERBOSE` informa del costo por instrucción, mientras que CloudWatch agrega la actividad por transacción. CloudWatch también incluye operaciones en segundo plano (como ANALYZE o compactaciones) y gastos de transacción (`BEGIN`/`COMMIT`) que `EXPLAIN ANALYZE VERBOSE` excluye intencionadamente.

- La variabilidad de la DPU entre las ejecuciones es normal en los sistemas distribuidos y no indica errores. Factores como el almacenamiento en caché, los cambios en el plan de ejecución, la simultaneidad o los cambios en la distribución de los datos pueden provocar que la misma consulta consuma recursos diferentes de una ejecución a la siguiente.
- Operaciones pequeñas por lotes: si la carga de trabajo emite muchas instrucciones pequeñas, considere agruparlas en lotes en operaciones más grandes (que no superen los 10 MB). Esto reduce los gastos generales de redondeo y produce estimaciones de costos más significativas.
- Úselo para ajustar, no para facturar: los datos de la DPU en `EXPLAIN ANALYZE VERBOSE` están diseñados para conocer los costos, ajustar las consultas y optimizar. No es una métrica apta para la facturación. Confíe siempre en las métricas de CloudWatch o en los informes de facturación mensuales para obtener datos fiables sobre costos y uso.

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

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

Configure y administre clústeres para una Región de AWS utilizando la AWS CLI o el lenguaje de programación preferido, incluido Python, C++, JavaScript, Java, Rust, Ruby, .NET y Golang. La AWS CLI proporciona un acceso rápido a través de comandos de intérprete de comandos, mientras que los kits de desarrollo de software (SDK) de AWS permiten el control programático mediante la compatibilidad con el lenguaje nativo.

Temas

Uso de los SDK de AWS

Los SDK de AWS proporcionan acceso programático a Aurora DSQL en el lenguaje de programación preferido. En las siguientes secciones, se muestra cómo realizar operaciones de clúster comunes con distintos lenguajes de programación.

Crear un clúster

En los ejemplos siguientes se muestra cómo crear un clúster de una región con diferentes lenguajes de programación.

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()
```

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;
}
```

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();
```

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);
        }
    }
}
```

Rust

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

```
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::{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 {
    let region_provider = Region::new(region);

    let config = load_defaults(BehaviorVersion::latest())
        .region(region_provider)
        .load()
        .await;

    let config = Config::new(&config);

    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")),
    ]);

    println!("Creating cluster in {region}");
    let cluster = client
        .create_cluster()
        .set_tags(Some(tags))
        .deletion_protection_enabled(true)
        .send()
        .await
        .unwrap();

    println!("Created {}", cluster.arn);

    println!("Waiting for {} to become ACTIVE", cluster.arn);
    let cluster_output = client
        .wait_until_cluster_active()
        .identifier(&cluster.identifier)
        .send()
```

```
        .await
        .unwrap();

    cluster_output
}

#[tokio::main]
async fn main() -> Result<(), Box<dyn std::error::Error>> {
    let region = "us-east-1";

    let cluster = create_cluster(region).await;

    println!("Created single region cluster:");
    println!("{:#?}", cluster);

    Ok(())
}
```

Ruby

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

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

def create_cluster(region)
  client = Aws::DSQL::Client.new(region: region)

  puts "Creating cluster in #{region}"
  cluster = client.create_cluster(
    deletion_protection_enabled: true,
    tags: {
      Name: "ruby single region cluster"
    }
  )
  puts "Created #{cluster.arn}"

  puts "Waiting for #{cluster.arn} to become ACTIVE"
  cluster = client.wait_until(:cluster_active, identifier: cluster.identifier) do |
w|
    # Wait for 5 minutes
    w.max_attempts = 30
    w.delay = 10
  end
```

```
  cluster
rescue Aws::Errors::ServiceError => e
  abort "Failed to create cluster: #{e.message}"
end

def main
  region = "us-east-1"

  cluster = create_cluster(region)

  puts "Created single region cluster:"
  pp cluster
end

main if $PROGRAM_NAME == __FILE__
```

.NET

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

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

namespace DSQLExamples.examples
{
    public class CreateCluster
    {
        /// <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 = new DefaultAWSCredentialsChain().GetCredentials();
            var clientConfig = new AmazonDSQLConfig
```

```
            {
                RegionEndpoint = region
            };
            return new AmazonDSQLClient(awsCredentials, clientConfig);
        }

        /// <summary>
        /// Create a cluster with deletion protection enabled and a name tag.
        /// </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
                };

                var cluster = await client.CreateClusterAsync(createClusterRequest);
                Console.WriteLine($"Created {cluster.Arn}");

                return cluster;
            }
        }

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

            var cluster = await Create(region);

            Console.WriteLine("Created single region cluster:");
            Console.WriteLine($"Cluster ARN: {cluster.Arn}");
        }
    }
}
```

Golang

Para crear 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/aws"
 "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)
 }

 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("failed to create cluster. %v", err)
 }

 // 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
 })

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

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

 fmt.Printf("Created single region cluster: %s\n", *clusterProperties.Arn)
 return nil
}

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

 err := CreateCluster(ctx, "us-east-1")
 if err != nil {
  fmt.Printf("failed to create cluster: %v", err)
  panic(err)
 }

}
```

Obtención de un clúster

Los ejemplos siguientes muestran cómo obtener información sobre un clúster de una región con diferentes lenguajes de programación.

Python

Para obtener información sobre un clúster de una región, 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++

Use el ejemplo siguiente para obtener información sobre un clúster de una región.

```
#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, 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.

```
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.

```
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.

```
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.

```
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.

```
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)
 }
}
```

Actualización del clúster

En los ejemplos siguientes se muestra cómo actualizar un clúster de una región con diferentes lenguajes de programación.

Python

Para actualizar un clúster de una región, 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++

Use el siguiente ejemplo para actualizar un clúster de una región.

```
#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, 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

Use el siguiente ejemplo para actualizar un clúster de una región.

```
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

Use el siguiente ejemplo para actualizar un clúster de una región.

```
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

Use el siguiente ejemplo para actualizar un clúster de una región.

```
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

Use el siguiente ejemplo para actualizar un clúster de una región.

```
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

Use el siguiente ejemplo para actualizar un clúster de una región.

```
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)
 }
}
```

Eliminación de un clúster

En los ejemplos siguientes se muestra cómo eliminar un clúster de una región con diferentes lenguajes de programación.

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()
```

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;
}
```

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();
```

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);
        }
    }
}
```

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(())
}
```

Ruby

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

```
require "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__
```

.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);
        }
    }
}
```

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 ver más ejemplos y ejemplos de código, visite el repositorio de GitHub de ejemplos de Aurora DSQL.

Uso de la CLI de AWS

La CLI de AWS proporciona una interfaz de línea de comandos para administrar los clústeres de Aurora DSQL. En los siguientes ejemplos se muestran las operaciones de administración de clústeres comunes.

Crear 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

Configure y administre clústeres en varias Regiones de AWS mediante la AWS CLI o su lenguaje de programación preferido, incluyendo Python, C++, JavaScript, Java, Rust, Ruby, .NET y Golang. La AWS CLI proporciona acceso rápido a través de comandos de intérprete de comandos, mientras que los SDK de AWS permiten el control programático mediante la compatibilidad con el lenguaje nativo.

Temas

- Uso de los SDK de AWS
- Uso de la CLI de AWS

Uso de los SDK de AWS

Los SDK de AWS proporcionan acceso programático a Aurora DSQL en el lenguaje de programación preferido. En las siguientes secciones, se muestra cómo realizar operaciones de clúster comunes con distintos lenguajes de programación.

Crear un clúster

En los ejemplos siguientes se muestra cómo crear un clúster multirregional con diferentes lenguajes de programación.

Python

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={
                'Delay': 10,
                'MaxAttempts': 30
            }
        )

        return (cluster_1, cluster_2)

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


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)
    print("Created multi region clusters:")
    print("Cluster id: " + cluster_1['arn'])
    print("Cluster id: " + cluster_2['arn'])
```

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

C++

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);

    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;

    // Create second cluster
    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 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 updateClusterCommand = new UpdateClusterCommand({
            identifier: response1.identifier,
            multiRegionProperties: {
                witnessRegion: witnessRegion,
                clusters: [response2.arn]
            }
        });
        await client1.send(updateClusterCommand);
        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 ${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 ${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

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

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

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__
```

Golang

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)
```

```
 }

}
```

.NET

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}");
        }
    }
}
```

Obtención de un clúster

En los ejemplos siguientes se muestra cómo obtener información sobre un clúster multirregional con diferentes lenguajes de programación.

Python

Para obtener información sobre un clúster 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 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 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 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 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 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 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 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)
 }
}
```

Actualización del clúster

En los ejemplos siguientes se muestra cómo actualizar un clúster multirregional con diferentes lenguajes de programación.

Python

Para actualizar un clúster 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++

Use el siguiente ejemplo para actualizar un clúster 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 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

Use el siguiente ejemplo para actualizar un clúster 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

Use el siguiente ejemplo para actualizar un clúster 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

Use el siguiente ejemplo para actualizar un clúster 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

Use el siguiente ejemplo para actualizar un clúster 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

Use el siguiente ejemplo para actualizar un clúster 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)
 }
}
```

Eliminación de un clúster

En los ejemplos siguientes se muestra cómo eliminar un clúster multirregional con diferentes lenguajes de programación.

Python

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

```
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 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 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 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 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 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 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 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)
 }
}
```

Para ver más ejemplos y ejemplos de código, visite el repositorio de GitHub de ejemplos de Aurora DSQL.

Uso de la CLI de AWS

La CLI de AWS proporciona una interfaz de línea de comandos para administrar los clústeres de Aurora DSQL multirregionales. Los siguientes ejemplos demuestran cómo crear, configurar y eliminar clústeres multirregionales.

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.

Creación 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 Aurora DSQL mediante AWS CloudFormation

Puede usar el mismo recurso 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
```

Ciclo de vida del clúster de Aurora DSQL

La comprensión del ciclo de vida del clúster de Aurora DSQL le ayuda a administrar los clústeres de forma eficaz. En esta sección, se describen las definiciones del estado de los clústeres y la característica de escalado a cero que optimiza los costos.

Definición 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 Consola de administración de AWS, la AWS CLI o la API 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
Creación	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.
Activo	El clúster está en funcionamiento y listo para utilizarse.
Idle	Un clúster queda inactivo cuando permanece inactivo el tiempo suficiente para que Aurora DSQL reduzca los recursos en ejecución a fin de reducir la capacidad y los costos. Cuando se conecta a un clúster inactivo, Aurora DSQL devuelve el clúster al estado Activo.
Inactive	Un clúster se vuelve inactivo cuando no ha habido actividad en el clúster durante un periodo prolongado. En este estado suspendido, los recursos en

Estado	Descripción
	ejecución se escalan a cero mientras se conservan los datos. Cuando intenta conectarse a un clúster inactivo, Aurora DSQL devuelve el clúster al estado Activo de forma automática. El tiempo de restauración depende del tamaño del clúster.
Actualización de	Un clúster pasa al estado Actualizando al realizar cambios en la configuración del clúster.
Eliminación	Un clúster pasa al estado Eliminando cuando se envía una solicitud para eliminarlo.
Deleted (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 elimina el último clúster emparejado.

Trabajo con clústeres inactivos

Cuando Aurora DSQL no detecta actividad de conexión en un clúster durante un periodo de tiempo, pasa el clúster al estado inactivo, lo que reduce los recursos en ejecución para minimizar la capacidad y los costos. Si la actividad de conexión permanece ausente durante un periodo prolongado, el clúster inactivo pasa automáticamente al estado inactivo, en el que los recursos en ejecución se escalan a cero mientras se conservan los datos.

Para reanudar las operaciones normales, simplemente conéctese al clúster como de costumbre. Cuando se conecta al clúster de forma automática, Aurora DSQL pasa el clúster a estado Activo de forma automática.

> **Note**
>
> El primer intento de conexión a un clúster desocupado o inactivo será más lento de lo habitual.

Operaciones que requieren un estado de clúster activo

Algunas operaciones requieren que el clúster esté en estado activo. Para realizar estas operaciones en un clúster desocupado o inactivo, debe hacer la transición del clúster de nuevo a Activo conectándose al clúster.

Operaciones de copia de seguridad

La realización de una copia de seguridad requiere un estado de clúster activo. Si el clúster está desocupado o inactivo, las copias de seguridad producen el siguiente error:

```
"Error": {
    "Code": "FailedPrecondition",
    "Message": "Cluster 'cluster-id' is in state 'IDLE' and can't be backed up.
    In order to take a backup of your cluster, it must be in Active state. Please
    connect to your cluster to transition it to Active to perform the backup."
}
```

Para continuar con la copia de seguridad:

1. Conéctese al clúster mediante el cliente de base de datos preferido o la consola de Aurora DSQL para activarlo.
2. Espere a que se produzca la transición automática al estado activo.
3. Inicie la copia de seguridad una vez que el clúster esté en pleno funcionamiento.

> **Note**
>
> Las copias de seguridad existentes realizadas antes de que el clúster estuviera desocupado o inactivo siguen siendo válidas y no se ven afectadas. Los nuevos intentos de copia de seguridad en el clúster producirán un error hasta que el clúster esté conectado para reactivarse automáticamente.

Visualización del estado del clúster de Aurora DSQL

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

Consola

Siga estos pasos para ver el estado del clúster en la Consola de administración de AWS:

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
```

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.

CloudFormation

AWS::DSQL::Cluster es un recurso de CloudFormation que le permite crear y administrar clústeres de Aurora DSQL como parte de la infraestructura como código. 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 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.

Conectores para Aurora DSQL

Aurora DSQL proporciona conectores especializados que amplían los controladores de bases de datos existentes para permitir una autenticación e integración de IAM sin problemas con los servicios

de AWS. Estos conectores están diseñados para funcionar con los marcos de trabajo y lenguajes de programación populares y, al mismo tiempo, mantener la compatibilidad con los flujos de trabajo de PostgreSQL existentes.

Conectores adicionales están previstos para futuras versiones. Para obtener la información más reciente sobre la disponibilidad de los conectores, consulte el repositorio de ejemplos de Aurora DSQL.

Conexión a clústeres de Aurora DSQL con un conector de JDBC

El conector de Aurora DSQL para JDBC está diseñado como un complemento de autenticación que amplía la funcionalidad del controlador JDBC de PostgreSQL para permitir que las aplicaciones se autentiquen con Aurora DSQL mediante credenciales de IAM. El conector no se conecta directamente a la base de datos, pero proporciona una autenticación de IAM perfecta además del controlador JDBC de PostgreSQL subyacente.

El conector de Aurora DSQL para JDBC está diseñado para funcionar con el controlador JDBC de PostgreSQL y proporciona una integración perfecta con los requisitos de autenticación de IAM de Aurora DSQL.

Junto con el controlador JDBC de PostgreSQL, el conector de Aurora DSQL para JDBC permite la autenticación basada en IAM para Aurora DSQL. Ingresa una profunda integración con servicios de autenticación de AWS como AWS Identity and Access Management (IAM).

Acerca del conector

Aurora DSQL es un servicio de base de datos SQL distribuido que proporciona alta disponibilidad y escalabilidad para aplicaciones compatibles con PostgreSQL. Aurora DSQL requiere una autenticación basada en IAM con tokens de tiempo limitado que los controladores JDBC existentes no admiten de forma nativa.

La idea principal del conector de Aurora DSQL para JDBC es agregar una capa de autenticación sobre el controlador JDBC de PostgreSQL que gestione la generación de los tókenes de IAM, lo que permite a los usuarios conectarse a Aurora DSQL sin cambiar sus flujos de trabajo de JDBC existentes.

¿Qué es la autenticación de Aurora DSQL?

En Aurora DSQL, la autenticación implica:

- Autenticación de IAM: todas las conexiones utilizan la autenticación basada en IAM con tokens de tiempo limitado
- Generación de tokens: los tokens de autenticación se generan mediante credenciales de AWS y tienen una vida útil configurable

El conector de Aurora DSQL para JDBC está diseñado para comprender estos requisitos y generar automáticamente los tókenes de autenticación de IAM al establecer las conexiones.

Beneficios del conector de Aurora DSQL para JDBC

Aunque Aurora DSQL proporciona una interfaz compatible con PostgreSQL, los controladores de PostgreSQL existentes actualmente no admiten los requisitos de autenticación de IAM de Aurora DSQL. El conector de Aurora DSQL para JDBC permite a los clientes seguir utilizando los flujos de trabajo de PostgreSQL existentes y, al mismo tiempo, habilitar la autenticación de IAM mediante:

- Generación automática de tokens: los tokens de IAM se generan automáticamente mediante credenciales de AWS
- Integración perfecta: funciona con los patrones de conexión JDBC existentes
- Soporte de credenciales de AWS: admite varios proveedores de credenciales de AWS (predeterminados, basados en perfiles, etc.)

Uso del conector de Aurora DSQL para JDBC con agrupación de conexiones

El conector de Aurora DSQL para JDBC funciona con bibliotecas de agrupamiento de conexiones como HikariCP. El conector gestiona la generación del token de IAM durante el establecimiento de la conexión, lo que permite que los grupos de conexiones funcionen con normalidad.

Características principales de

Generación automática de token

Los tokens de IAM se generan automáticamente mediante credenciales de AWS.

Integración sin problemas

Funciona con los patrones de conexión de JDBC existentes sin requerir cambios en el flujo de trabajo.

Soporte de credenciales de AWS

Admite varios proveedores de credenciales de AWS (predeterminados, basados en perfiles, etc.).

Compatibilidad de grupo de conexiones

Funciona a la perfección con bibliotecas de agrupación de conexiones como HikariCP.

Requisitos previos

Antes de comenzar, asegúrese de que cumple los siguientes requisitos previos:

- Se ha creado un clúster en Aurora DSQL.
- Se ha instalado el kit de desarrollo de Java (JDK). Asegúrese de tener la versión 17 o superior.
- Configure los permisos de IAM adecuados para permitir que la aplicación se conecte a Aurora DSQL.
- Credenciales de AWS configuradas (mediante AWS CLI, variables de entorno o roles de IAM).

Uso del conector de Aurora DSQL para JDBC

Para usar el conector de Aurora DSQL para JDBC en la aplicación de Java, siga estos pasos:

1. Agregue las siguientes dependencias al proyecto de Maven:

```
<dependencies>
    <!-- Aurora DSQL Connector for JDBC -->
    <dependency>
        <groupId>software.amazon.dsql</groupId>
        <artifactId>aurora-dsql-jdbc-connector</artifactId>
        <version>1.0.0</version>
    </dependency>
</dependencies>
```

   Para los proyectos de Gradle, agregue esta dependencia:

```
implementation("software.amazon.dsql:aurora-dsql-jdbc-connector:1.0.0")
```

2. Cree una conexión básica al clúster de Aurora DSQL mediante el formato de conector de AWS DSQL PostgreSQL:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
import java.sql.Statement;

public class DsqlJdbcConnectorExample {
    public static void main(String[] args) {
        // Using AWS DSQL PostgreSQL Connector prefix
        String jdbcUrl = "jdbc:aws-dsql:postgresql://your-cluster.dsql.us-
east-1.on.aws/postgres?user=admin";

        try (Connection connection = DriverManager.getConnection(jdbcUrl)) {
            // Use the connection
            try (Statement statement = connection.createStatement()) {
                // Create a table
                statement.execute("CREATE TABLE IF NOT EXISTS test_table (id UUID
 PRIMARY KEY DEFAULT gen_random_uuid(), name VARCHAR(100))");

                // Insert data
                statement.execute("INSERT INTO test_table (name) VALUES ('Test
 Name')");

                // Query data
                try (ResultSet resultSet = statement.executeQuery("SELECT * FROM
 test_table")) {
                    while (resultSet.next()) {
                        System.out.println("ID: " + resultSet.getInt("id") + ",
 Name: " + resultSet.getString("name"));
                    }
                }
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Propiedades de configuración

El conector de Aurora DSQL para JDBC admite las siguientes propiedades de conexión:

usuario

Determina el usuario de la conexión y el método de generación del token utilizado. Ejemplo:: `admin`

token-duration-secs

Duración en segundos de la validez del token. Para obtener más información sobre los límites de los tokens, consulte Generación de un token de autenticación en Amazon Aurora DSQL.

profile

Se utiliza para crear una instancia de ProfileCredentialsProvider para la generación de un token con el nombre de perfil proporcionado.

region

Región de AWS para conexiones de Aurora DSQL. Es opcional. Cuando se proporcione, invalidará la región extraída de la URL.

database

El nombre de la base de datos a la que se va a conectar. El valor predeterminado es `postgres`.

Registro

Habilite el registro para solucionar cualquier problema que pueda surgir al utilizar el conector JDBC de Aurora DSQL.

El conector utiliza el sistema de registro integrado (java.util.logging) de Java. Puede configurar los niveles de registro creando un archivo de `logging.properties`:

```
# Set root logger level to INFO for clean output
.level = INFO

# Show Aurora DSQL Connector for JDBC FINE logs for detailed debugging
software.amazon.dsql.level = FINE

# Console handler configuration
handlers = java.util.logging.ConsoleHandler
java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

# Detailed formatter pattern with timestamp and logger name
```

```
java.util.logging.SimpleFormatter.format = %1$tH:%1$tM:%1$tS.%1$tL [%4$s] %3$s - %5$s%n
```

Ejemplos

Para ver ejemplos y casos de uso más completos, consulte el repositorio del conector de Aurora DSQL para JDBC

Conector de Aurora DSQL para Python

El conector de Aurora DSQL para Python integra la autenticación de IAM para conectar aplicaciones Python a los clústeres de Amazon Aurora DSQL. Internamente, utiliza las bibliotecas de cliente psycopg, psycopg2 y asyncpg.

El conector de Aurora DSQL para Python está diseñado como un complemento de autenticación que amplía la funcionalidad de las bibliotecas de clientes psycopg, psycopg2 y asyncpg para permitir que las aplicaciones se autentiquen con Amazon Aurora DSQL mediante credenciales de IAM. El conector no se conecta directamente a la base de datos, pero proporciona una autenticación de IAM perfecta además de las bibliotecas de clientes subyacentes.

Acerca del conector

Amazon Aurora DSQL es un servicio de base de datos SQL distribuido que proporciona alta disponibilidad y escalabilidad para aplicaciones compatibles con PostgreSQL. Aurora DSQL requiere una autenticación basada en IAM con tókenes de tiempo limitado que las bibliotecas de Python existentes no admiten de forma nativa.

La idea detrás del conector de Aurora DSQL para Python es agregar una capa de autenticación sobre las bibliotecas de clientes de psycopg, psycopg2 y asyncpg que gestiona la generación de los tókenes de IAM, lo que permite a los usuarios conectarse a Aurora DSQL sin cambiar sus flujos de trabajo existentes.

¿Qué es la autenticación de Aurora DSQL?

En Aurora DSQL, la autenticación implica:

- Autenticación de IAM: todas las conexiones utilizan la autenticación basada en IAM con tokens de tiempo limitado
- Generación de tókenes: los tókenes de autenticación se generan mediante credenciales de AWS y tienen una vida útil configurable

El conector de Aurora DSQL para Python está diseñado para comprender estos requisitos y generar automáticamente los tókenes de autenticación de IAM al establecer las conexiones.

Características

- Autenticación automática de IAM: los tókenes de IAM se generan automáticamente mediante credenciales de AWS
- Basado en psycopg, psycopg2 y asyncpg: aprovecha las bibliotecas de clientes de psycopg, psycopg2 y asyncpg
- Integración perfecta: funciona con los patrones de conexión de psycopg, psycopg2 y asyncpg existentes sin necesidad de cambiar el flujo de trabajo
- Detección automática de regiones: extrae la región de AWS del nombre de host del clúster de DSQL
- Compatibilidad de credenciales de AWS: admite varios proveedores de credenciales de AWS (predeterminados, basados en perfiles, personalizados)
- Compatibilidad con la agrupación de conexiones: funciona con la agrupación de conexiones integrada de psycopg, psycopg2 y asyncpg

Guía de inicio rápido

Requisitos

- Python 3.10 o superior
- Acceso a un clúster de Aurora DSQL
- Configure los permisos de IAM adecuados para permitir que la aplicación se conecte a Aurora DSQL.
- Credenciales de AWS configuradas (mediante CLI de AWS, variables de entorno o roles de IAM)

Instalación

```
pip install aurora-dsql-python-connector
```

Instalación de psycopg o psycopg2 o asyncpg de forma independiente

El instalador del conector de Aurora DSQL para Python no instala las bibliotecas subyacentes. Se deben instalar de forma independiente, por ejemplo:

```
# Install psycopg and psycopg pool
```

```
pip install "psycopg[binary,pool]"
```

```
# Install psycopg2
pip install psycopg2-binary
```

```
# Install asyncpg
pip install asyncpg
```

Nota:

Solo se debe instalar la biblioteca necesaria. Por lo tanto, si el cliente va a utilizar psycopg, solo es necesario instalar psycopg. Si el cliente va a utilizar psycopg2, solo es necesario instalar psycopg2. Si el cliente va a utilizar asyncpg, solo es necesario instalar asyncpg.

Si el cliente necesita más de una, es necesario instalar todas las bibliotecas necesarias.

Uso básico

psycopg

```
    import aurora_dsql_psycopg as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
    }

    conn = dsql.connect(**config)
    with conn.cursor() as cur:
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
```

```
    'user': "admin",
}

conn = dsql.connect(**config)
with conn.cursor() as cur:
    cur.execute("SELECT 1")
    result = cur.fetchone()
    print(result)
```

asyncpg

```
import asyncio
import aurora_dsql_asyncpg as dsql

config = {
    'host': "your-cluster.dsql.us-east-1.on.aws",
    'region': "us-east-1",
    'user': "admin",
}

conn = await dsql.connect(**config)
result = await conn.fetchrow("SELECT 1")
await conn.close()
print(result)
```

Uso solo del host

psycopg

```
import aurora_dsql_psycopg as dsql

conn = dsql.connect("your-cluster.dsql.us-east-1.on.aws")
```

psycopg2

```
import aurora_dsql_psycopg2 as dsql

conn = dsql.connect("your-cluster.dsql.us-east-1.on.aws")
```

asyncpg

```
import asyncio
```

```
    import aurora_dsql_asyncpg as dsql

    conn = await dsql.connect("your-cluster.dsql.us-east-1.on.aws")
```

Uso solo del ID del clúster

psycopg

```
    import aurora_dsql_psycopg as dsql

    conn = dsql.connect("your-cluster")
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    conn = dsql.connect("your-cluster")
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    conn = await dsql.connect("your-cluster")
```

Nota:

En el escenario “solo se usa el ID de clúster”, se usa la región que se configuró previamente en la máquina, por ejemplo:

```
aws configure set region us-east-1
```

Si no se ha establecido la región o el ID de clúster indicado se encuentra en una región diferente, la conexión producirá un error. Para que funcione, proporcione la región como parámetro, como se muestra en el siguiente ejemplo:

psycopg

```
    import aurora_dsql_psycopg as dsql
```

```
    config = {
            "region": "us-east-1",
    }

    conn = dsql.connect("your-cluster", **config)
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    config = {
            "region": "us-east-1",
    }

    conn = dsql.connect("your-cluster", **config)
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    config = {
            "region": "us-east-1",
    }

    conn = await dsql.connect("your-cluster", **config)
```

Cadena de conexión

psycopg

```
    import aurora_dsql_psycopg as dsql

    conn = dsql.connect("postgresql://your-cluster.dsql.us-east-1.on.aws/postgres?
user=admin&token_duration_secs=15")
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql
```

```
    conn = dsql.connect("postgresql://your-cluster.dsql.us-east-1.on.aws/postgres?
user=admin&token_duration_secs=15")
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    conn = await dsql.connect("postgresql://your-cluster.dsql.us-east-1.on.aws/
postgres?user=admin&token_duration_secs=15")
```

Configuración avanzada

psycopg

```
    import aurora_dsql_psycopg as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
        "profile": "default",
        "token_duration_secs": "15",
    }

    conn = dsql.connect(**config)
    with conn.cursor() as cur:
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

psycopg2

```
    import aurora_dsql_psycopg2 as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
        "profile": "default",
        "token_duration_secs": "15",
    }
```

```
    conn = dsql.connect(**config)
    with conn.cursor() as cur:
        cur.execute("SELECT 1")
        result = cur.fetchone()
        print(result)
```

asyncpg

```
    import asyncio
    import aurora_dsql_asyncpg as dsql

    config = {
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'region': "us-east-1",
        'user': "admin",
        "profile": "default",
        "token_duration_secs": "15",
    }

    conn = await dsql.connect(**config)
    result = await conn.fetchrow("SELECT 1")
    await conn.close()
    print(result)
```

Opciones de configuración

Opción	Tipo	Obliga io	Descripción
`host`	`string`	Sí	Nombre de host o ID de clúster del clúster de DSQL
`user`	`string`	No	Nombre de usuario de DSQL. Predeterm inado: admin
`dbname`	`string`	No	Nombre de la base de datos. Predeterminado: postgres
`region`	`string`	No	Región de AWS (se detecta automáticamente desde el nombre de host si no se proporciona)

Opción	Tipo	Obliga io	Descripción
`port`	`int`	No	El valor predeterminado es 5432
`custom_credentials_provider`	`CredentialProvider`	No	Proveedor de credenciales de AWS personalizadas
`profile`	`string`	No	El nombre del perfil de IAM. Valor predeterminado: predeterminado.
`token_duration_secs`	`int`	No	Tiempo de caducidad del token en segundos

También se admiten todas las opciones de conexión estándar de las bibliotecas psycopg, psycopg2 y asyncpg subyacentes, con la excepción de los parámetros krbsrvname y gsslib que no admite DSQL.

Uso del conector de Aurora DSQL para Python con agrupación de conexiones

El conector de Aurora DSQL para Python funciona con la agrupación de conexiones integrada de psycopg, psycopg2 y asyncpg. El conector gestiona la generación del token de IAM durante el establecimiento de la conexión, lo que permite que los grupos de conexiones funcionen con normalidad.

psycopg

En el caso de psycopg, el conector implementa una clase de conexión denominada DSQLConnection que se puede pasar directamente al constructor psycopg_pool.ConnectionPool. Para las operaciones asíncronas, también hay una versión asíncrona de la clase denominada DSQLAsyncConnection.

```
    from psycopg_pool import ConnectionPool as PsycopgPool

    ...
    pool = PsycopgPool(
        "",
        connection_class=dsql.DSQLConnection,
```

```
        kwargs=conn_params,
        min_size=2,
        max_size=8,
        max_lifetime=3300
    )
```

Nota: Configuración de configuración max_lifetime

El parámetro max_lifetime se debe establecer en menos de 3600 segundos (una hora), ya que es la duración máxima de conexión permitida por la base de datos de Aurora DSQL. Si se establece max_lifetime más bajo, el grupo de conexiones puede administrar de forma proactiva el reciclaje de conexiones, lo que resulta más eficaz que gestionar los errores de tiempo de espera de conexión de la base de datos.

psycopg2

Para psycopg2, el conector proporciona una clase denominada AuroraDSQLThreadedConnectionPool que hereda de psycopg2.pool.ThreadedConnectionPool. La clase AuroraDSQLThreadedConnectionPool solo invalida el método de conexión interna. El resto de la implementación lo proporciona psycopg2.pool.ThreadedConnectionPool sin cambios.

```
    import aurora_dsql_psycopg2 as dsql

    pool = dsql.AuroraDSQLThreadedConnectionPool(
            minconn=2,
            maxconn=8,
            **conn_params,
    )
```

asyncpg

Para asyncpg, el conector proporciona una función create_pool que devuelve una instancia de asyncpg.Pool.

```
    import asyncio
    import os

    import aurora_dsql_asyncpg as dsql

    pool_params = {
```

```
        'host': "your-cluster.dsql.us-east-1.on.aws",
        'user': "admin",
        "min_size": 2,
        "max_size": 5,
    }

    pool = await dsql.create_pool(**pool_params)
```

Autenticación

El conector gestiona automáticamente la autenticación de DSQL mediante la generación de tókenes a través del generador de tókenes del cliente de DSQL. Si no se proporciona la región de AWS, se analizará automáticamente a partir del nombre de host proporcionado.

Para obtener más información sobre la autenticación en Aurora DSQL, consulte la guía del usuario.

Administrador frente a usuarios habituales

- Los usuarios denominados `"admin"` utilizan automáticamente los tókenes de autenticación de administrador
- Todos los demás usuarios utilizan tókenes de autenticación que no son de administrador
- Los tókenes se generan de forma dinámica para cada conexión

Ejemplos

Para ver el código de ejemplo completo, consulte los ejemplos que se indican en las secciones siguientes. Para obtener instrucciones sobre cómo ejecutar los ejemplos, consulte los archivos READMDE de ejemplos.

psycopg

Ejemplos README

Descripción	Ejemplos
Uso del conector de Aurora DSQL para Python para conexiones básicas	Ejemplo de conexión básica

Descripción	Ejemplos
Uso del conector de Aurora DSQL para Python para conexiones asíncronas básicas	Ejemplo de conexión asíncrona básica
Uso del conector de Aurora DSQL para Python con grupo de conexiones	Ejemplo de conexión básica con grupo de conexiones
	Ejemplo de conexiones simultáneas con grupo de conexiones
Uso del conector de Aurora DSQL para Python con grupo de conexiones asíncronas	Ejemplo de conexión básica con grupo de conexiones asíncronas

psycopg2

Ejemplos README

Descripción	Ejemplos
Uso del conector de Aurora DSQL para Python para conexiones básicas	Ejemplo de conexión básica
Uso del conector de Aurora DSQL para Python con grupo de conexiones	Ejemplo de conexión básica con grupo de conexiones
	Ejemplo de conexiones simultáneas con grupo de conexiones

asyncpg

Ejemplos README

Descripción	Ejemplos
Uso del conector de Aurora DSQL para Python para conexiones básicas	Ejemplo de conexión básica
Uso del conector de Aurora DSQL para Python con grupo de conexiones	Ejemplo de conexión básica con grupo de conexiones
	Ejemplo de conexiones simultáneas con grupo de conexiones

Conexión a clústeres de Aurora DSQL con un conector de Go

El conector de Aurora DSQL para Go envuelve pgx con la autenticación IAM automática. El conector se encarga de la generación de tokens, la configuración SSL y la gestión de conexiones para que usted pueda centrarse en la lógica de su aplicación.

Acerca del conector

Aurora DSQL requiere una autenticación basada en IAM con tokens de duración limitada que los controladores de Go PostgreSQL existentes no admiten de forma nativa. El conector de Aurora DSQL para Go añade una capa de autenticación sobre el controlador pgx que gestiona la generación de tokens IAM, lo que le permite conectarse a Aurora DSQL sin cambiar sus flujos de trabajo de pgx existentes.

¿Qué es la autenticación de Aurora DSQL?

En Aurora DSQL, la autenticación implica:

- Autenticación de IAM: todas las conexiones utilizan la autenticación basada en IAM con tokens de tiempo limitado
- Generación de tokens: el conector genera tokens de autenticación utilizando credenciales de AWS, y estos tokens tienen una duración configurable.

El conector de Aurora DSQL para Go está diseñado para comprender estos requisitos y generar automáticamente los tokens de autenticación de IAM al establecer las conexiones.

Ventajas del conector de Aurora DSQL para Go

El conector de Aurora DSQL para Go le permite seguir utilizando sus flujos de trabajo de pgx existentes, al tiempo que habilita la autenticación de IAM a través de:

- Generación automática de tokens: el conector genera automáticamente tokens de IAM para cada conexión.
- Agrupación de conexiones: soporte integrado para `pgxpool` con generación automática de tokens por conexión.
- Configuración flexible: compatibilidad con puntos de conexión completos o ID de clúster con detección automática de región.
- Compatibilidad con credenciales de AWS: admite perfiles de AWS y proveedores de credenciales personalizados.

Características principales de

Administración automática de tokens

El conector genera automáticamente los tokens de IAM para cada nueva conexión mediante credenciales previamente resueltas.

Grupo de conexiones

Agrupación de conexiones mediante `pgxpool` con generación automática de tokens por conexión.

Configuración de host flexible

Admite tanto puntos de conexión de clúster completos como ID de clúster con detección automática de región.

Seguridad SSL

SSL siempre está habilitado con el modo de verificación completa y negociación TLS directa.

Requisitos previos

- Go 1.24 o posterior
- Credenciales de AWS configuradas
- Un clúster de Aurora DSQL

El conector usa la cadena de credenciales predeterminada del SDK de AWS para Go v2, que resuelve las credenciales en el siguiente orden:

1. Variables de entorno (AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY)
2. Archivo de credenciales compartidas (~/.aws/credentials)
3. Archivo de configuración compartido (~/.aws/config)
4. Rol de IAM para Amazon EC2/ECS/Lambda

Instalación

Instale el conector mediante los módulos Go:

```
go get github.com/awslabs/aurora-dsql-connectors/go/pgx/dsql
```

Inicio rápido

En el siguiente ejemplo, se muestra cómo crear un grupo de conexiones y ejecutar una consulta:

```
package main

import (
    "context"
    "log"

    "github.com/awslabs/aurora-dsql-connectors/go/pgx/dsql"
)

func main() {
    ctx := context.Background()

    // Create a connection pool
    pool, err := dsql.NewPool(ctx, dsql.Config{
        Host: "your-cluster.dsql.us-east-1.on.aws",
    })
    if err != nil {
        log.Fatal(err)
    }
    defer pool.Close()

    // Execute a query
    var greeting string
```

```
    err = pool.QueryRow(ctx, "SELECT 'Hello, DSQL!'").Scan(&greeting)
    if err != nil {
        log.Fatal(err)
    }
    log.Println(greeting)
}
```

Opciones de configuración

El conector es compatible con las siguientes opciones de configuración:

Campo	Tipo	Predeterminado	Descripción
Host	cadena	(obligatorio)	Punto de conexión de clúster o ID de clúster
Región	cadena	(detectado automátic amente)	Región de AWS; obligatorio si el host es un ID de clúster
Usuario	cadena	"admin"	Usuario de base de datos
Database	cadena	"postgres"	Nombre de base de datos
Puerto	int	5432	Database port (Puerto de base de datos)
Perfil	cadena	""	Nombre de perfil de AWS para las credenciales
TokenDura tionSecs	int	900 (15 minutos)	Duración de validez del token en segundos (máximo permitido: 1 semana, valor predeterminado: 15 minutos)
MaxConns	int32	0	Conexiones máximas de grupo (0 = pgxpool predeterminado)
MinConns	int32	0	Conexiones de grupo mínimas (0 = pgxpool predeterminado)

Campo	Tipo	Predeterminado	Descripción
MaxConnLifetime	time.Duration	55 minutos	Tiempo máximo de conexión

Formato de la cadena de conexión

El conector admite los formatos de cadena de conexión PostgreSQL y DSQL:

```
postgres://[user@]host[:port]/[database][?param=value&...]
dsql://[user@]host[:port]/[database][?param=value&...]
```

Parámetros de consulta compatibles:

- `region`: región de AWS
- `profile`: nombre de perfil de AWS
- `tokenDurationSecs`: duración de la validez del token en segundos

Ejemplos:

```
// Full endpoint (region auto-detected)
pool, _ := dsql.NewPool(ctx, "postgres://admin@cluster.dsql.us-east-1.on.aws/postgres")

// Using dsql:// scheme (also supported)
pool, _ := dsql.NewPool(ctx, "dsql://admin@cluster.dsql.us-east-1.on.aws/postgres")

// With explicit region
pool, _ := dsql.NewPool(ctx, "postgres://admin@cluster.dsql.us-east-1.on.aws/mydb?
region=us-east-1")

// With AWS profile
pool, _ := dsql.NewPool(ctx, "postgres://admin@cluster.dsql.us-east-1.on.aws/postgres?
profile=dev")
```

Uso avanzado

Configuración de host

El conector admite dos formatos de host:

Punto de conexión completo (región detectada automáticamente):

```
pool, _ := dsql.NewPool(ctx, dsql.Config{
    Host: "your-cluster.dsql.us-east-1.on.aws",
})
```

ID de clúster (región requerida):

```
pool, _ := dsql.NewPool(ctx, dsql.Config{
    Host:   "your-cluster-id",
    Region: "us-east-1",
})
```

Ajuste de configuración de grupo

Configure el grupo de conexiones para su carga de trabajo:

```
pool, err := dsql.NewPool(ctx, dsql.Config{
    Host:              "your-cluster.dsql.us-east-1.on.aws",
    MaxConns:          20,
    MinConns:          5,
    MaxConnLifetime:   time.Hour,
    MaxConnIdleTime:   30 * time.Minute,
    HealthCheckPeriod: time.Minute,
})
```

Uso de conexión única

Para scripts simples o cuando no es necesaria la agrupación de conexiones:

```
conn, err := dsql.Connect(ctx, dsql.Config{
    Host: "your-cluster.dsql.us-east-1.on.aws",
})
if err != nil {
    log.Fatal(err)
}
defer conn.Close(ctx)

// Use the connection
rows, err := conn.Query(ctx, "SELECT * FROM users")
```

Uso de perfiles de AWS

Especificación de un perfil de AWS para credenciales:

```
pool, err := dsql.NewPool(ctx, dsql.Config{
    Host:    "your-cluster.dsql.us-east-1.on.aws",
    Profile: "production",
})
```

Reintento de OCC

Aurora DSQL utiliza control de simultaneidad optimista (OCC). Cuando dos transacciones modifican los mismos datos, la primera en confirmarse tiene prioridad y la segunda recibe un error de OCC.

El paquete `occretry` proporciona ayudantes para el reintento automático con retroceso exponencial y fluctuación. Instálelo con:

```
go get github.com/awslabs/aurora-dsql-connectors/go/pgx/occretry
```

Use `WithRetry` para escrituras transaccionales:

```
err := occretry.WithRetry(ctx, pool, occretry.DefaultConfig(), func(tx pgx.Tx) error {
    _, err := tx.Exec(ctx, "UPDATE accounts SET balance = balance - $1 WHERE id = $2",
 100, fromID)
    if err != nil {
        return err
    }
    _, err = tx.Exec(ctx, "UPDATE accounts SET balance = balance + $1 WHERE id = $2",
 100, toID)
    return err
})
```

Para instrucciones DDL o simples, use `ExecWithRetry`:

```
err := occretry.ExecWithRetry(ctx, pool, occretry.DefaultConfig(),
    "CREATE TABLE IF NOT EXISTS users (id UUID PRIMARY KEY, name TEXT)")
```

> **Important**
>
> `WithRetry` administra `BEGIN/COMMIT/ROLLBACK` internamente. Su devolución de llamada recibe una transacción y debe contener solo operaciones de base de datos, además de ser segura para reintentarlo.

Ejemplos

Para ver ejemplos y casos de uso más completos, consulte los ejemplos del conector de Aurora DSQL para Go.

Ejemplo	Descripción
example_preferred	Recomendado: grupo de conexiones con consultas simultáneas
transacción	Gestión de transacciones con BEGIN/COMMIT/ROLLBACK
occ_retry	Cómo gestionar los conflictos de OCC con un retroceso exponencial
connection_string	Uso de cadenas de conexión para la configuración

Conectores de Aurora DSQL para Node.js

El conector de Aurora DSQL para node-postgres y el conector de Aurora DSQL para Postgres.js son complementos de autenticación que amplían la funcionalidad de los clientes de node-postgres y Postgres.js para permitir que las aplicaciones se autentiquen con Aurora DSQL mediante credenciales de IAM.

Conector de Aurora DSQL para node-postgres

El conector de Aurora DSQL para node-postgres es un conector de Node.js creado en node-postgres que integra la autenticación de IAM para conectar aplicaciones de JavaScript/TypeScript a los clústeres de Amazon Aurora DSQL.

El conector de Aurora DSQL está diseñado como un complemento de autenticación que amplía la funcionalidad del cliente y grupo de node-postgres para permitir que las aplicaciones se autentiquen con Amazon Aurora DSQL mediante credenciales de IAM.

Acerca del conector

Amazon Aurora DSQL es una base de datos distribuida nativa en la nube compatible con PostgreSQL. Aunque requiere autenticación de IAM y tókenes de duración determinada, los controladores de bases de datos tradicionales de Node.js carecen de este soporte integrado.

El conector de Aurora DSQL para node-postgres cierra esta brecha al implementar un middleware de autenticación que funciona a la perfección con node-postgres. Este enfoque permite a los desarrolladores mantener su código de node-postgres existente y, al mismo tiempo, obtener un acceso seguro basado en IAM a los clústeres de Aurora DSQL mediante la administración automatizada de tókenes.

¿Qué es la autenticación de Aurora DSQL?

En Aurora DSQL, la autenticación implica:

- Autenticación de IAM: todas las conexiones utilizan la autenticación basada en IAM con tokens de tiempo limitado
- Generación de tókenes: los tókenes de autenticación se generan mediante credenciales de AWS y tienen una vida útil configurable

El conector de Aurora DSQL para node-postgres está diseñado para comprender estos requisitos y generar automáticamente los tókenes de autenticación de IAM al establecer las conexiones.

Características

- Autenticación automática de IAM: gestiona la generación y la actualización de los tókenes de DSQL.
- Basado en node-postgres: aprovecha el popular cliente de PostgreSQL para Node.js.
- Integración perfecta: funciona con los patrones de conexión de node-postgres existentes
- Detección automática de regiones: extrae la región de AWS del nombre de host del clúster de DSQL.
- Compatibilidad total con TypeScript: proporciona seguridad total de tipos
- Compatibilidad de credenciales de AWS: admite varios proveedores de credenciales de AWS (predeterminados, basados en perfiles, personalizados)
- Compatibilidad con la agrupación de conexiones: funciona a la perfección con la agrupación de conexiones integrada.

Aplicación de ejemplo

En el ejemplo, se incluye una aplicación de ejemplo que muestra cómo utilizar el conector de Aurora DSQL para node-postgres. Para ejecutar el ejemplo incluido, consulte el ejemplo README.

Guía de inicio rápido

Requisitos

- Node.js 20+
- Acceso a un clúster de Aurora DSQL
- Configure los permisos de IAM adecuados para permitir que la aplicación se conecte a Aurora DSQL.
- Credenciales de AWS configuradas (mediante CLI de AWS, variables de entorno o roles de IAM)

Instalación

```
npm install @aws/aurora-dsql-node-postgres-connector
```

Dependencias de los pares

```
npm install @aws-sdk/credential-providers @aws-sdk/dsql-signer pg tsx
npm install --save-dev @types/pg
```

Uso

Conexiones de clientes

```
import { AuroraDSQLClient } from "@aws/aurora-dsql-node-postgres-connector";

const client = new AuroraDSQLClient({
  host: "<CLUSTER_ENDPOINT>",
  user: "admin",
});
await client.connect();
const result = await client.query("SELECT NOW()");
await client.end();
```

Conexión de grupo

```
import { AuroraDSQLPool } from "@aws/aurora-dsql-node-postgres-connector";
```

```
const pool = new AuroraDSQLPool({
  host: "<CLUSTER_ENDPOINT>",
  user: "admin",
  max: 3,
  idleTimeoutMillis: 60000,
});

const result = await pool.query("SELECT NOW()");
```

Uso avanzado

```
import { fromNodeProviderChain } from "@aws-sdk/credential-providers";
import { AuroraDSQLClient } from "@aws/aurora-dsql-node-postgres-connector";

const client = new AuroraDSQLClient({
  host: "example.dsql.us-east-1.on.aws",
  user: "admin",
  customCredentialsProvider: fromNodeProviderChain(), // Optionally provide custom
 credentials provider
});

await client.connect();
const result = await client.query("SELECT NOW()");
await client.end();
```

Opciones de configuración

Opción	Tipo	Obligatorio	Descripción
`host`	`string`	Sí	Nombre de host del clúster de DSQL
`username`	`string`	Sí	Nombre de usuario de DSQL
`database`	`string`	No	Nombre de base de datos
`region`	`string`	No	Región de AWS (se detecta automáticamente desde el nombre de host si no se proporciona)

Opción	Tipo	Obligatorio	Descripción
`port`	`number`	No	El valor predeterminado es 5432
`customCredentialsProvider`	`AwsCredentialIdentity / AwsCredentialIdentityProvider`	No	Proveedor de credenciales de AWS personalizadas
`profile`	`string`	No	El nombre del perfil de IAM (valor predeterminado: “predeterminado”)
`tokenDurationSecs`	`number`	No	Tiempo de caducidad del token en segundos

Se admiten todos los demás parámetros del cliente/grupo.

Autenticación

El conector gestiona automáticamente la autenticación de DSQL mediante la generación de tókenes a través del generador de tókenes del cliente de DSQL. Si no se proporciona la región de AWS, se analizará automáticamente a partir del nombre de host proporcionado.

Para obtener más información sobre la autenticación en Aurora DSQL, consulte la guía del usuario.

Administrador frente a usuarios habituales

- Los usuarios denominados “admin” utilizan automáticamente los tókenes de autenticación de administrador
- Todos los demás usuarios utilizan tókenes de autenticación habituales
- Los tókenes se generan de forma dinámica para cada conexión.

Conector de Aurora DSQL para Postgres.js

El conector de Aurora DSQL para Postgres.js es un conector de Node.js creado en Postgres.js que integra la autenticación de IAM para conectar aplicaciones de JavaScript a los clústeres de Amazon Aurora DSQL.

El conector de Aurora DSQL para Postgres.js está diseñado como un complemento de autenticación que amplía la funcionalidad del cliente de Postgres.js para permitir que las aplicaciones se autentiquen con Amazon Aurora DSQL mediante credenciales de IAM. El conector no se conecta directamente a la base de datos, pero proporciona una autenticación de IAM perfecta además del controlador de Postgres.js subyacente.

Acerca del conector

Amazon Aurora DSQL es un servicio de base de datos SQL distribuido que proporciona alta disponibilidad y escalabilidad para aplicaciones compatibles con PostgreSQL. Aurora DSQL requiere una autenticación basada en IAM con tókenes de tiempo limitado que los controladores de Node.js existentes no admiten de forma nativa.

La idea detrás del conector de Aurora DSQL para Postgres.js es agregar una capa de autenticación sobre el cliente de Postgres.js que gestiona la generación de los tókenes de IAM, lo que permite a los usuarios conectarse a Aurora DSQL sin cambiar sus flujos de trabajo de Postgres.js existentes.

El conector de Aurora DSQL para Postgres.js funciona con la mayoría de las versiones de Postgres.js. Los usuarios proporcionan su propia versión instalando Postgres.js directamente.

¿Qué es la autenticación de Aurora DSQL?

En Aurora DSQL, la autenticación implica:

- Autenticación de IAM: todas las conexiones utilizan la autenticación basada en IAM con tokens de tiempo limitado
- Generación de tókenes: los tókenes de autenticación se generan mediante credenciales de AWS y tienen una vida útil configurable

El conector de Aurora DSQL para Postgres.js está diseñado para comprender estos requisitos y generar automáticamente los tókenes de autenticación de IAM al establecer las conexiones.

Características

- Autenticación automática de IAM: gestiona la generación y la actualización de los tókenes de DSQL
- Basado en Postgres.js: aprovecha el rápido cliente de PostgreSQL para Node.js
- Integración perfecta: funciona con los patrones de conexión de Postgres.js existentes
- Detección automática de regiones: extrae la región de AWS del nombre de host del clúster de DSQL

- Compatibilidad total con TypeScript: proporciona seguridad total de tipos
- Compatibilidad de credenciales de AWS: admite varios proveedores de credenciales de AWS (predeterminados, basados en perfiles, personalizados)
- Compatibilidad con la agrupación de conexiones: funciona a la perfección con la agrupación de conexiones integrada de Postgres.js

Guía de inicio rápido

Requisitos

- Node.js 20+
- Acceso a un clúster de Aurora DSQL
- Configure los permisos de IAM adecuados para permitir que la aplicación se conecte a Aurora DSQL.
- Credenciales de AWS configuradas (mediante CLI de AWS, variables de entorno o roles de IAM)

Instalación

```
npm install @aws/aurora-dsql-postgresjs-connector
# Postgres.js is a peer-dependency, so users must install it themselves
npm install postgres
```

Uso básico

```
import { auroraDSQLPostgres } from '@aws/aurora-dsql-postgresjs-connector';

const sql = auroraDSQLPostgres({
  host: 'your-cluster.dsql.us-east-1.on.aws',
  username: 'admin',

});

// Execute queries
const result = await sql`SELECT current_timestamp`;
console.log(result);

// Clean up
await sql.end();
```

Uso de un ID de clúster en lugar de un host

```
const sql = auroraDSQLPostgres({
  host: 'your-cluster-id',
  region: 'us-east-1',
  username: 'admin',

});
```

Cadena de conexión

```
const sql = AuroraDSQLPostgres(
  'postgres://admin@your-cluster.dsql.us-east-1.on.aws'
);

const result = await sql`SELECT current_timestamp`;
```

Configuración avanzada

```
import { fromNodeProviderChain } from '@aws-sdk/credential-providers';

const sql = AuroraDSQLPostgres({
  host: 'your-cluster.dsql.us-east-1.on.aws',
  database: 'postgres',
  username: 'admin',
  customCredentialsProvider: fromNodeProviderChain(), // Optionally provide custom
 credentials provider
  tokenDurationSecs: 3600,                            // Token expiration (seconds)

  // Standard Postgres.js options
  max: 20,                                // Connection pool size
  ssl: { rejectUnauthorized: false }    // SSL configuration
});
```

Opciones de configuración

Opción	Tipo	Obliga io	Descripción
`host`	`string`	Sí	Nombre de host o ID de clúster del clúster de DSQL

Opción	Tipo	Obliga io	Descripción
`database`	`string?`	No	Nombre de base de datos
`username`	`string?`	No	Nombre de usuario de la base de datos (usa admin si no se proporciona)
`region`	`string?`	No	Región de AWS (se detecta automátic amente desde el nombre de host si no se proporciona)
`customCre dentialsP rovider`	`AwsCreden tialIdent ityProvider?`	No	Proveedor de credenciales de AWS personalizadas
`tokenDura tionSecs`	`number?`	No	Tiempo de caducidad del token en segundos

También se admiten todas las opciones estándar de Postgres.js.

Autenticación

El conector gestiona automáticamente la autenticación de DSQL mediante la generación de tókenes a través del generador de tókenes del cliente de DSQL. Si no se proporciona la región de AWS, se analizará automáticamente a partir del nombre de host proporcionado.

Para obtener más información sobre la autenticación en Aurora DSQL, consulte la guía del usuario.

Administrador frente a usuarios habituales

- Los usuarios denominados “admin” utilizan automáticamente los tókenes de autenticación de administrador
- Todos los demás usuarios utilizan tókenes de autenticación habituales
- Los tókenes se generan de forma dinámica para cada conexión

Ejemplo de uso

En GitHub hay disponibles ejemplos de JavaScript que utilizan el conector de Aurora DSQL para Postgres.js. Para obtener instrucciones sobre cómo ejecutar los ejemplos, consulte el directorio de ejemplos.

Descripción	Ejemplo
Agrupación de conexiones con consultas simultáneas, incluyendo la creación de tablas, inserciones y lecturas en varios trabajadores.	Ejemplo de grupo de conexiones (preferido)
Operaciones CRUD (crear tabla, insertar, seleccionar, eliminar) sin agrupación de conexiones	Ejemplo sin grupo de conexiones

Conexión a clústeres de Aurora DSQL con un conector de Ruby

El conector de Aurora DSQL para Ruby es un conector de Ruby basado en pg que integra la autenticación de IAM para conectar aplicaciones Ruby a clústeres de Amazon Aurora DSQL.

El conector se encarga de la generación de tokens, la configuración SSL y la agrupación de conexiones para que usted pueda centrarse en la lógica de su aplicación.

Acerca del conector

Amazon Aurora DSQL requiere una autenticación de IAM con tokens de duración limitada que los controladores PostgreSQL para Ruby existentes no admiten de forma nativa. El conector de Aurora DSQL para Ruby añade una capa de autenticación sobre el pg gem que gestiona la generación de tokens de IAM, lo que le permite conectarse a Aurora DSQL sin cambiar sus flujos de trabajo de pg existentes.

¿Qué es la autenticación de Aurora DSQL?

En Aurora DSQL, la autenticación implica:

- Autenticación de IAM: todas las conexiones utilizan la autenticación basada en IAM con tokens de tiempo limitado
- Generación de tokens: el conector genera tokens de autenticación utilizando credenciales de AWS, y estos tokens tienen una duración configurable.

El conector de Aurora DSQL para Ruby comprende estos requisitos y genera automáticamente los tokens de autenticación de IAM al establecer las conexiones.

Características

- Autenticación automática de IAM: gestiona la generación y la actualización de token de Aurora DSQL.
- Basado en pg: envuelve el popular gem de PostgreSQL para Ruby.
- Integración perfecta: funciona con los flujos de trabajo de pg gem existentes.
- Agrupación de conexiones: soporte integrado a través del gem `connection_pool` con la aplicación de max_lifetime.
- Detección automática de regiones: extrae la región de AWS del nombre de host del clúster de DSQL.
- Compatibilidad con credenciales de AWS: admite perfiles de AWS y proveedores de credenciales personalizadas.
- Reintento de OCC: reintento opcional del control de simultaneidad optimista con retroceso exponencial.

Aplicación de ejemplo

Para ver un ejemplo completo, consulte la aplicación de ejemplo en GitHub.

Guía de inicio rápido

Requisitos

- Ruby 3.1 o posterior
- Acceso a un clúster de Aurora DSQL
- Credenciales de AWS configuradas (mediante AWS CLI, variables de entorno o roles de IAM).

Instalación

Añada a su archivo gem:

```
gem "aurora-dsql-ruby-pg"
```

O instale directamente:

```
gem install aurora-dsql-ruby-pg
```

Uso

Conexión de grupo

```
require "aurora_dsql_pg"

# Create a connection pool with OCC retry enabled
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  occ_max_retries: 3
)

# Read
pool.with do |conn|
  result = conn.exec("SELECT 'Hello, DSQL!'")
  puts result[0]["?column?"]
end

# Write — you must wrap writes in a transaction
pool.with do |conn|
  conn.transaction do
    conn.exec_params("INSERT INTO users (id, name) VALUES (gen_random_uuid(), $1)",
 ["Alice"])
  end
end

pool.shutdown
```

Conexión única de

Para scripts simples o cuando no es necesaria la agrupación de conexiones:

```
conn = AuroraDsql::Pg.connect(host: "your-cluster.dsql.us-east-1.on.aws")
conn.exec("SELECT 1")
conn.close
```

Uso avanzado

Configuración de host

El conector admite tanto puntos de conexión de clúster completos (detección automática de la región) como ID de clúster (se requiere especificar la región):

```
# Full endpoint (region auto-detected)
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws"
)

# Cluster ID (region required)
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster-id",
  region: "us-east-1"
)
```

AWS Perfiles de

Especificación de un perfil de AWS para credenciales:

```
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  profile: "production"
)
```

Formato de la cadena de conexión

El conector admite los formatos de cadena de conexión PostgreSQL:

```
postgres://[user@]host[:port]/[database][?param=value&...]
postgresql://[user@]host[:port]/[database][?param=value&...]
```

Parámetros de consulta compatibles: `region`, `profile`, `tokenDurationSecs`.

```
# Full endpoint with profile
```

```
pool = AuroraDsql::Pg.create_pool(
  "postgres://admin@cluster.dsql.us-east-1.on.aws/postgres?profile=dev"
)
```

Reintento de OCC

Aurora DSQL utiliza control de simultaneidad optimista (OCC). Cuando dos transacciones modifican los mismos datos, la primera en confirmarse tiene prioridad y la segunda recibe un error de OCC.

El reintento de OCC es opcional. Establezca `occ_max_retries` al crear el grupo para habilitar el reintento automático con retroceso exponencial y fluctuación en `pool.with`:

```
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  occ_max_retries: 3
)

pool.with do |conn|
  conn.transaction do
    conn.exec_params("UPDATE accounts SET balance = balance - $1 WHERE id = $2", [100,
 from_id])
    conn.exec_params("UPDATE accounts SET balance = balance + $1 WHERE id = $2", [100,
 to_id])
  end
end
```

> **Warning**
>
> `pool.with` NO envuelve automáticamente su bloque en una transacción. Debe llamar a `conn.transaction` usted mismo para operaciones de escritura. En caso de conflicto de OCC, el conector vuelve a ejecutar todo el bloque, por lo que este debe contener solo operaciones de base de datos y ser seguro para repetir el intento.

Para omitir los reintentos en llamadas individuales, pase `retry_occ: false`:

```
pool.with(retry_occ: false) do |conn|
  conn.exec("SELECT 1")
end
```

Opciones de configuración

Campo	Tipo	Predeterminado	Descripción
host	Cadena	(obligatorio)	Punto de conexión de clúster o ID de clúster
region	Cadena	(detectado automáticamente)	Región de AWS; obligatorio si el host es un ID de clúster
usuario	Cadena	"admin"	Usuario de base de datos
database	Cadena	"postgres"	Nombre de base de datos
puerto	Entero	5432	Database port (Puerto de base de datos)
profile	Cadena	nil	Nombre de perfil de AWS para las credenciales
token_duration	Entero	900 (15 minutos)	Duración de validez del token en segundos (máximo permitido: 1 semana, valor predeterminado: 15 minutos)
credentials_provider	Aws::Credentials	nil	Proveedor de credenciales personalizadas
max_lifetime	Entero	3300 (55 minutos)	Tiempo máximo de conexión en segundos
application_name	Cadena	nil	Prefijo ORM para application_name
registrador	Logger	nil	Registrador de advertencias de reintento de OCC

Campo	Tipo	Predeterminado	Descripción
occ_max_retries	Entero	nil (deshabilitado)	Número máximo de reintentos de OCC en `pool.with` ; habilita los reintentos cuando se establece

`create_pool` también admite una palabra clave de `pool:` con un hash de opciones que usted pasa directamente a `ConnectionPool.new`. Si omite `pool:`, el conector se establece de forma predeterminada en `{size: 5, timeout: 5}`. Las claves que proporcione solo anularán esos valores predeterminados específicos.

```
pool = AuroraDsql::Pg.create_pool(
  host: "your-cluster.dsql.us-east-1.on.aws",
  pool: { size: 10, timeout: 10 }
)
```

Autenticación

El conector gestiona automáticamente la autenticación de Aurora DSQL generando tokens usando las credenciales de AWS. Si no proporciona la región de AWS, el conector la analiza a partir del nombre de host.

Para obtener más información sobre la autenticación en Aurora DSQL, consulte Autenticación y autorización para Aurora DSQL.

Administrador frente a usuarios habituales

- Los usuarios denominados “admin” utilizan automáticamente los tókenes de autenticación de administrador
- Todos los demás usuarios utilizan tókenes de autenticación habituales
- El conector genera tokens de forma dinámica para cada conexión

Conexión a clústeres de Aurora DSQL con un conector de .NET

El conector de Amazon Aurora DSQL para .NET es un conector de .NET basado en Npgsql que integra la autenticación de IAM para conectar aplicaciones .NET a clústeres de Amazon Aurora DSQL.

El conector se encarga de la generación de tokens, la configuración SSL y la agrupación de conexiones para que usted pueda centrarse en la lógica de su aplicación.

Acerca del conector

Amazon Aurora DSQL requiere una autenticación de IAM con tokens de duración limitada que los controladores PostgreSQL para .NET existentes no admiten de forma nativa. El conector de Aurora DSQL para .NET añade una capa de autenticación sobre Npgsql, que gestiona la generación de tokens de IAM, lo que le permite conectarse a Aurora DSQL sin cambiar sus flujos de trabajo de Npgsql existentes.

¿Qué es la autenticación de Aurora DSQL?

En Aurora DSQL, la autenticación implica:

- Autenticación de IAM: todas las conexiones utilizan la autenticación basada en IAM con tokens de tiempo limitado
- Generación de tokens: el conector genera tokens de autenticación utilizando credenciales de AWS, y estos tokens tienen una duración configurable.

El conector de Aurora DSQL para .NET tiene en cuenta estos requisitos y genera automáticamente tokens de autenticación de IAM al establecer conexiones.

Características

- Autenticación automática de IAM: gestiona la generación y la actualización de token de Aurora DSQL.
- Basado en Npgsql: incluye el popular controlador de PostgreSQL para.NET.
- Integración perfecta: funciona con los flujos de trabajo de Npgsql existentes.
- Agrupación de conexiones: soporte integrado a través de `NpgsqlDataSource` con aplicación de la duración máxima.
- Detección automática de regiones: extrae la región de AWS del nombre de host del clúster de DSQL.
- Compatibilidad con credenciales de AWS: admite perfiles de AWS y proveedores de credenciales personalizadas.
- Reintento de OCC: reintento opcional del control de simultaneidad optimista con retroceso exponencial.

- Aplicación de SSL: siempre utiliza SSL con el modo `verify-full` y negociación de TLS directa.

Aplicación de ejemplo

Para ver un ejemplo completo, consulte la aplicación de ejemplo en GitHub.

Guía de inicio rápido

Requisitos

- .NET 8.0 o posterior
- Acceso a un clúster de Aurora DSQL
- Credenciales de AWS configuradas (mediante AWS CLI, variables de entorno o roles de IAM).

Instalación

Añada el paquete a su proyecto:

```
dotnet add package Amazon.AuroraDsql.Npgsql
```

Uso

Conexión de grupo

```
using Amazon.AuroraDsql.Npgsql;

// Create a connection pool
await using var ds = await AuroraDsql.CreateDataSourceAsync(new DsqlConfig
{
    Host = "your-cluster.dsql.us-east-1.on.aws",
    OccMaxRetries = 3
});

// Read
await using (var conn = await ds.OpenConnectionAsync())
{
    await using var cmd = conn.CreateCommand();
    cmd.CommandText = "SELECT 'Hello, DSQL!'";
    var greeting = await cmd.ExecuteScalarAsync();
    Console.WriteLine(greeting);
```

```
}

// Transactional write with OCC retry
await ds.WithTransactionRetryAsync(async conn =>
{
    await using var cmd = conn.CreateCommand();
    cmd.CommandText = "INSERT INTO users (id, name) VALUES (gen_random_uuid(), @name)";
    cmd.Parameters.AddWithValue("name", "Alice");
    await cmd.ExecuteNonQueryAsync();
});
```

Conexión única de

Para scripts sencillos o cuando no se necesita el uso de un grupo de conexiones:

```
await using var conn = await AuroraDsql.ConnectAsync(new DsqlConfig
{
    Host = "your-cluster.dsql.us-east-1.on.aws"
});

await using var cmd = conn.CreateCommand("SELECT 1");
await cmd.ExecuteScalarAsync();
```

Reintento de OCC

Aurora DSQL utiliza control de simultaneidad optimista (OCC). Cuando dos transacciones modifican los mismos datos, la primera en confirmarse tiene prioridad y la segunda recibe un error de OCC.

El reintento de OCC es opcional. Establezca `OccMaxRetries` en la configuración para habilitar el reintento automático con retroceso exponencial y fluctuación. Use `WithTransactionRetryAsync` para escrituras transaccionales:

```
await ds.WithTransactionRetryAsync(async conn =>
{
    await using var cmd = conn.CreateCommand();

    cmd.CommandText = "UPDATE accounts SET balance = balance - 100 WHERE id = @from";
    cmd.Parameters.AddWithValue("from", fromId);
    await cmd.ExecuteNonQueryAsync();

    cmd.CommandText = "UPDATE accounts SET balance = balance + 100 WHERE id = @to";
    cmd.Parameters.Clear();
```

```
    cmd.Parameters.AddWithValue("to", toId);
    await cmd.ExecuteNonQueryAsync();
});
```

Para instrucciones DDL o simples, use `ExecWithRetryAsync`:

```
await ds.ExecWithRetryAsync("CREATE TABLE IF NOT EXISTS users (id UUID PRIMARY KEY,
 name TEXT)");
```

> **Important**
>
> `WithTransactionRetryAsync` administra `BEGIN`/`COMMIT`/`ROLLBACK` internamente y abre una nueva conexión para cada intento. Su devolución de llamada debe contener solo operaciones de base de datos, además de ser segura para reintentarlo.

Opciones de configuración

El conector también admite las cadenas de conexión `postgres://` y `postgresql://` con los parámetros de consulta `region` y `profile`.

Campo	Tipo	Predeterminado	Descripción
`Host`	`string`	(obligatorio)	Punto de conexión de clúster o ID de clúster de 26 caracteres
`Region`	`string?`	(detectado automátic amente)	Región de AWS; obligatorio si el `Host` es un ID de clúster
`User`	`string`	`"admin"`	Usuario de base de datos
`Database`	`string`	`"postgres"`	Nombre de base de datos
`Port`	`int`	`5432`	Database port (Puerto de base de datos)
`Profile`	`string?`	`null`	Nombre de perfil de AWS para las credenciales

Campo	Tipo	Predeterminado	Descripción
`CustomCredentialsProvider`	`AWSCredentials?`	`null`	Proveedor de credenciales de AWS personalizadas
`TokenDurationSecs`	`int?`	`null` (SDK predeterminado, 900s)	Duración de la validez del token en segundos
`OccMaxRetries`	`int?`	`null` (deshabilitado)	El número máximo predeterminado de reintentos de OCC para los métodos de reintento en el origen de datos
`OrmPrefix`	`string?`	`null`	Prefijo ORM precedido de `application_name`
`LoggerFactory`	`ILoggerFactory?`	`null`	Fábrica de registradores para advertencias y diagnósticos de reintentos
`ConfigureConnectionString`	`Action<NpgsqlConnectionStringBuilder>?`	`null`	Devolución de llamada para anular la configuración del grupo de conexiones o establecer propiedades adicionales de la cadena de conexión de Npgsql. SSL y `Enlist` son invariables de seguridad y no se pueden anular.

Autenticación

El conector gestiona automáticamente la autenticación de Aurora DSQL generando tokens usando las credenciales de AWS. Si no proporciona la región de AWS, el conector la analiza a partir del nombre de host.

Para obtener más información sobre la autenticación en Aurora DSQL, consulte Autenticación y autorización para Aurora DSQL.

Administrador frente a usuarios habituales

- Los usuarios denominados “admin” utilizan automáticamente los tókenes de autenticación de administrador
- Todos los demás usuarios utilizan tókenes de autenticación habituales
- El conector genera tokens de forma dinámica para cada conexión

Acceso a Aurora DSQL con clientes compatibles con PostgreSQL

Aurora DSQL usa el protocolo de conexión de PostgreSQL. Puede conectarse a PostgreSQL mediante una variedad de herramientas y clientes, como AWS CloudShell, psql, DBeaver y DataGrip. En la siguiente tabla, se resume cómo Aurora DSQL asigna los parámetros de conexión de PostgreSQL más comunes:

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 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 Uso de roles de base de datos y autenticación de IAM.
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.

PostgreSQL	Aurora DSQL	Notas
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 autenticación	Aurora DSQL requiere tokens de autenticación temporales en lugar de contraseñas de larga duración. Para obtener más información, consulte Generación de un token de autenticación en Amazon Aurora DSQL.

Al conectarse, Aurora DSQL requiere un token de autenticación de IAM firmado en lugar de una contraseña tradicional. Estos tókenes temporales se generan con la versión 4 de AWS Signature y solo se utilizan durante el establecimiento de la conexión. Una vez conectada, la sesión permanece activa hasta que finalice o el cliente se desconecte.

Si intenta abrir una nueva sesión con un token caducado, la solicitud de conexión producirá un error y deberá generarse un nuevo token. Para obtener más información, consulte Generación de un token de autenticación en Amazon Aurora DSQL.

Acceso a Aurora DSQL con clientes de SQL

Aurora DSQL admite varios clientes compatibles con PostgreSQL para conectarse al clúster. En las siguientes secciones, se describe cómo conectarse mediante PostgreSQL con AWS CloudShell o con la línea de comandos local, así como herramientas basadas en la interfaz gráfica de usuario, como DBeaver y JetBrains DataGrip. Cada cliente requiere un token de autenticación válido tal y como se describe en la sección anterior.

Temas

- Uso de DBeaver para acceder a Aurora DSQL
- Uso de JetBrains DataGrip para acceder a Aurora DSQL
- Uso del terminal interactivo de PostgreSQL (psql) para acceder a Aurora DSQL
- Uso del controlador de Aurora DSQL para SQLTools
- Solución de problemas

Uso de DBeaver para acceder a Aurora DSQL

DBeaver es un cliente SQL universal que se puede utilizar para administrar cualquier base de datos que tenga un controlador JDBC. Es ampliamente utilizado entre los desarrolladores y administradores de bases de datos debido a sus sólidas capacidades de visualización, edición y administración de datos. Gracias a las opciones de conectividad en la nube de DBeaver, puede conectar DBeaver a Aurora DSQL de forma nativa.

DBeaver Pro

Los productos DBeaver PRO ofrecen una integración nativa con Aurora DSQL a partir de la versión 25.3. Siga las instrucciones de la documentación de DBeaver para conectarse a su clúster de Aurora DSQL.

DBeaver Community Edition

DBeaver Community Edition es la versión gratuita y de código abierto. Visite la página de descarga para consultar las instrucciones de instalación. Para conectarse a DSQL desde DBeaver Community Edition, debe instalar el complemento Aurora DSQL para DBeaver.

El complemento Aurora DSQL para DBeaver se basa en el conector de Aurora DSQL para JDBC y permite la autenticación de IAM en los clústeres de Aurora DSQL. Se instala cómodamente a través de la interfaz de usuario de DBeaver y elimina la necesidad de escribir un código de generación de token o de proporcionar manualmente un token de IAM válido, lo que simplifica la autenticación y elimina los riesgos de seguridad asociados a las contraseñas tradicionales generadas por los usuarios.

Características

- Soporte de autenticación de IAM: conéctese a los clústeres de Aurora DSQL con las credenciales de AWS IAM para una autenticación segura y sin contraseñas
- Administración automática de controladores: instala y configura fácilmente el conector Aurora DSQL para JDBC
- Opciones de conexión flexibles: elija entre la configuración de conexión basada en el host o basada en la URL de JDBC

Complemento Aurora DSQL para la instalación de DBeaver

1. Con DBeaver abierto, vaya al menú desplegable Ayuda → Instalar nuevo software.

2. Haga clic en Añadir para añadir un nuevo repositorio.
3. Escriba:
   - Nombre: `Aurora DSQL Plugin`
   - Ubicación: `https://awslabs.github.io/aurora-dsql-dbeaver-plugin/update-site/`
4. Marque Conector de Aurora DSQL para JDBC.
5. Haga clic en Siguiente, acepte la licencia y complete la instalación.
6. Reinicie DBeaver cuando se le solicite.

Creación de una conexión de Aurora DSQL

1. Haga clic en la Nueva conexión de base de datos.
2. Seleccione Aurora SQL.
3. En Servidor, seleccione una de las siguientes opciones en la opción Conectar mediante:
   - Host
     - para habilitar las entradas de texto de la interfaz de usuario de los siguientes campos:
       - Punto de conexión: punto de conexión del clúster de DSQL.
       - Nombre de usuario: nombre de usuario de DSQL (por ejemplo, administrador).
       - Perfil de AWS: por ejemplo, predeterminado; el perfil estándar que se utiliza cuando no se especifica ningún perfil específico.
       - Región de AWS (opcional): debe coincidir con la región en la que existe su clúster de DSQL; de lo contrario, la autenticación fallará.
   - URL
     - URL de JDBC en este formato:

```
jdbc:aws-dsql:postgresql://{cluster_endpoint}/{database}?
user=admin&profile=default&region=us-east-1
```

     - Nota: En este modo, solo está habilitada la entrada de URL. Para añadir parámetros a la cadena de conexión JDBC, utilice el formato de parámetros de consulta de URL que comience por ? como primer parámetro y añada & para los parámetros siguientes.
4. Haga clic en Probar conexión para comprobar que la conexión de Aurora DSQL funciona.
5. Haga clic en Finalizar.

Solución de problemas

Problema con Windows el almacén de certificados de Windows

Los usuarios de Windows pueden tener problemas al descargar el controlador JDBC del conector de Aurora DSQL para desde Maven Central.

Causa: es posible que el almacén de certificados de Windows no incluya los certificados necesarios para acceder al repositorio de Maven Central.

Solución:

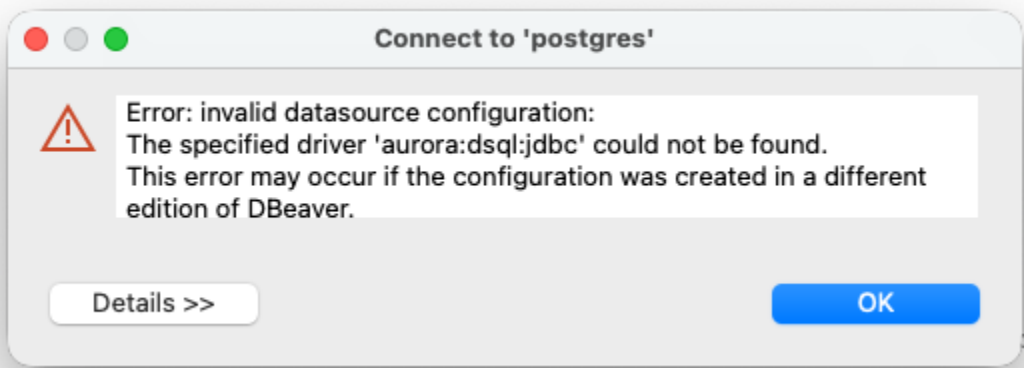
1. Ejecute DBeaver como «administrador».
2. Desactive esta opción: Windows **>** Preferencias **>** Conexiones **>** «Usar el almacén de certificados de Windows».

Error de controlador ausente

Si aparece un icono de controlador ausente o errores de conexión, es posible que Aurora DSQL (complemento comunitario) no esté instalado en su versión actual de DBeaver. A continuación, se muestran algunos ejemplos de errores y cómo solucionarlos:

- Creación de una nueva conexión con el controlador ausente:

- Intento de conexión sin el controlador:



Causa: cuando se instalan varias versiones de DBeaver, la configuración de conexión se comparte pero los controladores se instalan por aplicación.

Solución: vuelva a instalar Aurora DSQL (complemento comunitario) siguiendo los pasos de instalación anteriores.

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

Uso de JetBrains DataGrip para acceder a Aurora DSQL

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, ingrese 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 and PGSSLMODE=verify-full`. Aurora DSQL aplica la comunicación SSL en el servidor y rechaza las conexiones que no sean SSL. Para la opción `verify-full`, necesitará instalar los certificados SSL localmente. Para obtener más información, consulte Certificados SSL/TLS.

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

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

Uso del terminal interactivo de PostgreSQL (psql) para acceder a Aurora DSQL

Use AWS CloudShell para acceder a Aurora DSQL con la terminal interactiva de PostgreSQL (psql)

Use el procedimiento siguiente para acceder a Aurora DSQL con el terminal interactivo de PostgreSQL desde AWS CloudShell. Para obtener más información, consulte ¿Qué es AWS CloudShell?

Conexión mediante AWS CloudShell

1. Inicie sesión en la consola de Aurora DSQL.
2. Elija el clúster para el que desea abrir en CloudShell. 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 Creación de un clúster multirregional.
3. Elija Conectar con el editor de consultas y, a continuación, elija Conectar con CloudShell.
4. Elija si desea conectarse como admin o con un rol de base de datos personalizado.
5. Elija Lanzar en CloudShell y elija Ejecutar en el siguiente cuadro de diálogo de CloudShell.

Uso de la CLI local para acceder a Aurora DSQL con la terminal interactiva de PostgreSQL (psql)

Utilice `psql`, una utilidad frontend de PostgreSQL basado en terminal para ingresar consultas de forma interactiva, enviarlas a PostgreSQL y ver los resultados de las consultas.

> **Note**
>
> Para mejorar los tiempos de respuesta de las consultas, utilice el cliente de PostgreSQL versión 17. Si utiliza la CLI en un entorno diferente, asegúrese de configurar de forma manual Python versión 3.8 o superiores y psql versión 14 o superiores.

Descargue el instalador del sistema operativo desde la página de Descargas de PostgreSQL. Para obtener más información sobre `psql`, consulte Aplicaciones cliente de PostgreSQL en el sitio web de PostgreSQL.

Si ya tiene instalada la AWS CLI, utilice el siguiente ejemplo para conectarse al clúster.

```
# 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
```

Uso del controlador de Aurora DSQL para SQLTools

El controlador de Aurora DSQL para SQLTools es una extensión de Visual Studio Code para Amazon Aurora DSQL que se integra con SQLTools. Permite a los desarrolladores conectarse a las bases de datos de Aurora DSQL y consultarlas directamente desde VS Code. El controlador está disponible para su instalación en Visual Studio Marketplace y Open VSX Registry. Kiro, Cursor y otros IDE basados en VSCode pueden utilizar Open VSX Registry para instalar el controlador siguiendo el procedimiento de instalación estándar que se describe en esta página.

Características

- Autenticación automática de IAM
- Operaciones de bases de datos estándar, como navegar por esquemas y tablas y ejecutar consultas SQL.

Instalación

1. Abra la vista Extensiones.
2. Busque «Controlador de Aurora DSQL para SQLTools».

3. Haga clic en «Instalar».

Nota:

La Extensión SQLTools se instalará automáticamente si aún no está instalada.

Autenticación

En Aurora DSQL, todas las conexiones utilizan la autenticación basada en IAM con tokens de tiempo limitado. El controlador gestiona automáticamente la autenticación de Aurora DSQL mediante el Conector de Aurora DSQL para node-postgres.

Para obtener más información sobre la autenticación en Aurora DSQL, consulte la guía del usuario.

Creación de una conexión de Aurora DSQL

Requisitos previos

- Credenciales de AWS configuradas (mediante CLI de AWS, variables de entorno o roles de IAM)

Steps

1. Haga clic en el icono de SQLTools en la barra lateral izquierda.
2. En el panel de SQLTools, coloque el cursor sobre CONEXIONES y haga clic en el icono de añadir nueva conexión.
3. En la pestaña de configuración de SQLTools, seleccione el controlador de Aurora DSQL en la lista.
4. Rellene los parámetros de conexión.
   - Región AWS
     - Opcional: la región se analizará desde el punto de conexión del clúster de Aurora DSQL.
     - Se requiere cuando solo se especifica un ID de clúster en el campo «Clúster de DSQL».
   - Perfil de AWS
     - Se utiliza para la generación de tokens.
     - Si no se especifica, se utiliza el perfil predeterminado.
5. Haga clic en el botón «Probar conexión» para probar la conexión.
6. Haz clic en «Guardar conexión».

Solución de problemas

Expiración de las credenciales de autenticación de los clientes de SQL

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 la conexión. El intento de abrir una nueva sesión (por ejemplo, para enumerar las tablas nuevas o abrir 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. Tenga esto en cuenta cuando elija la duración del token de autenticación de IAM con la opción `expires-in`, que se puede configurar en quince minutos de forma predeterminada y establecerse en un valor máximo de siete días.

Además, consulte la sección Solución de problemas de la documentación de Aurora DSQL.

Herramientas de conectividad de clústeres de Amazon Aurora DSQL

Aurora DSQL es compatible con muchos controladores de bases de datos y bibliotecas ORM de terceros. AWS ofrece dos tipos de herramientas para simplificar el trabajo con Aurora DSQL:

- Conectores: complementos de autenticación que amplían los controladores de bases de datos para gestionar automáticamente la generación de tokens de IAM. Utilice conectores cuando trabaje directamente con controladores de bases de datos.
- Adaptadores y dialectos: extensiones para marcos ORM específicos que proporcionan autenticación de IAM y compatibilidad mejorada con Aurora DSQL. Utilice adaptadores cuando trabaje con un marco ORM compatible.

Adaptadores y dialectos de Aurora DSQL

En la siguiente tabla se muestran los adaptadores y los dialectos disponibles para Aurora DSQL.

Lenguaje de programación	ORM/Marco de trabajo	Enlace al repositorio
Java	Hibernate	https://github.com/awslabs/aurora-dsql-orms/tree/main/java/hibernate
Python	Django	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/django
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/sqlalchemy
Python	Tortoise ORM	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/tortoise-orm

Ejemplos de controlador de base de datos

En la siguiente tabla se muestra un código de ejemplo para conectarse a Aurora DSQL mediante controladores de bases de datos de terceros.

Lenguaje de programación	Controlador	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
Go	pgx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/go/pgx

Lenguaje de programación	Controlador	Enlace al repositorio de ejemplos
Java	HikariCP + pgJDBC	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/pgjdbc
JavaScript	node-postgres (AWS Lambda)	https://github.com/aws-samples/aurora-dsql-samples/tree/main/lambda
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	asyncpg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/python/asyncpg
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
Ruby	pg	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/ruby-pg

Lenguaje de programación	Controlador	Enlace al repositorio de ejemplos
Rust	SQLx	https://github.com/aws-samples/aurora-dsql-samples/tree/main/rust/sqlx

Ejemplos de ORM y marco

En la siguiente tabla se muestra código de ejemplo para usar marcos y bibliotecas ORM de terceros con Aurora DSQL.

Lenguaje de programación	ORM/Marco de trabajo	Enlace al repositorio de ejemplos
Java	Hibernate	https://github.com/awslabs/aurora-dsql-orms/tree/main/java/hibernate/examples/pet-clinic-app
Java	Liquibase	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/liquibase
Java	Spring Boot	https://github.com/aws-samples/aurora-dsql-samples/tree/main/java/spring_boot
Python	Django	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/django/examples/pet-clinic-app
Python	SQLAlchemy	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/sqlalchemy/examples/pet-clinic-app

Lenguaje de programación	ORM/Marco de trabajo	Enlace al repositorio de ejemplos
Python	Tortoise ORM	https://github.com/awslabs/aurora-dsql-orms/tree/main/python/tortoise-orm/example
Ruby	Rails	https://github.com/aws-samples/aurora-dsql-samples/tree/main/ruby/rails
TypeScript	Prisma	https://github.com/aws-samples/aurora-dsql-samples/tree/main/typescript/prisma
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

Carga de datos en Aurora DSQL

Tanto si está migrando desde una base de datos existente, importando archivos desde Amazon Simple Storage Service o cargando datos desde su sistema local, Aurora DSQL ofrece varios métodos para importar sus datos. En esta sección se describen las herramientas y técnicas recomendadas para cargas de datos de cualquier tamaño, desde gigabytes hasta cientos de terabytes.

Elección de un método de carga

Aurora DSQL es compatible con los comandos estándar de carga de datos de PostgreSQL, pero para cargar datos de forma eficiente a gran escala es necesario gestionar la paralelización, la administración de conexiones y la recuperación ante errores. En la siguiente tabla se resumen sus opciones:

Método	Lo mejor para	Consideraciones
Aurora DSQL Loader: utilidad de código abierto que facilita la paralelización de inserciones cuando se utiliza Aurora DSQL.	La mayoría de los casos de carga de datos, especialmente las migraciones y las importaciones masivas.	Gestiona automáticamente la paralelización, el uso compartido de conexiones, la resolución de conflictos y la autenticación de IAM. Disponible como código fuente o binario.
`\copy`PostgreSQL: metacomando `psql` del lado del cliente	Se carga fácilmente cuando ya está conectado a través de `psql`.	Lee archivos en el cliente y transmite datos a través de la conexión; usted mismo gestiona la paralelización.
Transacciones de **`INSERT`**: SQL DML estándar	Conjuntos de datos pequeños o inserciones impulsadas por aplicaciones	El método más simple, pero el más lento para los datos masivos.

Para la mayoría de las tareas de carga de datos, utilice Aurora DSQL Loader. Gestiona la complejidad operativa que supone la carga de datos en una base de datos distribuida, lo que incluye la ejecución en paralelo a través de varias conexiones y el reintento automático de las operaciones fallidas.

Aurora DSQL Loader

Aurora DSQL Loader es una utilidad de línea de comandos de código abierto diseñada para cargar datos de manera eficiente en los clústeres Aurora DSQL. Administra el agrupamiento de conexiones, paraleliza la transferencia de datos entre varios procesadores y gestiona automáticamente los conflictos y los reintentos.

Características principales de

Aurora DSQL Loader ofrece las siguientes funcionalidades:

Carga paralela

Los subprocesos de trabajo configurables permiten la carga simultánea de datos a través de varias conexiones, lo que mejora el rendimiento.

Grupo de conexiones

Administra un conjunto de conexiones a su clúster de Aurora DSQL, gestionando automáticamente la autenticación de IAM y el ciclo de vida de las conexiones.

Compatibilidad con varios formatos de archivo

Admite los formatos CSV (valores separados por comas), TSV (valores separados por tabulaciones) y el formato de columnas de Apache Parquet. El cargador detecta automáticamente el formato del archivo basándose en la extensión de la URI de origen.

Inferencia automática de esquemas

Cuando se utiliza con el indicador `--if-not-exists`, el cargador puede crear automáticamente tablas con los tipos de columna adecuados en función de los datos.

Gestión de conflictos

Cuando la tabla de destino tenga restricciones únicas, configure cómo gestiona el cargador los conflictos mediante la opción `--on-conflict`: omitir duplicados, actualizar o insertar registros, o devolver un error.

Tolerancia a errores

Las funciones de reintentos automáticos y reanudación de tareas garantizan que las cargas interrumpidas puedan continuar desde el punto en el que se detuvieron, en lugar de tener que reiniciarse por completo.

Fuentes locales y de S3

Cargue datos desde rutas del sistema de archivos local o directamente desde buckets de Amazon S3 utilizando URI de S3.

Requisitos previos

Antes de utilizar Aurora DSQL Loader, asegúrese de que dispone de lo siguiente:

- Un clúster activo de Aurora DSQL con un punto de conexión válido.
- Credenciales de AWS configuradas a través de AWS CLI (aws configure), AWS inicio de sesión único (aws sso login) o roles de IAM.

- Permisos de IAM: `dsql:DbConnectAdmin` o `dsql:DbConnect` en su clúster de Aurora DSQL.
- En el caso de las fuentes de S3, se requieren los permisos adecuados para leer desde el bucket de origen.

Instalación

Descargue la última versión desde la página de versiones de GitHub. Hay versiones precompiladas disponibles para las plataformas más habituales. Para obtener instrucciones sobre cómo compilar desde el código fuente, consulte el repositorio de Aurora DSQL Loader.

Ejemplos de uso

En los siguientes ejemplos se muestran casos de uso habituales de Aurora DSQL Loader.

Example Carga de un archivo CSV local

En este ejemplo se carga un archivo CSV desde su sistema de archivos local en una tabla ya existente:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table
```

Example Carga de datos desde Amazon S3

En este ejemplo se carga un archivo Parquet desde un bucket de Amazon S3:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri s3://my-bucket/data.parquet \
  --table my_table
```

Example Creación automática de tablas

En este ejemplo se crea automáticamente una nueva tabla basándose en el esquema de datos:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table \
```

```
  --if-not-exists
```

Example Validar antes de cargar

En este ejemplo se comprueba la configuración sin cargar realmente los datos:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table \
  --dry-run
```

Example Reanudación de una carga interrumpida

Si se interrumpe una operación de carga, puede reanudarla utilizando el ID de trabajo de la ejecución anterior:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri data.csv \
  --table my_table \
  --resume-job-id job-id \
  --manifest-dir ./loader-state
```

> **Note**
>
> Al reanudarla, el cargador omite la mayor parte del trabajo ya completado, pero puede volver a intentar procesar algunos registros. Si la tabla de destino tiene restricciones únicas, utilice la opción `--on-conflict` para gestionar los duplicados; por ejemplo, `DO NOTHINGx` para omitir o `DO UPDATE` para actualizar o insertar.

Opciones de línea de comandos

Aurora DSQL Loader admite las siguientes opciones de línea de comandos:

`--endpoint`

(Obligatorio) El punto de conexión del clúster de Aurora DSQL. Ejemplo:: *cluster-id*`.dsql.`*region*`.on.aws`

`--source-uri`

(Obligatorio) La ruta al archivo de datos. Puede ser una ruta de archivo local o un URI de S3 (por ejemplo, `s3://bucket-name/file.parquet`).

`--table`

(Obligatorio) El nombre de la tabla de destino de la base de datos de Aurora DSQL.

`--if-not-exists`

(Opcional) Cree automáticamente la tabla de destino si no existe. El cargador deduce el esquema a partir de los datos.

`--dry-run`

(Opcional) Compruebe la configuración y los datos sin cargarlos realmente en la base de datos.

`--resume-job-id`

(Opcional) Reanude una operación de carga interrumpida anteriormente utilizando el ID de trabajo especificado.

`--manifest-dir`

(Opcional) Directorio para almacenar el estado de los trabajos y los manifiestos, que se utiliza para reanudar los trabajos.

`--on-conflict`

(Opcional) Especifica cómo gestionar los conflictos al insertar filas que incumplan las restricciones únicas de la tabla de destino. Los valores válidos son `error` (devolver un error), `do-nothing` (omitir las filas duplicadas) o `do-update` (actualizar las filas existentes con valores nuevos).

Para obtener una lista completa de opciones y parámetros de configuración adicionales, ejecute:

```
aurora-dsql-loader load --help
```

Prácticas recomendadas

- Utilizar una simulación para la validación: compruebe siempre la configuración de la carga con `--dry-run` antes de introducir datos en las tablas de producción.

- Definir restricciones únicas para la reanudación: si necesita reanudar cargas interrumpidas, defina restricciones únicas en sus tablas de destino y utilice la opción `--on-conflict` para gestionar los registros ya cargados.
- Utilizar Parquet para conjuntos de datos de gran tamaño: el formato de columnas de Parquet suele ofrecer una mejor compresión y una carga más rápida para conjuntos de datos de gran tamaño en comparación con los formatos CSV o TSV.
- Conservar los directorios de manifiestos: conserve el directorio de manifiestos para los trabajos de carga hasta que confirme que la carga se ha completado correctamente, lo que permite la reanudación si es necesario.
- Crear tablas previamente siempre que sea posible: defina la tabla de destino con los tipos de datos de las columnas y las claves principales de forma explícita antes de cargar los datos. Los esquemas predefinidos le permiten controlar la precisión de los tipos y la indexación, lo que suele traducirse en un mejor rendimiento de las consultas en comparación con los esquemas inferidos automáticamente.

Solución de problemas

Errores de autenticación

Compruebe que sus credenciales de AWS estén configuradas correctamente y que su identidad de IAM disponga de los permisos `dsql:DbConnect` o `dsql:DbConnectAdmin` necesarios en el clúster de destino.

Errores de acceso a S3

Asegúrese de que su identidad de IAM disponga de los permisos de lectura de S3 adecuados para el bucket de origen y los objetos.

Errores de inferencia de esquemas

Cuando utilice `--if-not-exists`, asegúrese de que su archivo de datos tenga tipos de columna coherentes. La presencia de tipos mixtos en una columna puede provocar que falle la inferencia del esquema.

Errores de claves duplicadas en la reanudación

Si se producen errores de claves duplicadas al reanudar una carga, añada restricciones únicas a la tabla de destino para que el cargador pueda utilizar `ON CONFLICT DO NOTHING` para omitir los registros ya cargados.

Para obtener más información sobre la resolución de problemas, consulte el repositorio de GitHub de Aurora DSQL Loader.

Rutas de migración

En las siguientes secciones se describe cómo migrar datos desde sistemas de origen habituales a Aurora DSQL.

Migración desde PostgreSQL

Para migrar datos de una base de datos de PostgreSQL existente a Aurora DSQL:

1. Exporte sus datos de PostgreSQL a formato CSV o Parquet. Puede usar el comando `COPY` de PostgreSQL para exportar cada tabla:

   ```
   COPY my_table TO '/path/to/my_table.csv' WITH (FORMAT csv, HEADER true);
   ```

2. Cree la tabla de destino en Aurora DSQL. Puede crear el esquema manualmente o utilizar el indicador `--if-not-exists` del cargador para que este deduzca el esquema a partir de sus datos.
3. Cargue los datos exportados mediante Aurora DSQL Loader:

   ```
   aurora-dsql-loader load \
     --endpoint cluster-id.dsql.region.on.aws \
     --source-uri /path/to/my_table.csv \
     --table my_table
   ```

> **Tip**
>
> En el caso de migraciones de gran volumen, considere la posibilidad de exportar al formato Parquet para obtener una mejor compresión y una carga más rápida. Herramientas como DuckDB pueden convertir archivos CSV a Parquet de manera eficiente.

Migración desde MySQL

Para migrar datos de MySQL a Aurora DSQL:

1. Exporte sus datos de MySQL a formato CSV utilizando `SELECT INTO OUTFILE` o una herramienta como mysqldump con la opción `--tab`:

```
SELECT * FROM my_table
INTO OUTFILE '/path/to/my_table.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

2. Cree la tabla de destino en Aurora DSQL con los tipos de datos compatibles con PostgreSQL adecuados.
3. Cargue los datos exportados mediante Aurora DSQL Loader:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri /path/to/my_table.csv \
  --table my_table
```

> **Note**
>
> MySQL y PostgreSQL tienen sistemas de tipos de datos diferentes. Revise su esquema y ajuste los tipos de datos según sea necesario al crear tablas en Aurora DSQL.

Carga desde Amazon S3

Si sus datos ya se encuentran en Amazon S3, puede cargarlos directamente sin necesidad de descargarlos en su sistema local. Aurora DSQL Loader es compatible de forma nativa con las URI de S3:

```
aurora-dsql-loader load \
  --endpoint cluster-id.dsql.region.on.aws \
  --source-uri s3://my-bucket/path/to/data.parquet \
  --table my_table
```

Asegúrese de que su identidad de IAM tenga el permiso `s3:GetObject` sobre los objetos de origen.

Uso de PostgreSQL \copy

Si ya está conectado a Aurora DSQL a través de una sesión `psql` que gestiona la autenticación de IAM, puede utilizar el metacomando `\copy` del lado del cliente para cargar datos desde su sistema de archivos local. A diferencia de la instrucción `COPY` del lado del servidor, `\copy` lee el archivo en el equipo cliente y transmite los datos a través de la conexión existente, por lo que no es necesario acceder al archivo desde el lado del servidor. Este método funciona bien para cargas sencillas y de un solo subproceso.

Example Carga de un archivo CSV con \copy

```
\copy my_table FROM '/path/to/data.csv' WITH (FORMAT csv, HEADER true);
```

Si usa `\copy` directamente, usted es responsable de lo siguiente:

- Administrar la paralelización al cargar varios archivos o conjuntos de datos de gran tamaño.
- Gestionar la administración de conexiones y la actualización del token de autenticación.
- Implementar la lógica de reintento para operaciones fallidas.

Prácticas recomendadas para las transacciones INSERT

Cuando utilice instrucciones `INSERT` para cargar datos en Aurora DSQL, siga estas prácticas para mejorar el rendimiento y la fiabilidad:

- Agrupar filas en INSERT de varias filas: agrupe varias filas en una sola instrucción `INSERT` para reducir el número de idas y vueltas. Por ejemplo, `INSERT INTO my_table VALUES (1, 'a'), (2, 'b'), (3, 'c')` es más eficiente que tres instrucciones independientes.
- Utilizar consultas parametrizadas: utilice instrucciones preparadas con enlace de parámetros en lugar de concatenación de cadenas. Esto evita los riesgos de inyección de código SQL y permite que la base de datos reutilice los planes de consultas.
- Mantener las transacciones pequeñas: Aurora DSQL utiliza un control de simultaneidad optimista, por lo que las transacciones grandes que tocan muchas filas tienen más probabilidades de encontrar conflictos. Intente realizar transacciones de unos pocos cientos de filas en lugar de miles.
- Implementar una lógica de reintento: en un sistema distribuido es habitual que se produzcan errores transitorios, como los conflictos derivados del control de simultaneidad optimista (OCC). Implemente un retroceso exponencial con reintentos para las transacciones fallidas.

- Paralelizar entre conexiones: abra varias conexiones y distribuya las inserciones entre ellas. Cada conexión puede procesar un subconjunto de datos diferente de forma simultánea.

En la mayoría de los casos de uso, Aurora DSQL Loader ofrece un método más sencillo y fiable para la carga de datos.

Recursos adicionales

- Aurora DSQL Loader en GitHub: código fuente, documentación y seguimiento de incidencias
- Generación de un token de autenticación en Amazon Aurora DSQL: obtenga información sobre los tokens de autenticación de IAM para Aurora DSQL.
- Acceso a Aurora DSQL con clientes compatibles con PostgreSQL: conéctese a Aurora DSQL mediante diversos clientes y herramientas.

IA generativa para Aurora DSQL

En esta sección, se proporcionan instrucciones detalladas sobre cómo utilizar las herramientas de IA generativa con Aurora DSQL.

Servidor MCP de Aurora DSQL de Laboratorios de AWS

Un servidor de protocolo de contexto para modelos (MCP) de Laboratorios de AWS para Aurora DSQL

Características

- Conversión de preguntas y comandos legibles por humanos en consultas SQL estructuradas compatibles con Postgres y ejecutarlas en la base de datos de Aurora DSQL configurada.
- De forma predeterminada, son de solo lectura y las transacciones están habilitadas con `--allow-writes`
- Reutilización de conexiones entre solicitudes para mejorar el rendimiento
- Acceso integrado en la documentación, la búsqueda y las recomendaciones de prácticas recomendadas de Aurora DSQL

Herramientas disponibles

Operaciones de base de datos

- readonly_query: ejecute consultas SQL de solo lectura en el clúster de DSQL
- transact: ejecute operaciones de escritura en una transacción (obligatorio `--allow-writes`)
- get_schema: recupere la información del esquema de la tabla

Documentación y recomendaciones

- dsql_search_documentation: busque en la documentación de Aurora DSQL
  - Parámetros: `search_phrase` (obligatorio), `limit` (opcional)
- dsql_read_documentation: lee páginas específicas de documentación de DSQL
  - Parámetros: `url` (obligatorio), `start_index` (opcional), `max_length` (opcional)
- dsql_recommend: obtenga recomendaciones sobre las prácticas recomendadas de DSQL
  - Parámetros: `url` (obligatorio)

Requisitos previos

1. Una cuenta de AWS con un clúster de Aurora DSQL
2. Este servidor MCP solo se puede ejecutar localmente en el mismo host que el cliente de LLM.
3. Configuración de las credenciales de AWS con acceso a los servicios de AWS
   - Necesita una cuenta de AWS con un rol que incluya los siguientes permisos:
     - `dsql:DbConnectAdmin`: conéctese a los clústeres de DSQL como usuario administrador
     - `dsql:DbConnect`: conéctese a los clústeres de DSQL con roles de base de datos personalizados (solo es necesario si se utilizan usuarios que no sean administradores)
   - Configuración de las credenciales de AWS con `aws configure` o variables de entorno

Instalación

Para la mayoría de las herramientas, basta con actualizar la configuración siguiendo las instrucciones de Instalación predeterminada.

Se detallan instrucciones aparte para Claude Code y Codex.

Instalación predeterminada: actualización del archivo de configuración MCP pertinente

Uso de **uv**

1. Instalación de uv desde Astral o README de GitHub
2. Instalación de Python mediante `uv python install 3.10`

Configure el servidor MCP en la configuración del cliente de MCP (Búsqueda del archivo de configuración MCP)

```
{
  "mcpServers": {
    "awslabs.aurora-dsql-mcp-server": {
      "command": "uvx",
      "args": [
        "awslabs.aurora-dsql-mcp-server@latest",
        "--cluster_endpoint",
        "[your dsql cluster endpoint, e.g. abcdefghijklmnopqrst234567.dsql.us-
east-1.on.aws]",
        "--region",
        "[your dsql cluster region, e.g. us-east-1]",
        "--database_user",
        "[your dsql username, e.g. admin]",
        "--profile",
        "[your aws profile, e.g. default]"
      ],
      "env": {
        "FASTMCP_LOG_LEVEL": "ERROR"
      },
      "disabled": false,
      "autoApprove": []
    }
  }
}
```

Instalación de Windows

Para los usuarios de Windows, el formato de configuración del servidor MCP es ligeramente diferente:

```
{
  "mcpServers": {
    "awslabs.aurora-dsql-mcp-server": {
      "disabled": false,
      "timeout": 60,
      "type": "stdio",
      "command": "uv",
      "args": [
        "tool",
        "run",
        "--from",
        "awslabs.aurora-dsql-mcp-server@latest",
        "awslabs.aurora-dsql-mcp-server.exe"
      ],
      "env": {
        "FASTMCP_LOG_LEVEL": "ERROR",
        "AWS_PROFILE": "your-aws-profile",
        "AWS_REGION": "us-east-1"
      }
    }
  }
}
```

Búsqueda del archivo de configuración del cliente de MCP

Para ver algunas de las herramientas de desarrollo de Agentic más comunes, puede encontrar las configuraciones del cliente de MCP en las siguientes rutas de archivo:

- Kiro:
  - Configuración de usuario: `~/.kiro/settings/mcp.json`
  - Configuración del espacio de trabajo: `/path/to/workspace/.kiro/settings/mcp.json`
- Claude Code: consulte Instalación de Claude Code para obtener ayuda detallada sobre la configuración
  - Configuración de usuario: `~/.claude.json` en `"mcpServers"`
  - Configuración del proyecto: `/path/to/project/.mcp.json`
  - Configuración local: `~/.claude.json` en `"projects" -> "path/to/project" -> "mcpServers"`
- Cursor:
  - Global: `~/.cursor/mcp.json`
  - Proyecto: `/path/to/project/.cursor/mcp.json`
- Codex: `~/.codex/config.toml`

  - Cada servidor MCP se configura con una tabla `[mcp_servers.<server-name>]` en el archivo de configuración. Consulte las instrucciones de instalación de Codex personalizadas
- Warp:
  - Edición de archivos: `~/.warp/mcp_settings.json`
  - Editor de aplicaciones: `Settings > AI > Manage MCP Servers` y pegue json
- CLI de Amazon Q Developer: `~/.aws/amazonq/mcp.json`
- Cline: por lo general, una ruta de VS Code anidada: `~/.vscode-server/path/to/cline_mcp_settings.json`

Claude Code

Requisitos previos

Importante: La administración de servidores MCP solo está disponible a través de la experiencia de terminal de la CLI de Claude Code, no del modo de panel nativo de VS Code.

Instale primero la CLI de Claude Code. Para ello, siga el proceso recomendado de instalación nativa de Claude.

Elección del ámbito adecuado

Claude Code ofrece tres ámbitos diferentes: local (predeterminado), proyecto y usuario, y detalla qué ámbito elegir en función de la confidencialidad de las credenciales y de la necesidad de compartirlas. Consulte la documentación de Claude Code sobre los ámbitos de instalación de MCP para obtener más detalles.

1. Los servidores de ámbito local representan el nivel de configuración predeterminado y se almacenan en `~/.claude.json` en la ruta del proyecto. Ambos son privados para usted y solo se puede acceder a ellos en el directorio del proyecto actual. Este es el `scope` predeterminado al crear servidores MCP.
2. Los servidores con ámbito de proyecto permiten la colaboración en equipo, sin dejar de seguir siendo accesibles solo en un directorio de proyecto. Los servidores con ámbito de proyecto agregan un archivo `.mcp.json` al directorio raíz del proyecto. Este archivo está diseñado para registrarse en el control de versiones, lo que garantiza que todos los miembros del equipo tengan acceso a las mismas herramientas y servicios de MCP. Al agregar un servidor con ámbito de proyecto, Claude Code crea o actualiza este archivo de forma automática con la estructura de configuración adecuada.

3. Los servidores con ámbito de usuario se almacenan en `~/.claude.json` y proporcionan accesibilidad entre proyectos, lo que permite que estén disponibles en todos los proyectos de su máquina y, al mismo tiempo, mantengan la privacidad para su cuenta de usuario.

Uso de la CLI de Claude (recomendado)

El uso de una sesión de la CLI de `claude` interactiva permite mejorar la experiencia de solución de problemas, por lo que esta es la ruta recomendada.

```
claude mcp add amazon-aurora-dsql \
  --scope [one of local, project, or user] \
  --env FASTMCP_LOG_LEVEL="ERROR" \
  -- uvx "awslabs.aurora-dsql-mcp-server@latest" \
  --cluster_endpoint "[dsql-cluster-id].dsql.[region].on.aws" \
  --region "[dsql cluster region, eg. us-east-1]" \
  --database_user "[your-username]"
```

Solución de problemas: uso de Claude Code con Bedrock en otra cuenta de AWS

Si ha configurado Claude Code con una cuenta o un perfil de AWS de Bedrock que sea distinto del perfil necesario para conectarse a su clúster de DSQL, tendrá que proporcionar argumentos de entorno adicionales:

```
  --env AWS_PROFILE="[dsql profile, eg. default]" \
  --env AWS_REGION="[dsql cluster region, eg. us-east-1]" \
```

Modificación directa en el archivo de configuración

Claude Code requiere una nomenclatura alfanumérica, por lo que recomendamos nombrar su servidor: `aurora-dsql-mcp-server`.

Ámbito local

Actualice `~/.claude.json` dentro del campo `mcpServers` específico del proyecto:

```
{
```

```
  "projects": {
    "/path/to/project": {
      "mcpServers": {}
    }
  }
}
```

Ámbito de proyecto

Actualice `/path/to/project/root/.mcp.json` en el campo `mcpServers`:

```
{
  "mcpServers": {}
}
```

Ámbito de usuario

Actualice `~/.claude.json` dentro del campo `mcpServers` específico del proyecto:

```
{
  "mcpServers": {}
}
```

Codex

Opción 1: CLI de Codex

Si tiene instalada la CLI de Codex, puede utilizar el comando codex mcp para configurar los servidores MCP.

```
codex mcp add amazon-aurora-dsql \
  --env FASTMCP_LOG_LEVEL="ERROR" \
  -- uvx "awslabs.aurora-dsql-mcp-server@latest" \
  --cluster_endpoint "[dsql-cluster-id].dsql.[region].on.aws" \
  --region "[dsql cluster region, eg. us-east-1]" \
  --database_user "[your-username]"
```

Opción 2: config.toml

Para controlar con mayor precisión las opciones del servidor MCP, puede editar manualmente el archivo de configuración `~/.codex/config.toml`. Cada servidor MCP se configura con una tabla `[mcp_servers.<server-name>]` en el archivo de configuración.

```
[mcp_servers.amazon-aurora-dsql]
command = "uvx"
args = [
  "awslabs.aurora-dsql-mcp-server@latest",
  "--cluster_endpoint", "<DSQL_CLUSTER_ID>.dsql.<AWS_REGION>.on.aws",
  "--region", "<AWS_REGION>",
  "--database_user", "<DATABASE_USERNAME>"
]

[mcp_servers.amazon-aurora-dsql.env]
FASTMCP_LOG_LEVEL = "ERROR"
```

Verificación de la instalación

Para la CLI de Amazon Q Developer, la CLI de Kiro, la CLI/TUI de Claude o la CLI/TUI de Codex, ejecute `/mcp` para ver el estado del servidor MCP.

En el caso del IDE de Kiro, también puede ir a la pestaña `MCP SERVERS` del panel de Kiro, donde se muestran todos los servidores MCP configurados y sus indicadores de estado de conexión.

Opciones de configuración del servidor

`--allow-writes`

De forma predeterminada, el servidor mcp de dsql no permite operaciones de escritura (“modo de solo lectura”). Cualquier invocación de la herramienta de transacciones producirá un error en este modo. Para usar la herramienta de transacciones, permita escribir pasando el parámetro `--allow-writes`.

Recomendamos utilizar el acceso con privilegios mínimos al conectarse a DSQL. Por ejemplo, los usuarios deben usar un rol de solo lectura siempre que sea posible. El modo de solo lectura hace todo lo posible por parte del cliente para rechazar las mutaciones.

`--cluster_endpoint`

Este parámetro es obligatorio para especificar el clúster al que se va a conectar. Debe ser el punto de conexión completo del clúster, por ejemplo, `01abc2ldefg3hijklmnopqurstu.dsql.us-east-1.on.aws`

`--database_user`

Este parámetro es obligatorio para especificar el usuario como el que se va a conectar. Por ejemplo, `admin` o `my_user`. Tenga en cuenta que las credenciales de AWS que utilice deben tener permiso para iniciar sesión como ese usuario. Para obtener más información sobre la configuración y el uso de los roles de base de datos en DSQL, consulte Uso de los roles de base de datos con los roles de IAM.

`--profile`

Puede especificar el perfil de AWS que va a utilizar para las credenciales. Tenga en cuenta que esto no es compatible con la instalación de docker.

También se admite el uso de la variable de entorno `AWS_PROFILE` en la configuración de MCP:

```
"env": {
  "AWS_PROFILE": "your-aws-profile"
}
```

Si no se proporciona ninguno de los dos, el servidor MCP utilizará de forma predeterminada el perfil “predeterminado” del archivo de configuración de AWS.

`--region`

Se trata de un parámetro obligatorio para especificar la región de la base de datos de DSQL.

`--knowledge-server`

Parámetro opcional para especificar el punto de conexión remoto del servidor MCP para las herramientas de conocimiento de DSQL (búsqueda de documentación, lectura y recomendaciones). De forma predeterminada, está preconfigurado.

Ejemplo:

```
--knowledge-server https://custom-knowledge-server.example.com
```

Nota: Por motivos de seguridad, utilice solo puntos de conexión del servidor de información de confianza. El servidor debe ser un punto de conexión HTTPS.

`--knowledge-timeout`

Parámetro opcional para especificar el tiempo de espera en segundos para las solicitudes al servidor de información.

Valor predeterminado: `30.0`

Ejemplo:

```
--knowledge-timeout 60.0
```

Aumente este valor si se agotan los tiempos de espera al acceder a la documentación en redes lentas.

Dirección de Aurora DSQL: Skills y Powers

En esta sección se describe cómo configurar la dirección de IA para Aurora DSQL utilizando Skills y Powers. Estos archivos de configuración basados en Markdown proporcionan contexto y orientación que los asistentes de IA aplican automáticamente al generar código para mejorar la calidad del desarrollo de los agentes.

Descripción general

Los Skills y Powers son capacidades modulares que amplían la funcionalidad del asistente de IA para Aurora DSQL. Incluyen instrucciones, metadatos y recursos que los asistentes de IA utilizan automáticamente cuando trabajan con las bases de datos de Aurora DSQL.

Por qué usar Skills y Powers

Los Skills y Powers proporcionan varias ventajas clave para el desarrollo de Aurora DSQL:

- Asistentes de IA especializados: ofrecen conocimientos especializados específicos del dominio para Aurora DSQL, incluidas las prácticas recomendadas, los patrones SQL compatibles con Postgres y las optimizaciones de bases de datos distribuidas.

- Reducción de repetición: cree una vez y úselo de forma automática. Elimina la necesidad de proporcionar repetidamente la misma orientación en varias conversaciones.
- Eficiencia contextual: las Skills se cargan bajo demanda en lugar de consumir contexto por adelantado. La IA carga la información por etapas según se necesita.
- Aprendizaje continuo: a medida que evolucionan las funciones de Aurora DSQL, los asistentes de IA acceden automáticamente a los patrones actualizados cuando se actualizan las Skills.

Rutas de configuración recomendadas

Elija la ruta de configuración que se adapte a su entorno de desarrollo:

- the section called “CLI de Skills” (independiente de agente)
- the section called “Kiro Power”
- the section called “Claude Skill”
- the section called “Gemini Skill”
- the section called “Codex Skill”

La Skill de DSQL también se puede usar con otros agentes de codificación de IA copiando la carpeta de Skills en las `rules` de la herramienta o el directorio de `skills`.

CLI de Skills

La Skill de DSQL se puede instalar mediante la CLI de Skills. Este método de configuración independiente de agente funciona con la mayoría de los asistentes de codificación de IA y le permite instalar la Skill en varios agentes a la vez.

Configuración

Ejecute el siguiente comando para instalar la Skill de Aurora DSQL:

```
npx skills add awslabs/mcp --skill dsql
```

La CLI lo guiará a través de lo siguiente:

- Selección de agentes: elija en qué agentes desea instalar (Kiro, Claude Code, Cursor, Copilot, Gemini, Codex, Roo, Cline, OpenCode, Windsurf, etc.).
- Alcance de la instalación: elija entre las siguientes opciones:

  - Proyecto: instalación en el directorio actual (asignado a su proyecto)
  - Global: instalación en el directorio principal (disponible para todos los proyectos)
- Método de instalación: elija entre las siguientes opciones:
  - Enlace simbólico (recomendado): fuente única de información veraz, actualizaciones sencillas
  - Copia para todos los agentes: copias independientes para cada agente

Administración de Skills

Compruebe y actualice sus Skills en cualquier momento mediante:

```
npx skills check
npx skills update
```

Kiro Power

Los Powers de Kiro son paquetes unificados que incluyen herramientas de MCP con experiencia en marcos e instrucciones de dirección. Cada Power incluye un documento de punto de entrada que explica las herramientas MCP disponibles y los desencadenantes de activación, la configuración del servidor MCP y orientación adicional específica del flujo de trabajo cargada bajo demanda.

Los Powers se activan de forma dinámica en función del contexto del usuario. En lugar de cargar todas las herramientas por adelantado, los Powers mantienen un uso básico prácticamente nulo hasta que las palabras clave relevantes activen la activación.

Configuración

Para configurar el Kiro Power para Aurora DSQL:

1. Instálelo directamente desde el registro de Powers de Kiro.
2. Una vez redirigido al Power en el IDE, puede:
   - Seleccionar el botón Try Power (Probar Power). Recomendado para usuarios que desean que la IA les guíe en la configuración del servidor MCP o que buscan una experiencia de incorporación interactiva con Aurora DSQL para crear un nuevo clúster.
   - Abra un nuevo chat de Kiro y pregunte cualquier cosa relacionada con Aurora DSQL. Si lo desea, actualice la configuración de MCP con los datos de su clúster actual para probar la conexión del servidor MCP, de modo que pueda utilizarse inmediatamente con el Power. El agente de Kiro activará automáticamente el Power si lo identifica como valioso para completar la tarea del usuario.

Claude Skill

Las Skills de Claude son capacidades modulares que amplían la funcionalidad de Claude. Cada Skill incluye instrucciones, metadatos y recursos opcionales que Claude usa automáticamente cuando es relevante. Las Skills se basan en un sistema de archivos y se cargan bajo demanda para minimizar el uso del contexto.

Configuración sencilla con la CLI de Skills

La Skill se puede instalar en Claude Code mediante la the section called “CLI de Skills”. Para especificar únicamente Claude Code como agente en el que realizar la instalación, utilice:

```
npx skills add awslabs/mcp --skill dsql --agent claude-code
```

Alternativa: configuración directa mediante un clon de Git

La configuración alternativa toma un clon disperso del directorio dsql-skill y crea un enlace simbólico de este clon en la carpeta de `~/.claude/skills/`. Esto permite realizar cambios en la Skill siempre que sea necesario actualizarla.

Requisitos previos

- Git instalado

Pasos de configuración

1. Cree un directorio de repositorios base

```
mkdir -p .dsql_skill_repos
```

2. Clone de forma dispersa la Skill desde el repositorio de MCP

Clone solo la carpeta de `dsql-skill` (no otros archivos):

```
cd .dsql_skill_repos
git clone --filter=blob:none --no-checkout https://github.com/awslabs/mcp.git
cd mcp
git sparse-checkout init --cone
git sparse-checkout set src/aurora-dsql-mcp-server/skills/dsql-skill
git checkout
cd ../..
```

3. Aplique un enlace simbólico de la Skill al directorio de Skills

Añada el directorio de Skills (predeterminado: global/orientado al usuario):

```
mkdir -p ~/.claude/skills
```

> **Note**
>
> Si desea que esta sea una Skill específica del proyecto, utilice el directorio raíz de `.claude/skills/` del proyecto.

Añada el enlace simbólico:

```
ln -s "$(pwd)/.dsql_skill_repos/mcp/src/aurora-dsql-mcp-server/skills/dsql-skill"
 ~/.claude/skills/dsql-skill
```

4. Verifique la configuración

```
# Should show SKILL.md and other skill files
ls -la ~/.claude/skills/dsql-skill/
```

5. Verifique el uso de la Skill

Una vez configurada la Skill, de tener un nuevo comando de esta: `/dsql`. Es posible que deba reiniciar Claude Code después de añadir la habilidad para que pueda detectarse. Puede utilizar este comando desde la CLI o el panel de Claude Code, según sus preferencias.

Actualización de la Skill

Para obtener los cambios más recientes del repositorio:

```
cd .dsql_skill_repos/mcp
git pull
```

Estructura de directorios

Tras configurar una Skill global, debería ver estos directorios:

```
.dsql_skill_repos/
### mcp/                              # Sparse git checkout
```

```
    ### src/
        ### aurora-dsql-mcp-server/
            ### skills/
                ### dsql-skill/
                    ### SKILL.md
                    ### ...

~/.claude/
### skills/
    ### dsql-skill -> /path/to/.dsql_skill_repos/mcp/src/aurora-dsql-mcp-server/skills/
dsql-skill
```

> **Note**
>
> Añada `.dsql_skill_repos/` a su `.gitignore` si no desea realizar rastrearlo. El checkout disperso solo conserva la carpeta de Skills, lo que minimiza el uso del disco.

Gemini Skill

Para añadir la Skill de Aurora DSQL directamente en Gemini, elija un ámbito: `workspace` (contenido en el proyecto) o `user` (predeterminado, global) y utilice el instalador de Skills.

Configuración

```
gemini skills install https://github.com/awslabs/mcp.git --path src/aurora-dsql-mcp-
server/skills/dsql-skill --scope $SCOPE
```

Sustituya `$SCOPE` por `workspace` o `user`.

A continuación, puede utilizar el comando de Skill `/dsql` con Gemini, y Gemini detectará automáticamente cuándo debe utilizarse la Skill.

Codex Skill

Utilice el instalador de Skill desde la TUI o CLI de Codex utilizando la Skill `$skill-installer`.

Configuración

```
$skill-installer install dsql skill: https://github.com/awslabs/mcp/tree/main/src/
aurora-dsql-mcp-server/skills/dsql-skill
```

Reinicie Codex para obtener la Skill. Luego, la Skill se puede activar usando `$dsql`.

Comience a utilizar el editor de consultas de Aurora DSQL

Con el editor de consultas de Aurora DSQL, puede conectarse de forma segura a los clústeres de Aurora DSQL y ejecutar consultas SQL directamente desde la consola de administración de AWS sin instalar ni configurar clientes externos. Proporciona un espacio de trabajo intuitivo con resaltado de sintaxis integrado, autocompletado y asistencia de código inteligente. Puede explorar rápidamente los objetos del esquema, desarrollar y ejecutar consultas SQL y ver los resultados, todo desde una única interfaz.

En este tema se explican los pasos necesarios para conectarse a un clúster, ejecutar consultas, ver los resultados y explorar funciones avanzadas, como los planes de ejecución.

> **Note**
>
> El editor de consultas está disponible en todas las regiones donde sea compatible con Aurora DSQL. Para obtener más información acerca de la disponibilidad regional, consulte Servicios regionales de AWS.

Requisitos previos

Antes de comenzar, asegúrese de que cumple los siguientes requisitos:

- Tiene al menos un clúster de Aurora DSQL disponible. Para obtener más información sobre la creación de clústeres, consulte Paso 1: creación de un clúster de Aurora DSQL de una sola región.
- El punto de conexión del clúster es de acceso público. Actualmente, el editor de consultas no admite clústeres cuyo acceso público esté bloqueado por políticas basadas en recursos o clústeres administrados a través de puntos de conexión de VPC. Para obtener más información sobre las restricciones de acceso, consulte Bloqueo de acceso público con políticas basadas en recursos en Aurora DSQL y Administración y conexión a clústeres de Amazon Aurora DSQL mediante AWS PrivateLink.
- El usuario o el rol de IAM tiene los permisos necesarios para acceder al clúster y conectarse al mismo. Para obtener más información sobre los permisos, consulte . Uso de roles de base de datos y autenticación de IAM.

Trabajo con el editor de consultas

Apertura del editor de consultas

Apertura del editor de consultas

1. Abra la consola de Aurora DSQL.
2. En el panel de navegación, elija Query Editor (Editor de consultas).

Como alternativa, en la página Clústeres, seleccione el clúster que desee consultar y elija Conectar con el editor de consultas para iniciar el editor directamente.

> **Note**
>
> El estado de trabajo y de conexión no se guardan. Si sale de la consola de Aurora DSQL, cierra la pestaña del navegador o cierra sesión, se perderán las conexiones, el texto de la consulta y los resultados.

Conexión a un clúster

Conexión a un clúster

1. Si no existe ninguna conexión de clúster, el editor muestra No se ha conectado ningún clúster. Elija Conectar o seleccione + (Agregar) en el panel Explorador de clústeres para conectarse a un clúster existente.
2. (Opcional) Conéctese a varios clústeres o al mismo clúster usando roles diferentes.

Exploración de objetos de clúster

El explorador de clústeres muestra todas las conexiones de clústeres disponibles y le permite buscar objetos como bases de datos, esquemas, tablas y vistas. También proporciona acciones comunes como actualizar, crear una tabla y otras opciones específicas del contexto.

Ejecución de consultas

Para ejecutar una consulta

1. En el panel de pestañas del editor de consultas, ingrese la instrucción SQL. Por ejemplo:

   ```
   SELECT * FROM public.orders LIMIT 10;
   ```

2. Compruebe el contexto de clúster activo que se muestra en la parte superior derecha de la pestaña de consulta. Esto indica la conexión de clúster asociada a la pestaña de consulta actual.
3. (Opcional) Use el menú desplegable de conexiones para revisar todas las conexiones disponibles o cambiar a un clúster diferente. Al cambiar la conexión, se actualiza el lugar donde se ejecutan las consultas en esa pestaña.
4. Seleccione Ejecutar para ejecutar la consulta.

> **Note**
>
> Cada consulta puede devolver hasta 10 000 filas en el panel de resultados. Para conjuntos de datos más grandes, ajuste la consulta con filtros o límites.

Revisión de los resultados y los planes de ejecución

Una vez ejecutada la consulta, revise el resultado en el panel de resultados situado en la parte inferior del editor. De forma predeterminada, cada ejecución de consulta muestra la pestaña Resultados (tabla), que muestra el resultado de la consulta en forma de tabla.

Para obtener el plan de ejecución de la consulta, ejecute `EXPLAIN ANALYZE` o `EXPLAIN ANALYZE VERBOSE` para obtener información adicional sobre el rendimiento de la consulta. Para obtener más información sobre los planes de ejecución, consulte Lectura de los planes EXPLAIN de Aurora DSQL.

> **Tip**
>
> El comando `EXPLAIN ANALYZE VERBOSE` muestra las estimaciones de uso de la DPU, incluidos los valores de cálculo, lectura, escritura y total de la DPU, lo que proporciona una visibilidad inmediata de los recursos que consumen las instrucciones SQL individuales.

Editores de consultas: uso de JupyterLab con Aurora DSQL

En esta guía, se proporcionan instrucciones paso a paso acerca de cómo conectar y consultar Amazon Aurora DSQL mediante JupyterLab con Python. JupyterLab es un popular entorno informático interactivo que combina código, texto y visualizaciones en un solo documento. Se usa ampliamente para aplicaciones de ciencia de datos e investigación.

Las instrucciones que aparecen a continuación tratarán los aspectos básicos del uso de Aurora DSQL tanto en una instalación local de JupyterLab como en el uso de Amazon SageMaker AI, un servicio de machine learning totalmente administrado que proporciona un entorno alojado con una interfaz de usuario para los flujos de trabajo de datos.

Introducción

Requisitos

- Un clúster de Aurora DSQL
- Credenciales de AWS configuradas (solo para instalación local)
- Python versión 3.9 o superior (solo instalación local)

Uso de JupyterLab local

Para empezar a usar JupyterLab, los usuarios primero deben instalar la aplicación con el comando pip de Python:

```
pip install jupyterlab
```

A continuación, se puede abrir JupyterLab ejecutando **`jupyter lab`**. Esto abrirá la aplicación de JupyterLab en localhost:8888, accesible desde un navegador. Asegúrese de tener las credenciales de AWS configuradas en el entorno local antes de continuar.

Uso de Amazon SageMaker AI

En la consola de AWS, vaya a la página de la consola de Amazon SageMaker AI y, a continuación, a la sección Cuadernos en Aplicaciones e IDE. Desde allí, puede seleccionar Crear instancia de cuaderno para empezar a crear un entorno de SageMaker. Seleccione un tipo de instancia y una plataforma antes de hacer clic en Crear instancia de cuaderno.

Consulte la documentación de configuración de Amazon SageMaker AI para obtener más información sobre las opciones de configuración e instancia.

> **Note**
>
> Advertencia: El uso de Amazon SageMaker AI puede suponer cargos en la cuenta de AWS.

Cuando la instancia de SageMaker esté activa, podrá abrirla desde la sección de instancias de cuaderno con Abrir JupyterLab. Antes de empezar a utilizar Aurora DSQL en el cuaderno, debe proporcionar acceso al clúster de DSQL en el rol de IAM de la instancia de SageMaker. La forma más sencilla de hacerlo es seguir el enlace al rol de IAM en la página de instancias del cuaderno. Desde allí, puede editar las políticas adjuntas al rol de IAM de SageMaker. Consulte Autenticación y autorización para obtener más información sobre la configuración de una política de IAM para permitir el acceso a Aurora DSQL.

Conexión a Aurora DSQL mediante JupyterLab

Tras configurar una instancia de JupyterLab, los pasos para conectarse a Aurora DSQL son los mismos a nivel local y en SageMaker AI. Cree un cuaderno de Python 3 vacío, en el que pueda agregar celdas con código de Python.

En una celda de Python, descargue el certificado raíz de Amazon de la tienda de confianza oficial:

```
import urllib.request
urllib.request.urlretrieve('https://www.amazontrust.com/repository/AmazonRootCA1.pem',
 'root.pem')
```

Para conectarse a Aurora DSQL, instale primero el conector de Aurora DSQL para Python y el controlador de Psycopg en una celda de Python y, a continuación, impórtelo:

```
pip install aurora_dsql_python_connector psycopg
```

```
import aurora_dsql_psycopg as dsql
```

Una vez importado el conector, puede crear una configuración de DSQL y conectarse. El conector de Aurora DSQL Python gestionará automáticamente la creación de un token de autenticación en cada conexión.

```
config = {
    'host': "your-cluster.dsql.us-east-1.on.aws",
    'region': "us-east-1",
    'user': "admin"
    }

conn = dsql.connect(**config)
```

Al ejecutar el código, ahora debería tener una conexión de Psycopg a Aurora DSQL. A continuación, puede ejecutar consultas mediante el cursor de Psycopg y proporcionando su consulta SQL. Consulte la documentación de Psycopg para obtener más información sobre el uso de Psycopg con una base de datos compatible con Postgres. Esta consulta dará como resultado una lista de tuplas en `results_list`.

```
with conn:
    with conn.cursor() as cur:
        cur.execute("SELECT * FROM table")
        results_list = cur.fetchall()
```

Luego, puede usar marcos de Python como Pandas para analizar o visualizar los resultados de las consultas, por ejemplo:

```
pip install pandas

import pandas as pd

df = pd.DataFrame(tuples_list)
print(df)
print(f"Total records: {len(df)}")
```

cuaderno de ejemplo

En el repositorio de ejemplos de Aurora DSQL hay disponible un cuaderno de ejemplo con Aurora DSQL.

Documentación adicional

Documentación de configuración de Amazon SageMaker AI

Conector de Aurora DSQL para Python

Documentación de Pandas

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 Consola de administración de AWS, 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.

Restauración de clústeres de una sola región

Para restaurar un clúster de Aurora DSQL de una sola región, utilice la consola: 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. Para obtener instrucciones detalladas, consulte Restaurar un clúster de Aurora DSQL de una sola región.

Restauración de clústeres de varias regiones

La restauración de un clúster de varias regiones de Aurora DSQL se admite tanto a través de la consola: https://console.aws.amazon.com/backup como de la AWS CLI. Para obtener instrucciones detalladas, consulte Restaurar un clúster de Aurora DSQL de varias regiones.

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, debe asegurarse de que hay una copia idéntica de la copia de seguridad en todas las Regiones de AWS para los clústeres de varias regiones. Si aún no tiene esas copias, primero debe copiar la copia de seguridad en otra Región de AWS que admita clústeres de varias regiones.

Recomendamos crear copias de seguridad en Regiones de AWS clave para permitir opciones sólidas de recuperación ante desastres y cumplir con los requisitos de conformidad. Para ver Regiones de AWS disponibles de Aurora DSQL, consulte the section called “Región de AWSDisponibilidad de ”.

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

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 más información, consulte la Guía del usuario de AWS CloudTrail.

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
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.
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

Nombre de métrica de CloudWatch	Métrica	Dimensión: ResourceId	Unidad	Descripción
				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 Consola de administración de AWS 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 la nube en AWS es la máxima 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 aborda tanto la seguridad de la nube como la 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 viene determinada por el servicio de AWS que utilice. También es 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

- Resiliencia en Amazon Aurora DSQL
- 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 por 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 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 asociada 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 por 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 realizar:

- Creación, eliminación y actualización de clústeres de Aurora DSQL, incluidos los clústeres multirregionales
- Administración de las políticas insertadas del clúster (políticas de creación, visualización, actualización y eliminación)
- Agregación y eliminación de etiquetas de clústeres
- Muestra de los clústeres y visualización de la información sobre clústeres individuales
- Visualización de las etiquetas asociadas a los clústeres de Aurora DSQL
- Conexión a la base de datos como cualquier usuario, incluido el administrador
- Realización de operaciones de copia de seguridad y restauración para los clústeres de Aurora DSQL, como inicio, detención y supervisión de los trabajos de copia de seguridad y restauración
- Uso de claves de AWS KMS administradas por el cliente para el cifrado de clústeres
- Visualización de cualquier métrica de CloudWatch de la cuenta
- Uso de AWS Fault Injection Service (AWS FIS) para inyectar errores en los clústeres de Aurora DSQL para realizar pruebas de tolerancia a errores
- Creación de roles vinculados a servicios para el servicio de `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.
- `fis`: concede permisos para usar AWS Fault Injection Service (AWS FIS) para inyectar errores en los clústeres de Aurora DSQL para realizar pruebas de tolerancia a errores.

Puede encontrar la política de `AmazonAuroraDSQLFullAccess` en la consola de IAM y 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 y ver las políticas insertadas del clúster. 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 de `AmazonAuroraDSQLReadOnlyAccess` en la consola de IAM y 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 Consola de administración de AWS. Las entidades principales con estos permisos pueden realizar:

- Creación, eliminación y actualización de clústeres de Aurora DSQL, incluidos los clústeres multirregionales, con la consola
- Administración de las políticas insertadas del clúster a través de la consola (políticas de creación, visualización, actualización y eliminación)
- Muestra de los clústeres y visualización de la información sobre clústeres individuales
- Visualización de las etiquetas de cualquier recurso en la cuenta

- Conexión a la base de datos como cualquier usuario, incluido el administrador
- Realización de operaciones de copia de seguridad y restauración para los clústeres de Aurora DSQL, como inicio, detención y supervisión de los trabajos de copia de seguridad y restauración
- Uso de claves de AWS KMS administradas por el cliente para el cifrado de clústeres
- Lanzamiento de AWS CloudShell desde la Consola de administración de AWS
- Visualización de cualquier métrica de CloudWatch en la cuenta
- Uso de AWS Fault Injection Service (AWS FIS) para inyectar errores en los clústeres de Aurora DSQL para realizar pruebas de tolerancia a errores
- Creación de roles vinculados a servicios para el servicio de `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 Consola de administración de AWS.
- `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.

- `fis`: concede permisos para usar AWS FIS para inyectar errores en los clústeres de Aurora DSQL para realizar pruebas de tolerancia a errores.
- `access-analyzer:ValidatePolicy` concede permiso para el linter en el editor de políticas, que proporciona información en tiempo real sobre los errores, las advertencias y los problemas de seguridad de la política actual.
- `fis`: concede permisos para usar AWS Fault Injection Service (AWS FIS) para inyectar errores en los clústeres de Aurora DSQL para realizar pruebas de tolerancia a errores.

Puede encontrar la política de `AmazonAuroraDSQLConsoleFullAccess` en la consola de IAM y 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 y AmazonAuroraDSQLConsoleFullAccess	Se ha agregado compatibilidad para la integración de AWS Fault Injection Service (AWS FIS) con Aurora DSQL. Esto le permite inyectar errores en los clústeres de	19 de agosto de 2025

Cambio	Descripción	Fecha
	Aurora DSQL de una región y mutirregionales para probar la tolerancia a errores de las aplicaciones. Puede crear plantillas de experimentos en la consola de AWS FIS para definir escenarios de error y dirigirse a clústeres de Aurora DSQL específicos para realizar pruebas. Para obtener más información sobre estas políticas, consulte AmazonAuroraDSQLFullAccess y AmazonAuroraDSQLConsoleFullAccess.	

Cambio	Descripción	Fecha
Actualización de AmazonAuroraDSQLFullAccess, AmazonAuroraDSQLReadOnlyAccess y AmazonAuroraDSQLConsoleFullAccess	Se ha agregado compatibilidad con políticas basadas en recursos (RBP) con nuevos permisos: `PutClusterPolicy` , `GetClusterPolicy` y `DeleteClusterPolicy` . Estos permisos permiten administrar políticas insertadas en los clústeres de Aurora DSQL para el control de acceso detallado. Para obtener más información, consulte AmazonAuroraDSQLFullAccess, AmazonAuroraDSQLReadOnlyAccess y AmazonAuroraDSQLConsoleFullAccess.	15 de octubre de 2025

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

Cambio	Descripción	Fecha
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

Cambio	Descripción	Fecha
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, consulte las Preguntas frecuentes sobre la privacidad de datos. Para obtener información sobre la protección de datos en Europa, consulte la publicación del blog Shared Responsibility Model and 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 . Exigimos 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 la información confidencial almacenada 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 introduzca en etiquetas o campos de formato libre utilizados para los nombres se pueden 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.

Conformidad con FIPS

Los puntos de conexión del plano de datos de Aurora DSQL (puntos de conexión del clúster que se utilizan para las conexiones de bases de datos) utilizan módulos criptográficos validados por FIPS 140-2 de forma predeterminada. No se requieren puntos de conexión FIPS independientes para las conexiones de los clústeres.

Para las operaciones del plano de control, Aurora DSQL proporciona puntos de conexión FIPS dedicados en las regiones compatibles. Para obtener más información acerca de los puntos de conexión FIPS del plano de control, consulte Puntos de conexión y cuotas de Aurora DSQL en Referencia general de AWS.

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 Direct Connect. Para obtener más información, consulte ¿Qué es 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.

Protección de datos en regiones testigo

Al crear un clúster multirregional, una región testigo ayuda a permitir la recuperación automática de errores al participar en la replicación sincrónica de las transacciones cifradas. Si un clúster interconectado deja de estar disponible, la región testigo permanece disponible para validar y procesar las operaciones de escritura en la base de datos, lo que garantiza que no se pierda la disponibilidad.

Las regiones testigo protegen y aseguran sus datos mediante estas características de diseño:

- La región testigo recibe y almacena únicamente registros de transacciones cifradas. Nunca aloja, almacena ni transmite sus claves de cifrado.
- La región testigo se centra únicamente en las funciones de registro de transacciones de escritura y cuórum. No puede leer los datos por diseño.
- La región testigo funciona sin puntos de conexión de clústeres ni procesadores de consultas. Esto impide el acceso de los usuarios a la base de datos.

Para obtener más información sobre las regiones testigo, consulte Configuración de clústeres multirregionales.

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
- Cifrado en reposo en Aurora DSQL
- Uso de AWS KMS y claves de datos con Aurora DSQL
- Autorización del uso de la AWS KMS key para Aurora DSQL
- Contexto de cifrado de Aurora DSQL

- Supervisión de la interacción de Aurora DSQL con AWS KMS
- Creación de un clúster de Aurora DSQL cifrado
- Eliminación o actualización de una clave para el clúster de Aurora DSQL
- Consideraciones sobre el cifrado con Aurora DSQL

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 clientes

: 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

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
- Uso de claves de clúster con Aurora DSQL
- Almacenamiento en caché de la clave de clúster

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.

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 Consola de administración de AWS o desde la AWS CLI.

Console

Creación de un clúster cifrado en la Consola de administración de AWS

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 Consola de administración de AWS 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 Consola de administración de AWS

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 Consola de administración de AWS 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 Consola de administración de AWS, 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 administradores 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 AWS que se puede utilizar sin cargo adicional.

Temas

Público

La forma de utilizar AWS Identity and Access Management (IAM) varía en función del rol:

- Usuario del servicio: solicite permisos al administrador si no puede acceder a las características (consulte Solución de problemas de identidades y accesos en Amazon Aurora DSQL).
- Administrador del servicio: determine el acceso de los usuarios y envíe las solicitudes de permiso (consulte Cómo funciona Amazon Aurora DSQL con IAM).
- Administrador de IAM: escribe las políticas para administrar el acceso (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 autenticarse como el Usuario raíz de la cuenta de AWS, como un usuario de IAM o asumiendo un rol de IAM.

Se puede iniciar sesión como una identidad federada con las credenciales de un origen de identidad, como AWS IAM Identity Center (IAM Identity Center), la autenticación de inicio de sesión único o las credenciales de Google/Facebook. Para obtener más información sobre el inicio de sesión, consulte Cómo iniciar sesión en la Cuenta de AWS en la Guía del usuario de AWS Sign-In.

Para el acceso mediante programación, AWS proporciona un SDK y una CLI para firmar criptográficamente las solicitudes. Para obtener más información, consulte AWS Signature Version 4 para solicitudes de API en la Guía del usuario de IAM.

Usuario raíz de la Cuenta de AWS

Cuando se crea una Cuenta de AWS, se comienza con una identidad de inicio de sesión denominada usuario raíz de la Cuenta de AWS que tiene acceso completo a todos los Servicios de AWS y recursos. Se recomiendaencarecidamente que no utilice el usuario raíz para las tareas diarias. Para ver las tareas que requieren credenciales de usuario raíz, consulte Tareas que requieren credenciales de usuario raíz en la Guía del usuario de IAM.

Identidad federada

Como práctica recomendada, exija a los usuarios humanos que utilicen la federación con un proveedor de identidades para acceder a Servicios de AWS con credenciales temporales.

Una identidad federada es un usuario del directorio empresarial, proveedor de identidad web o Directory Service que accede a los Servicios de AWS mediante credenciales de un origen de identidad. Las identidades federadas asumen roles que proporcionan credenciales temporales.

Para una administración de acceso centralizada, se recomienda AWS IAM Identity Center. Para obtener más información, consulte ¿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 con permisos específicos para una sola persona o aplicación. Recomendamos el uso de credenciales temporales en lugar de usuarios de IAM con credenciales de

larga duración. Para obtener más información, consulte Exigir que los usuarios humanos utilicen la federación con un proveedor de identidades para acceder a AWS con credenciales temporales en la Guía del usuario de IAM.

Un grupo de IAM especifica un conjunto de usuarios de IAM y facilita la administración de los permisos para grupos grandes de usuarios. 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 con permisos específicos que proporciona credenciales temporales. Se puede asumir un rol cambiando de un usuario a un rol de IAM (consola) o llamando a una operación de la API de la AWS CLI o AWS. Para obtener más información, consulte Métodos para asumir un rol en la Guía del usuario de IAM.

Los roles de IAM son útiles para el acceso de usuario federado, los permisos de usuario de IAM temporales, el acceso entre cuentas, el acceso entre servicios y las aplicaciones que se ejecutan en Amazon EC2. Para obtener más información, consulte Acceso a recursos entre cuentas en IAM en la Guía del usuario de IAM.

Administración del acceso con políticas

Para controlar el acceso en AWS, se crean políticas y se adjuntan a identidades o recursos de AWS. Una política define los permisos cuando se asocia a una identidad o un recurso. AWS evalúa estas políticas cuando una entidad principal realiza una solicitud. La mayoría de las políticas se almacenan en AWS como documentos JSON. Para obtener más información sobre los documentos de políticas de JSON, consulte Información general de políticas de JSON en la Guía del usuario de IAM.

Mediante las políticas, los administradores especifican quién tiene acceso a qué, definiendo qué entidad principal puede realizar acciones sobre qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Un administrador de IAM crea políticas de IAM y las agrega a roles, que los usuarios pueden asumir posteriormente. Las políticas de IAM definen permisos independientemente del método que se utilice para realizar la operación.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de política de permisos JSON que asocia a una identidad (usuario, grupo o rol). Estas políticas controlan qué acciones pueden realizar las

identidades, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en la identidad, consulte Definición de permisos de IAM personalizados con políticas administradas por el cliente en la Guía del usuario de IAM.

Las políticas basadas en identidad pueden ser políticas insertadas (incrustadas directamente en una sola identidad) o políticas administradas (políticas independientes asociadas a varias identidades). Para obtener información sobre cómo elegir entre políticas administradas e insertadas, consulte Selección 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 políticas JSON que se asocian a un recurso. Los ejemplos incluyen 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. Debe especificar una entidad principal en una política basada en recursos.

Las políticas basadas en recursos son políticas insertadas que se encuentran en ese servicio. No se pueden utilizar políticas de IAM administradas por AWS en una política basada en recursos.

Otros tipos de políticas

AWS admite tipos de políticas adicionales que pueden establecer el máximo de permisos concedidos por los tipos de políticas más frecuentes:

- Límites de permisos: establecen los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM. Para obtener más información, consulte Límites de permisos para las entidades de IAM en la Guía del usuario de IAM.
- Políticas de control de servicios (SCP): especifican los permisos máximos para una organización o unidad organizativa en AWS Organizations. Para obtener más información, consulte Políticas de control de servicios en la Guía del usuario de AWS Organizations.
- Políticas de control de recursos (RCP): definen los permisos máximos disponibles para los recursos de las cuentas. Para obtener más información, consulte Políticas de control de recursos (RCP) en la Guía del usuario de AWS Organizations.
- Políticas de sesión: políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal para un rol o un usuario federado. Para obtener 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	Sí
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
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 la identidad, consulte Definición de permisos de IAM personalizados con políticas administradas por el cliente 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. Para obtener más información sobre los elementos que puede utilizar en una política de JSON, consulte Referencia de los elementos de la política 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

Compatibilidad con las políticas basadas en recursos: sí

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 basada en 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 puede utilizar políticas administradas de AWS de IAM en una política basada en recursos.

Para obtener información sobre cómo crear y administrar políticas basadas en recursos para los clústeres de Aurora DSQL, consulte Políticas basadas en recursos para Aurora DSQL.

Acciones de políticas para Aurora DSQL

Compatibilidad con las acciones de políticas: sí

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.

El elemento `Action` de una política JSON describe las acciones que puede utilizar para conceder o denegar el acceso en una política. 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 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.

El elemento `Resource` de la política JSON especifica el objeto u objetos a los que se aplica la acción. Como práctica recomendada, especifique un recurso utilizando el Nombre de recurso de Amazon (ARN). En el caso de las acciones que no admiten permisos por recurso, utilice 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 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.

El elemento `Condition` especifica cuándo se ejecutan las instrucciones en función de criterios definidos. Puede 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. 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 denominados etiquetas. Puede asociar etiquetas a entidades de IAM y recursos de AWS y, a continuación, diseñar políticas de ABAC para permitir operaciones cuando la etiqueta de la entidad principal coincida con la etiqueta del recurso.

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

Las credenciales temporales proporcionan acceso a corto plazo a los recursos de AWS y se crean automáticamente cuando se utiliza la federación o se cambia de rol. 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 y Servicios de AWS que funcionan con IAM en la Guía del usuario de IAM.

Permisos de entidades principales entre servicios de Aurora DSQL

Admite sesiones de acceso directo (FAS): sí

Las sesiones de acceso directo (FAS) utilizan 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. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte Sesiones de acceso directo.

Roles de servicio para Aurora DSQL

Compatible 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 Crear 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

Compatible con roles vinculados al servicio: sí

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.

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. Un administrador de IAM puede crear políticas de IAM para conceder permisos a los usuarios para realizar acciones en los recursos que necesitan.

Para obtener información acerca de cómo crear una política basada en identidad de IAM mediante el uso de estos documentos de políticas JSON de ejemplo, consulte Creación de políticas de IAM 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
- Permitir la administración del clúster y la conexión a bases de datos
- Acceso a recursos de Aurora DSQL basados en etiquetas

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:

- Comience a utilizar las políticas administradas de AWS y avance hacia permisos de privilegios mínimos. Para empezar a conceder permisos a los usuarios y cargas de trabajo, utilice las políticas administradas de AWS que otorgan permisos para muchos casos de uso comunes. Están disponibles en su Cuenta de AWS. Se recomienda definir políticas administradas 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, consulte 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, consulte 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, puede escribir una condición de políticas para especificar que todas las solicitudes deben enviarse utilizando SSL. También puede usar condiciones para conceder acceso a acciones de servicios si se emplean a través de un Servicio de AWS determinado como, por ejemplo, CloudFormation.

Para obtener más información, consulte 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 conceder 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, permita 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 administradas 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": "*"
        }
    ]
}
```

Permitir la administración del clúster y la conexión a bases de datos

La siguiente política otorga a un usuario de IAM permiso para administrar y conectarse a un clúster de Aurora DSQL específico. La política aplica las acciones de administración y conexión de clústeres a un único nombre de recurso de Amazon (ARN) del clúster, al tiempo que permite `dsql:ListClusters` en todos los recursos, ya que esta acción no admite permisos a nivel de recurso.

Este ejemplo se utiliza `dsql:DbConnectAdmin` para conectarse con el rol de `admin`. Para conectarse con un rol de base de datos personalizado, sustituya `dsql:DbConnectAdmin` por `dsql:DbConnect`. Para obtener más información, consulte Autenticación y autorización para Aurora DSQL.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowClusterManagement",
            "Effect": "Allow",
            "Action": [
                "dsql:GetCluster",
                "dsql:UpdateCluster",
                "dsql:DeleteCluster",
                "dsql:DbConnectAdmin",
                "dsql:TagResource",
                "dsql:ListTagsForResource",
                "dsql:UntagResource"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/my-cluster-id"
        },
        {
            "Sid": "AllowListClusters",
            "Effect": "Allow",
            "Action": "dsql:ListClusters",
            "Resource": "*"
        }
    ]
}
```

Acceso a recursos de Aurora DSQL basados en etiquetas

Puede utilizar las condiciones de su política basada en identidad para controlar el acceso a los recursos de Aurora DSQL basados en etiquetas. En el siguiente ejemplo se muestra cómo se puede crear una política que permita visualizar un clúster. Sin embargo, los permisos solo se conceden si la etiqueta de clúster `0wner` tiene el valor del nombre de usuario de dicho usuario. Esta política también proporciona los permisos necesarios para llevar a cabo esta acción en la consola.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ListClustersInConsole",
            "Effect": "Allow",
            "Action": "dsql:ListClusters",
            "Resource": "*"
        },
        {
            "Sid": "ViewClusterIfOwner",
            "Effect": "Allow",
            "Action": "dsql:GetCluster",
            "Resource": "arn:aws:dsql:*:*:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

También puede asociar esta política al usuario de IAM en su cuenta. Si un usuario llamado `richard-roe` intenta ver un clúster de Aurora DSQL, el clúster debe tener la etiqueta `0wner=richard-roe` o `owner=richard-roe`. De lo contrario, IAM deniega el acceso. La clave de la etiqueta de condición `0wner` coincide con los nombres de las claves de condición `0wner` y `owner` porque no distinguen entre mayúsculas y minúsculas. Para obtener más información, consulte Elementos de la política de JSON de IAM: Condición en la Guía del usuario de IAM.

La siguiente política permite a un usuario crear clústeres solo si etiqueta el clúster con su propio nombre de usuario como `Owner`. También permite etiquetar solo los clústeres que el usuario ya posee.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowCreateTaggedCluster",
            "Effect": "Allow",
            "Action": "dsql:CreateCluster",
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "${aws:username}"
                }
            }
        },
        {
            "Sid": "AllowTagOwnedClusters",
            "Effect": "Allow",
            "Action": "dsql:TagResource",
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

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

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, contacte con su administrador de . El administrador 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 administrador de AWS. El administrador 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

Se puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Se 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 sobre cómo proporcionar acceso a los recursos de las Cuentas de AWS de su propiedad, consulte Proporcionar acceso a un usuario de IAM a otra Cuenta de AWS de la que es propietario en la Guía del usuario de IAM.
- Para obtener información acerca de cómo proporcionar acceso a sus recursos a Cuentas de AWS de terceros, consulte 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, consulte 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.

Políticas basadas en recursos para Aurora DSQL

Utilice políticas basadas en recursos para Aurora DSQL a fin de restringir o conceder el acceso a los clústeres mediante documentos de política JSON que se adjuntan directamente a los recursos del clúster. Estas políticas proporcionan un control detallado sobre quién puede acceder al clúster y en qué condiciones.

Se puede acceder a los clústeres de Aurora DSQL desde el Internet público de forma predeterminada, con la autenticación de IAM como control de seguridad principal. Las políticas basadas en recursos le permiten agregar restricciones de acceso, especialmente para bloquear el acceso desde el Internet público.

Las políticas basadas en recursos funcionan junto con políticas basadas en identidad de IAM. AWS evalúa ambos tipos de políticas para determinar los permisos finales para cualquier solicitud de acceso al clúster. De forma predeterminada, los clústeres de Aurora DSQL son accesibles dentro de una cuenta. Si un usuario o rol de IAM tiene permisos de Aurora DSQL, puede acceder a los clústeres sin una política basada en recursos adjunta.

> **Note**
>
> Con el tiempo, los cambios en las políticas basadas en los recursos son coherentes y, por lo general, se hacen efectivos en un minuto.

Para obtener más información sobre las diferencias entre las políticas basadas en identidad y las políticas basadas en recursos, consulte Políticas basadas en identidad y políticas basadas en recursos en la Guía del usuario de IAM.

Cuándo utilizar políticas basadas en recursos

Las políticas basadas en recursos son particularmente útiles en estos escenarios:

- Control de acceso basado en la red: restrinja el acceso en función de la VPC o la dirección IP desde la que se originan las solicitudes o bloquee por completo el acceso público a Internet. Use claves de condición como `aws:SourceVpc` y `aws:SourceIp` para controlar el acceso a la red.
- Varios equipos o aplicaciones: conceda acceso al mismo clúster para varios equipos o aplicaciones. En lugar de administrar las políticas de IAM individuales para cada entidad principal, se definen las reglas de acceso una vez en el clúster.

- Acceso condicional complejo: controle el acceso en función de varios factores, como los atributos de la red, el contexto de la solicitud y los atributos del usuario. Puede combinar varias condiciones en una sola política.
- Gobernanza de seguridad centralizada: permita a los propietarios de los clústeres controlar el acceso mediante una sintaxis de políticas de AWS familiar que se integre con las prácticas de seguridad actuales.

> **Note**
>
> Las políticas basadas en recursos de Aurora DSQL aún no admiten el acceso entre cuentas, pero estará disponible en futuras versiones.

Cuando alguien intenta conectarse al clúster de Aurora DSQL, AWS evalúa su política basada en recursos como parte del contexto de autorización, junto con cualquier política de IAM pertinente, para determinar si la solicitud se debe permitir o rechazar.

Las políticas basadas en recursos pueden conceder acceso a las entidades principales de la misma cuenta de AWS que el clúster. En el caso de los clústeres de varias regiones, cada clúster regional tiene su propia política basada en los recursos, lo que permite establecer controles de acceso específicos para cada región cuando sea necesario.

> **Note**
>
> Las claves de contexto de condición pueden variar entre regiones (como los ID de VPC).

Temas

Creación de clústeres con políticas basadas en recursos

Puede adjuntar políticas basadas en recursos al crear un nuevo clúster para garantizar que los controles de acceso estén implementados desde el principio. Cada clúster puede tener una única política insertada adjunta directamente al clúster.

AWSConsola de administración de

Agregación de una política basada en recursos durante la creación del clúster

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. Elija Create cluster.
3. Configure el nombre, las etiquetas y los ajustes multirregionales del clúster según sea necesario.
4. En la sección Configuración del clúster, busque la opción de política basada en recursos.
5. Active Agregar una política basada en recursos.
6. Ingrese el documento de la política en el editor JSON. Por ejemplo, para bloquear el acceso público a Internet:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

7. Puede usar Editar instrucción o Agregar nueva instrucción para crear la política.
8. Complete la configuración del clúster restante y elija Crear clúster.

AWS CLI

Utilice el parámetro `--policy` al crear un clúster para adjuntar una política insertada:

```
aws dsql create-cluster --policy '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
        }
    }]
}'
```

AWS SDK

Python

```
import boto3
import json

client = boto3.client('dsql')

policy = {
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
        }
    }]
}
```

```
response = client.create_cluster(
    policy=json.dumps(policy)
)

print(f"Cluster created: {response['identifier']}")
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.CreateClusterRequest;
import software.amazon.awssdk.services.dsql.model.CreateClusterResponse;

DsqlClient client = DsqlClient.create();

String policy = """
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Principal": {"AWS": "*"},
    "Resource": "*",
    "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
    "Condition": {
      "StringNotEquals": { "aws:SourceVpc": "vpc-123456" }
    }
  }]
}
""";

CreateClusterRequest request = CreateClusterRequest.builder()
    .policy(policy)
    .build();

CreateClusterResponse response = client.createCluster(request);
System.out.println("Cluster created: " + response.identifier());
```

Agregación y edición de políticas basadas en recursos para clústeres

AWSConsola de administración de

Agregación de una política basada en recursos a un clúster existente

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. Elija el clúster de la lista de clústeres para abrir la página de detalles del clúster.
3. Elija la pestaña Permisos.
4. En la sección Política basada en recursos, elija Agregar política.
5. Ingrese el documento de la política en el editor JSON. Puede usar Editar instrucción o Agregar nueva instrucción para crear la política.
6. Elija Add Policy (Agregar política).

Edición de una política basada en recursos existente

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. Elija el clúster de la lista de clústeres para abrir la página de detalles del clúster.
3. Elija la pestaña Permisos.
4. En la sección Política basada en recursos, seleccione Editar.
5. Modifique el documento de la política en el editor JSON. Puede usar Editar instrucción o Agregar nueva instrucción para actualizar la política.
6. Seleccione Save changes (Guardar cambios).

AWS CLI

Use el comando `put-cluster-policy` para adjuntar una nueva política o actualizar una política existente en un clúster:

```
aws dsql put-cluster-policy --identifier your_cluster_id --policy '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
```

```
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "Null": { "aws:SourceVpc": "true" }
        }
    }]
}'
```

AWS SDK

Python

```
import boto3
import json

client = boto3.client('dsql')

policy = {
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Deny",
        "Principal": {"AWS": "*"},
        "Resource": "*",
        "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
        "Condition": {
            "Null": {"aws:SourceVpc": "true"}
        }
    }]
}

response = client.put_cluster_policy(
    identifier='your_cluster_id',
    policy=json.dumps(policy)
)
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.PutClusterPolicyRequest;
```

```
DsqlClient client = DsqlClient.create();

String policy = """
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Deny",
    "Principal": {"AWS": "*"},
    "Resource": "*",
    "Action": ["dsql:DbConnect", "dsql:DbConnectAdmin"],
    "Condition": {
      "Null": {"aws:SourceVpc": "true"}
    }
  }]
}
""";

PutClusterPolicyRequest request = PutClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .policy(policy)
    .build();

client.putClusterPolicy(request);
```

Visualización de políticas basadas en recursos

Puede ver las políticas basadas en recursos asociadas a los clústeres para comprender los controles de acceso vigentes actualmente.

AWSConsola de administración de

Visualización de políticas basadas en recursos

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. Elija el clúster de la lista de clústeres para abrir la página de detalles del clúster.
3. Elija la pestaña Permisos.
4. Consulte la política adjunta en la sección de política basada en recursos.

AWS CLI

Use el comando `get-cluster-policy` para ver la política basada en recursos de un clúster:

```
aws dsql get-cluster-policy --identifier your_cluster_id
```

AWS SDK

Python

```
import boto3
import json

client = boto3.client('dsql')

response = client.get_cluster_policy(
    identifier='your_cluster_id'
)

# Parse and pretty-print the policy
policy = json.loads(response['policy'])
print(json.dumps(policy, indent=2))
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.GetClusterPolicyRequest;
import software.amazon.awssdk.services.dsql.model.GetClusterPolicyResponse;

DsqlClient client = DsqlClient.create();

GetClusterPolicyRequest request = GetClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .build();

GetClusterPolicyResponse response = client.getClusterPolicy(request);
System.out.println("Policy: " + response.policy());
```

Eliminación de políticas basadas en recursos

Puede eliminar las políticas basadas en recursos de los clústeres para cambiar los controles de acceso.

> **Important**
>
> Cuando elimine todas políticas basadas en recursos de un clúster, el acceso lo controlarán por completo las políticas basadas en la identidad de IAM.

AWSConsola de administración de

Eliminación de una política basada en recursos

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. Elija el clúster de la lista de clústeres para abrir la página de detalles del clúster.
3. Elija la pestaña Permisos.
4. En la sección Política basada en recursos, elija Eliminar.
5. En el cuadro de diálogo de confirmación, ingrese **`confirm`** para confirmar la eliminación.
6. Elija Eliminar.

AWS CLI

Utilice el comando `delete-cluster-policy` para eliminar una política de un clúster:

```
aws dsql delete-cluster-policy --identifier your_cluster_id
```

AWS SDK

Python

```
import boto3

client = boto3.client('dsql')

response = client.delete_cluster_policy(
```

```
    identifier='your_cluster_id'
)

print("Policy deleted successfully")
```

Java

```
import software.amazon.awssdk.services.dsql.DsqlClient;
import software.amazon.awssdk.services.dsql.model.DeleteClusterPolicyRequest;

DsqlClient client = DsqlClient.create();

DeleteClusterPolicyRequest request = DeleteClusterPolicyRequest.builder()
    .identifier("your_cluster_id")
    .build();

client.deleteClusterPolicy(request);
System.out.println("Policy deleted successfully");
```

Ejemplos de políticas basadas en recursos comunes

Estos ejemplos muestran patrones comunes para controlar el acceso a los clústeres de Aurora DSQL. Puede combinar y modificar estos patrones para adaptarlos a sus requisitos de acceso específicos.

Bloqueo del acceso público a Internet

Esta política bloquea las conexiones a los clústeres de Aurora DSQL desde el Internet público (no VPC). La política no especifica desde qué VPC pueden conectarse los clientes, solo que deben conectarse desde una VPC. Para limitar el acceso a una VPC específica, use `aws:SourceVpc` con el operador de condición `StringEquals`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
```

```
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

> **Note**
>
> Este ejemplo solo usa `aws:SourceVpc` para comprobar las conexiones de VPC. Las claves de condición `aws:VpcSourceIp` y `aws:SourceVpce` proporcionan un mayor grado de detalle, pero no son necesarias para el control de acceso básico exclusivo de la VPC.

Para proporcionar una excepción para roles específicos, utilice esta política en su lugar:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAccessFromOutsideVPC",
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Condition": {
        "Null": {
          "aws:SourceVpc": "true"
        },
        "StringNotEquals": {
```

```
          "aws:PrincipalArn": [
            "arn:aws:iam::123456789012:role/ExceptionRole",
            "arn:aws:iam::123456789012:role/AnotherExceptionRole"
          ]
        }
      }
    }
  ]
}
```

Restricción del acceso a una organización de AWS

Esta política restringe el acceso a las entidades principales de una organización de AWS:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "dsql:DbConnect",
        "dsql:DbConnectAdmin"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/
mydsqlclusterid0123456789a",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgID": "o-exampleorgid"
        }
      }
    }
  ]
}
```

Restricción del acceso a una unidad organizativa específica

Esta política restringe el acceso a las entidades principales de una unidad organizativa (UO) específica de una organización de AWS, lo que proporciona un control más detallado que el acceso a toda la organización:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "dsql:DbConnect"
      ],
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/
mydsqlclusterid0123456789a",
      "Condition": {
        "StringNotLike": {
          "aws:PrincipalOrgPaths": "o-exampleorgid/r-examplerootid/ou-exampleouid/*"
        }
      }
    }
  ]
}
```

Políticas de clúster multirregionales

En el caso de los clústeres multirregionales, cada clúster regional tiene su propia política de recursos, lo que permite establecer controles específicos para cada región. A continuación, se muestra un ejemplo con diferentes políticas por región:

Política de us-east:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect"
      ],
```

```
      "Condition": {
        "StringNotEquals": {
          "aws:SourceVpc": "vpc-east1-id"
        },
        "Null": {
          "aws:SourceVpc": "true"
        }
      }
    }
  ]
}
```

Política de us-east:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Resource": "*",
      "Action": [
        "dsql:DbConnect"
      ],
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": "vpc-east2-id"
        }
      }
    }
  ]
}
```

> **Note**
>
> Las claves de contexto de condición pueden variar entre Regiones de AWS (como los ID de VPC).

Bloqueo de acceso público con políticas basadas en recursos en Aurora DSQL

El bloqueo de acceso público (BPA) es una característica que identifica y evita que se asocien políticas basadas en recursos que conceden acceso público a los clústeres de Aurora DSQL en las cuentas de AWS. Con BPA, puede impedir el acceso público a los recursos de Aurora DSQL. BPA realiza comprobaciones durante la creación o modificación de una política basada en recursos y ayuda a mejorar la seguridad con Aurora DSQL.

BPA utiliza un razonamiento automatizado para analizar el acceso que concede su política basada en recursos y le avisa si encuentra dichos permisos al administrar una política basada en recursos. El análisis verifica el acceso a todas las instrucciones de políticas basadas en recursos, las acciones y el conjunto de claves de condición que se utilizan en sus políticas.

> **Important**
>
> BPA ayuda a proteger los recursos al impedir que se conceda acceso público a través de las políticas basadas en recursos que se asocian directamente a los recursos de Aurora DSQL, como clústeres. Además de habilitar BPA, analice detenidamente las siguientes políticas para confirmar que no otorgan acceso público:
>
> - Políticas basadas en identidad asociadas a las entidades principales de AWS vinculadas (por ejemplo, los roles de IAM)
> - Políticas basadas en recursos adjuntas a recursos de AWS asociados (por ejemplo, claves de AWS Key Management Service [KMS])

Debe asegurarse de que la entidad principal no incluya una entrada * o que una de las claves de condición especificadas restrinja el acceso de las entidades principales al recurso. Si la política basada en recursos concede acceso público al clúster en las cuentas de AWS, Aurora DSQL le impedirá crear o modificar la política hasta que se corrija la especificación de la política y se considere no pública.

Para que una política no sea pública, especifique una o más entidades principales dentro del bloque `Principal`. El siguiente ejemplo de política basada en recursos bloquea el acceso público al especificar dos entidades principales.

```
{
```

```
  "Effect": "Allow",
  "Principal": {
    "AWS": [
      "123456789012",
      "111122223333"
    ]
  },
  "Action": "dsql:*",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id"
}
```

Las políticas que restringen el acceso al especificar determinadas claves de condición tampoco se consideran públicas. Junto con la evaluación de la entidad principal especificada en la política basada en recursos, se utilizan las siguientes claves de condición fiables para completar la evaluación de una política basada en recursos para el acceso no público:

- `aws:PrincipalAccount`
- `aws:PrincipalArn`
- `aws:PrincipalOrgID`
- `aws:PrincipalOrgPaths`
- `aws:SourceAccount`
- `aws:SourceArn`
- `aws:SourceVpc`
- `aws:SourceVpce`
- `aws:UserId`
- `aws:PrincipalServiceName`
- `aws:PrincipalServiceNamesList`
- `aws:PrincipalIsAWSService`
- `aws:Ec2InstanceSourceVpc`
- `aws:SourceOrgID`
- `aws:SourceOrgPaths`

Además, para que una política basada en recursos no sea pública, los valores del Nombre de recurso de Amazon (ARN) y las claves de cadena no deben contener comodines ni variables. Si

su política basada en recursos usa la clave `aws:PrincipalIsAWSService`, debe asegurarse de establecer el valor de la clave en true.

La siguiente política limita el acceso al usuario `Ben` de la cuenta especificada. La condición limita a la `Principal` y no se considera pública.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "*"
  },
  "Action": "dsql:*",
  "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id",
  "Condition": {
    "StringEquals": {
      "aws:PrincipalArn": "arn:aws:iam::123456789012:user/Ben"
    }
  }
}
```

El siguiente ejemplo de una política basada en recursos no pública restringe `sourceVPC` al usar el operador `StringEquals`.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "dsql:*",
      "Resource": "arn:aws:dsql:us-east-1:123456789012:cluster/cluster-id",
      "Condition": {
        "StringEquals": {
          "aws:SourceVpc": [
            "vpc-91237329"
          ]
        }
      }
    }
  ]
```

```
}
```

Operaciones de la API de Aurora DSQL y políticas basadas en recursos

Las políticas basadas en recursos de Aurora DSQL controlan el acceso a operaciones de API específicas. En las siguientes secciones se muestran todas las operaciones de la API de Aurora DSQL organizadas por categoría, con una indicación de cuáles admiten políticas basadas en recursos.

La columna Admite RBP indica si la operación de la API está sujeta a una evaluación de políticas basada en los recursos cuando se adjunta una política al clúster.

API de etiquetas

Operación de API	Descripción	Admite RBP
ListTagsForResource	Muestra las etiquetas de un recurso de Aurora DSQL	Sí
TagResource	Agrega etiquetas a un recurso de Aurora DSQL	Sí
UntagResource	Elimina etiquetas de un recurso de Aurora DSQL	Sí

API de administración de clústeres

Operación de API	Descripción	Admite RBP
CreateCluster	Crea un nuevo clúster.	No
DeleteCluster	Elimina un clúster	Sí
GetCluster	Recupera información sobre un clúster	Sí
GetVpcEndpointServiceName	Recupera el nombre de servicio del punto de conexión de VPC para un clúster	Sí
ListClusters	Muestra los clústeres de la cuenta	No

Operación de API	Descripción	Admite RBP
UpdateCluster	Actualiza la configuración de un clúster	Sí

API de propiedad multirregional

Operación de API	Descripción	Admite RBP
AddPeerCluster	Agrega un clúster de emparejamiento a una configuración multirregional	Sí
PutMultiRegionProperties	Establece las propiedades multirregionales de un clúster	Sí
PutWitnessRegion	Establece la región testigo de un clúster multirregional	Sí

API de política basada en recursos

Operación de API	Descripción	Admite RBP
DeleteClusterPolicy	Elimina la política basada en recursos de un clúster	Sí
GetClusterPolicy	Recupera la política basada en recursos de un clúster	Sí
PutClusterPolicy	Crea o actualiza la política basada en recursos de un clúster	Sí

AWS Fault Injection ServiceLas API de

Operación de API	Descripción	Admite RBP
InjectError	Inyecta errores en las pruebas de inyección de errores	No

API de copia de seguridad y restauración

Operación de API	Descripción	Admite RBP
GetBackupJob	Recupera información sobre un trabajo de copia de seguridad	No
GetRestoreJob	Recupera información sobre un trabajo de restauración	No
StartBackupJob	Inicia un trabajo de copia de seguridad para un clúster	Sí
StartRestoreJob	Inicia un trabajo de restauración a partir de una copia de seguridad	No
StopBackupJob	Detiene un trabajo de copia de seguridad en ejecución	No
StopRestoreJob	Detiene un trabajo de restauración en ejecución	No

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 a servicios, 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 al servicio 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:
>
> JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Action": ["dsql:CreateCluster"],
            "Resource": [
                "arn:aws:dsql:us-east-1:*:cluster/*",
                "arn:aws:dsql:us-east-2:*:cluster/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

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.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dsql:CreateCluster",
                "dsql:PutMultiRegionProperties",
                "dsql:AddPeerCluster"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dsql:PutWitnessRegion"
            ],
            "Resource": "arn:aws:dsql:*:123456789012:cluster/*",
            "Condition": {
                "StringEquals": {
                    "dsql:WitnessRegion": [
                        "us-west-2"
                    ]
                }
            }
        }
    ]
}
```

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
- AWS Prácticas recomendadas para la seguridad, la identidad y el cumplimiento de
- 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 alcance de programas de cumplimiento específicos, se consulta Servicios de AWS incluidos por programa de cumplimiento y se elige el programa de cumplimiento que interese. Para obtener información general, consulte Programas de conformidad de AWS.

Puede descargar los informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte Descarga de informes en AWS Artifact.

Su responsabilidad de conformidad al utilizar Servicios de AWS se determina en función de la confidencialidad de los datos, los objetivos de conformidad de la empresa, así como de la legislación y los reglamentos aplicables. Para obtener más información sobre la responsabilidad de cumplimiento al usar Servicios de AWS, consulte la Documentación de seguridad de AWS.

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. Si la región testigo deja de funcionar correctamente, la disponibilidad del clúster no se ve afectada. Las transacciones de escritura podrían experimentar un ligero aumento de la latencia hasta que la región testigo se recupere.

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.

Pruebas de inyección de errores

Amazon Aurora DSQL se integra con AWS Fault Injection Service (AWS FIS), un servicio totalmente administrado para ejecutar experimentos de inyección de errores controlados a fin de mejorar la resiliencia de una aplicación. Con AWS FIS, puede:

- Creación de plantillas de experimentos que definan escenarios de error específicos
- Inserción de errores (tasas elevadas de errores de conexión al clúster) para validar los mecanismos de gestión y recuperación de errores de las aplicaciones
- Prueba del comportamiento de las aplicaciones multirregionales para validar el cambio de tráfico de una aplicación entre Regiones de AWS cuando una Región de AWS está experimentando altas tasas de error de conexión

Por ejemplo, en un clúster multirregional que abarca Este de EE. UU. (Norte de Virginia) y Este de EE. UU. (Ohio), puede ejecutar un experimento en Este de EE. UU. (Ohio) para probar errores allí, mientras que Este de EE. UU. (Norte de Virginia) continúa con las operaciones normales. Estas pruebas controladas le ayudan a identificar y resolver posibles problemas antes de que afecten a las cargas de trabajo de producción.

Consulte los Objetivos de acción en la Guía del usuario de AWS FIS para obtener una lista completa de las acciones de AWS FIS compatibles.

Para obtener información sobre las acciones de Amazon Aurora DSQL disponibles en AWS FIS, consulte la Referencia sobre las acciones de Aurora DSQL en la Guía del usuario de AWS FIS.

Para empezar a realizar experimentos de inyección de errores, consulte Planificación de los experimentos de AWS FIS en la Guía del usuario de AWS FIS.

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 con 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 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 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.dsql.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-xxxxxxxxxxxxxxxxx \
    --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
```

Respuesta de ejemplo

```
{
    "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);
```

Configuración adicional al conectarse mediante Direct Connect o el emparejamiento de Amazon VPC

Puede que se requiera una configuración adicional para conectarse a los clústeres de Aurora DSQL mediante un punto de conexión de AWS PrivateLink desde dispositivos en las instalaciones a través del emparejamiento de Amazon VPC o Direct Connect. Esta configuración no es necesaria si su aplicación se ejecuta en la misma instancia de Amazon VPC que su punto de conexión de AWS PrivateLink. Las entradas de DNS privado creadas anteriormente no se resolverán de forma correcta fuera de la instancia de Amazon VPC del punto de conexión, pero puede crear sus propios registros DNS privados que se resuelvan en su punto de conexión de AWS PrivateLink.

Cree un registro DNS de CNAME privado que apunte al nombre de dominio completo del punto de conexión de AWS PrivateLink. El nombre de dominio del registro DNS creado debe construirse a partir de los siguientes componentes:

1. El identificador del servicio a partir del nombre del servicio. Por ejemplo: .: `dsql-fnh4`
2. la Región de AWS,

Cree el registro DNS de CNAME con un nombre de dominio que tenga el siguiente formato: `*.service-identifier.region.on.aws`

El formato del nombre de dominio es importante por dos motivos:

1. El nombre de host utilizado para conectarse a Aurora DSQL debe coincidir con el certificado de servidor de Aurora DSQL cuando se utilice el modo SSL `verify-full`. Esto garantiza el máximo nivel de seguridad de la conexión.
2. Aurora DSQL utiliza la parte del ID de clúster del nombre de host que se utiliza para conectarse a Aurora DSQL a fin de identificar el clúster que se conecta.

Si no es posible crear registros DNS privados, puede seguir conectándose a Aurora DSQL. Consulte Conexión a un clúster de Aurora DSQL mediante un punto de conexión de AWS PrivateLink sin DNS privado..

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. Por ejemplo: .: `us-east-1`

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
```

Conexión a un clúster de Aurora DSQL mediante un punto de conexión de AWS PrivateLink sin DNS privado.

Las instrucciones de conexión anteriores se basan en registros DNS privados. Si la aplicación se ejecuta en la misma instancia de Amazon VPC que el punto de conexión de AWS PrivateLink, los registros DNS se crean de forma automática. Como alternativa, si se conecta desde dispositivos en las instalaciones a través del emparejamiento de Amazon VPC o Direct Connect, podrá crear sus propios registros DNS privados. Sin embargo, la configuración de los registros DNS no es siempre posible debido a las restricciones de red impuestas por sus equipos de seguridad. Si la aplicación debe conectarse mediante Direct Connect o desde una instancia de Amazon VPC emparejada, y la configuración de registros DNS no es posible, puede seguir conectándose a Aurora DSQL.

Aurora DSQL utiliza la parte del ID de clúster del nombre de host para identificar el clúster que se conecta, pero si la configuración de registros DNS no es posible, Aurora DSQL permite especificar el clúster de destino mediante la opción de conexión `amzn-cluster-id`. Con esta opción, es posible utilizar el nombre de dominio completo del punto de conexión de AWS PrivateLink como nombre de host al conectarse.

> **Important**
>
> Al conectarse con la dirección IP o el nombre de dominio completo del punto de conexión de AWS PrivateLink, no se admite el modo SSL `verify-full`. Por este motivo, se prefiere configurar un DNS privado.

Ejemplo: Especificación de la opción de conexión de ID de clúster mediante PostgreSQL

```
# Set environment variables
export CLUSTERID=your-cluster-id
export REGION=us-east-1
export HOSTNAME=vpce-04037adb76c111221-d849uc2p.dsql-fnh4.us-east-1.vpce.amazonaws.com
 # This should match your endpoint's fully-qualified domain name

# Construct the hostname used to generate the authentication token
export AUTH_HOSTNAME="$CLUSTERID.dsql.$REGION.on.aws"

# Generate authentication token
export PGPASSWORD=$(aws dsql --region $REGION generate-db-connect-admin-auth-token --
hostname $AUTH_HOSTNAME)
```

```
# Specify the amzn-cluster-id connection option
export PGOPTIONS="-c amzn-cluster-id=$CLUSTERID"

# 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 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.
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
- AWS PrivateLink documentación
- 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:

- 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 puede 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:dsql:*: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": "backup.amazonaws.com"
    },
    "Action": "dsql:GetCluster",
    "Resource": [
      "arn:aws:dsql:*:123456789012:cluster/*"
    ],
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:backup:*: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 Consola de administración de AWS, 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.

- Los cálculos del límite de almacenamiento pueden tardar un tiempo en reflejar el almacenamiento liberado después de ejecutar un comando `DROP TABLE`. Si necesita capacidad de almacenamiento adicional, consulte Cuotas de clúster para solicitar actualizaciones de la cuota.
- Para tablas grandes en Aurora DSQL, utilice el catálogo del sistema para recuperar el recuento de filas de la tabla en lugar de operaciones `COUNT(*)`. Para obtener más información, consulte Uso de tablas y comandos del sistema en Aurora DSQL.
- Aurora DSQL administra los permisos a través de concesiones a nivel de esquema. Los usuarios administradores crean esquemas utilizando `CREATE SCHEMA` y conceden acceso a otros roles utilizando `GRANT USAGE ON SCHEMA`. Los usuarios administradores gestionan objetos en el esquema público, mientras que los usuarios no administradores crean objetos en esquemas creados por los usuarios. El rol de administrador puede otorgarse a sí mismo cualquier otro rol para obtener permisos sobre los objetos creados por los usuarios. Para obtener más información, consulte Autorización de roles de base de datos para utilizar SQL en la base de datos.
- Cuando los controladores llaman a `PG_PREPARED_STATEMENTS`, Aurora DSQL proporciona una vista de todo el clúster de las instrucciones preparadas en caché. Es posible que vea más instrucciones preparadas por conexión de lo esperado para el mismo clúster y el mismo rol de IAM. Aurora DSQL administra los nombres de las instrucciones de forma dinámica durante la preparación.
- Cuando se conecte desde instancias de solo IPv4, asegúrese de que su cliente esté configurado para conexiones IPv4. Algunos clientes de PostgreSQL intentan conexiones IPv4 e IPv6 en modo de pila doble. Si la conexión IPv4 sufre limitaciones, el cliente puede intentar IPv6 y devolver un error `NetworkUnreachable` en hosts de solo IPv4. Configure su cliente para que utilice IPv4 explícitamente y así evitar este comportamiento.
- Después de que un usuario administrador crea un nuevo esquema, los cambios `GRANT` y `REVOKE` se propagan a las conexiones existentes dentro del tiempo de vida de la conexión (hasta una hora). Para que surta efecto inmediato, establezca una nueva conexión después de los cambios de permiso.

- En casos excepcionales de recuperación de clústeres vinculados entre varias regiones, las operaciones de recuperación automática de clústeres mantienen una alta disponibilidad, pero es posible que se produzcan errores transitorios de control de simultaneidad o de conexión. En la mayoría de los casos, solo se ve afectado un porcentaje de la carga de trabajo. Cuando encuentre estos errores transitorios, vuelva a intentar la transacción o vuelva a conectarse con su cliente.
- Algunos clientes de SQL, como Datagrip, solicitan metadatos extensos del sistema para completar la información del esquema. Aurora DSQL proporciona metadatos básicos para la funcionalidad de consultas SQL. La visualización del esquema en estos clientes puede mostrar información limitada en comparación con su conjunto completo de características.
- Para garantizar que las consultas reconozcan los esquemas y tablas recién creados, actualice su conexión después de crear o eliminar objetos de la base de datos. Esto incluye situaciones en las que aparecen los errores `Schema Already Exists` después de eliminar un esquema o al consultar objetos creados en otra conexión. Desconecte y vuelva a conectarse, o vuelva a ejecutar `SET search_path` para actualizar la caché del catálogo.
- Para consultas complejas, utilice `EXPLAIN ANALYZE VERBOSE` para identificar operaciones de alta latencia y optimizar los planes de consulta. Los índices de cobertura pueden reducir significativamente los costos de DPU al permitir escaneos solo de índices en lugar de escaneos completos de tablas. Para obtener más información, consulte Trabajo con planes EXPLAIN de Aurora DSQL.
- Los límites de conexión se administran a nivel del clúster. Consulte Cuotas de clúster para solicitar actualizaciones de cuotas.

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 256 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 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`
Cantidad máxima de secuencias	5 000	No	`54000`	`ERROR: creating more than 500 not allowed`

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 Consola de administración de AWS 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 o no se puede aceptar la conexión, no se recibió sni

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.

Asignación incorrecta entre usuario y acción

Este error se produce cuando el tipo de token de autenticación no coincide con el rol de la base de datos. Aurora DSQL utiliza dos tipos de tokens: `DbConnectAdmin` para el rol de `admin` y `DbConnect` para roles de base de datos personalizados.

- Si ve `Wrong user to action mapping. user: admin, action: DbConnect`, use `generate-db-connect-admin-auth-token` en lugar de `generate-db-connect-auth-token`.
- Si ve `Wrong user to action mapping. user: myusername, action: DbConnectAdmin`, use `generate-db-connect-auth-token` en lugar de `generate-db-connect-admin-auth-token`.

Solución de problemas de errores de autorización

Role <role> not supported

Aurora DSQL no admite la operación `GRANT`. Consulte Subconjuntos de comandos SQL admitidos en Aurora DSQL.

Cannot establish trust with role <role>

Aurora DSQL no admite la operación `GRANT`. Consulte Subconjuntos de comandos SQL 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: 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”

Esta transacción intentó modificar las mismas tuplas que otra transacción simultánea. Esto indica que hay contención en las tuplas modificadas. Para obtener más información, consulte Control de simultaneidad en Aurora DSQL

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.

Aportación de comentarios sobre Amazon Aurora DSQL

Si encuentra características que sean fundamentales para su migración pero que actualmente no sean compatibles con Aurora DSQL, AWS ofrece varios canales para recibir comentarios:

Canales de comentarios

Servidor Discord de Aurora DSQL

: Únase al servidor Discord de Aurora DSQL para conectar con el equipo y la comunidad de AWS. Comparta solicitudes de características, analice los desafíos de la migración y reciba comentarios en tiempo real.

AWS Support

: Si tiene un plan de AWS Support, cree un caso de soporte para analizar sus necesidades de tiempo y sus requisitos específicos.

AWS re:Post

: Utilice AWS re:Post para hacer preguntas y compartir comentarios con la comunidad y los expertos de AWS.

Solicitudes de características efectivas

Cuando solicite características, proporcione:

- Descripción del caso de uso: explique qué intenta lograr y por qué.
- Solución alternativa actual: describa las alternativas que haya probado.
- Impacto empresarial: explique cómo la característica que falta afecta a la cronología de su migración o a la funcionalidad de la aplicación.
- Nivel de prioridad: indique si esto bloquea su migración o si sería una mejora deseable.

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
Contenido nuevo: Conector de Aurora DSQL para .NET Npgsql	Se ha añadido documentación del conector de Aurora DSQL para .NET Npgsql, que envuelve Npgsql con autenticación de IAM automática. El conector se encarga de la generación de tokens, la configuración SSL y la gestión de conexiones para las aplicaciones .NET. Para obtener más información, consulte Conexión a clústeres de Aurora DSQL con un conector de .NET Npgsql.	20 de marzo de 2026
Contenido actualizado: Referencia de comandos SQL y consultas del sistema	Se han añadido START TRANSACTION y ROLLBACK a la referencia de comandos de control de transacciones, con END y ABORT como alias. Se han añadido consultas de sistema útiles para obtener información sobre la versión de Aurora DSQL y PostgreSQL. Para obtener más información, consulte Referencia de	13 de marzo de 2026

	compatibilidad con PostgreSQL.	
Contenido actualizado: Carga de datos en Aurora DSQL	Se ha actualizado la guía de carga de datos con información sobre el uso de `\copy` en el lado del cliente, las prácticas recomendadas para la instrucción INSERT y consejos para crear las tablas previamente antes de la carga. Para obtener más información, consulte Carga de datos en Aurora DSQL.	13 de marzo de 2026
Contenido nuevo: Conector de Aurora DSQL para Ruby pg	Se ha añadido documentación del conector de Aurora DSQL para Ruby pg, que envuelve pg gem con autenticación de IAM automática. El conector se encarga de la generación de tokens, la configuración SSL y la gestión de conexiones para las aplicaciones Ruby. Para obtener más información, consulte Conexión a clústeres de Aurora DSQL con un conector de Ruby pg.	12 de marzo de 2026

Contenido actualizado: Trabajos DDL asíncronos	Se ha actualizado la documentación de `sys.jobs` con información más detallada sobre la supervisión y la administración de operaciones DDL asíncronas. Para obtener más información, consulte Referencia de compatibilidad con PostgreSQL.	6 de marzo de 2026
Contenido actualizado: Generación de token de autenticación PHP	Se ha añadido una pestaña de SDK de PHP a la página de generación de tokens de autenticación. Para obtener más información, consulte Generación de tokens de autenticación.	5 de marzo de 2026
Contenido nuevo: Carga de datos en Aurora DSQL	Se ha añadido una guía para cargar datos en clústeres de Aurora DSQL, que incluye el uso de la utilidad Aurora DSQL Loader. Para obtener más información, consulte Carga de datos en Aurora DSQL.	5 de marzo de 2026

Contenido nuevo: Secuencias y columnas de identidad	Se ha añadido compatibilidad con secuencias y columnas de identidad. Nuevas páginas de referencia de comandos SQL para CREATE SEQUENCE, ALTER SEQUENCE, DROP SEQUENCE y funciones de manipulación de secuencias. Se han actualizado CREATE TABLE y ALTER TABLE para incluir la sintaxis de la columna de identidad. Se ha añadido una nueva guía para elegir los tipos de identificadores y los tamaños de caché. Para obtener más información, consulte Secuencias y columnas de identidad.	11 de febrero de 2026
Contenido nuevo: Conector de Aurora DSQL para Go	Se ha añadido documentación del conector de Aurora DSQL para Go, que envuelve pgx con autenticación IAM automática. El conector se encarga de la generación de tokens, la configuración SSL y la gestión de conexiones para las aplicaciones Go. Para obtener más información, consulte Conexión a clústeres de Aurora DSQL con un conector de Go.	5 de febrero de 2026

Contenido actualizado: Herramientas de conectividad de clústeres de Amazon Aurora DSQL	Se ha reorganizado la documentación de las herramientas de conectividad de clúster para aclarar la distinción entre conectores, adaptadores y herramientas de terceros proporcionados por AWS. Se han añadido los enlaces que faltaban a los ejemplos de código. Para obtener información, consulte Herramientas de conectividad de clústeres de Amazon Aurora DSQL.	26 de enero de 2026
Contenido nuevo: Complemento Aurora DSQL para DBeaver Community Edition	Se ha añadido documentación para el complemento Aurora DSQL para DBeaver Community Edition, que permite la autenticación IAM y simplifica la configuración de la conexión para los clústeres de Aurora DSQL. Incluye instrucciones de instalación, configuraciones de conexiones y guía para la solución de problemas. Para obtener más información, consulte Uso de DBeaver para acceder a Aurora DSQL.	26 de enero de 2026

Contenido nuevo: Controlador de Aurora DSQL para SQLTools	Se ha añadido documentación para el controlador de Aurora DSQL para SQLTools, una extensión de Visual Studio Code que permite a los desarrolladores conectarse y consultar bases de datos de Aurora DSQL directamente desde VS Code con autenticación automática de IAM. Para obtener más información, consulte Uso del controlador de Aurora DSQL para SQLTools.	26 de enero de 2026
Contenido actualizado: Dirección de Aurora DSQL: Skills y Powers	Se ha añadido documentación para respaldar la instalación mediante la CLI de Skills para obtener asistencia independiente del agente. La CLI de Skills proporciona un método de configuración simplificado que funciona con varios asistentes de codificación de IA, incluidos Claude Code, Cursor, Copilot, Gemini y otros. Para obtener más información, consulte Dirección DSQL de Aurora: Skills y Powers.	23 de enero de 2026

Contenido nuevo: Adaptador de Aurora DSQL para Tortoise ORM	Se ha añadido compatibilidad con Tortoise ORM, un marco ORM asíncrono de Python. El adaptador Aurora DSQL para Tortoise ORM permite a los desarrolladores utilizar Tortoise ORM con clústeres de Aurora DSQL. Para obtener más información, consulte Adaptadores y dialectos de Aurora DSQL.	23 de enero de 2026
Contenido nuevo: Dirección de Aurora DSQL: Skills y Powers	Se ha añadido nueva documentación para configura r la dirección de IA con Aurora DSQL utilizando Skills y Powers. Incluye instrucciones de configuración para Kiro Powers, Claude Skills, Gemini Skills y Codex Skills. Para obtener más información, consulte Dirección DSQL de Aurora: Skills y Powers.	16 de enero de 2026
Compatibilidad con índices de tipos de datos numéricos	Se ha añadido compatibi lidad con los tipos de datos `numeric` en Aurora DSQL. Ahora puede utilizar las columnas `numeric` como claves principales y en los índices secundarios. Para obtener más información, consulte Tipos de datos admitidos en Aurora DSQL.	13 de enero de 2026

Contenido actualizado: Subconjuntos de comandos SQL admitidos	Se ha reorganizado la documentación de los comandos SQL en páginas separadas para mejorar la navegación y la claridad. Cada comando (CREATE TABLE, ALTER TABLE, CREATE VIEW, ALTER VIEW, DROP VIEW) tiene ahora su propia página específica. Para obtener más información, consulte Subconjuntos de comandos SQL admitidos.	6 de enero de 2026
Contenido actualizado: Servidor MCP de Aurora DSQL de AWS Labs	Se ha actualizado la documentación del servidor MCP con métodos de instalación detallados para Claude Code y Codex, incluyendo ejemplos de configuración basada en CLI y archivos de configuración. Se ha añadido una guía completa para encontrar los archivos de configuración del cliente de MCP en diferentes herramientas de desarrollo. Para obtener más información, consulte Servidor MCP de Aurora DSQL de AWS.	19 de diciembre de 2025

Contenido actualizado: Migración de PostgreSQL a Aurora DSQL	Se ha reescrito la sección sobre compatibilidad con PostgreSQL para convertir la en una guía de migración completa. Incluye información sobre la compatibilidad de los marcos, patrones de migración comunes, diferencias arquitectónicas y orientación sobre migración asistida por IA. Se ha añadido un nuevo capítulo para enviar comentarios sobre Aurora DSQL. Para obtener más información, consulte Migración de PostgreSQL a Aurora DSQL.	16 de diciembre de 2025
Contenido actualizado: Conexión a Aurora DSQL utilizando AWS PrivateLink	Se ha añadido documentación sobre la configuración de DNS privado y la opción de conexión de ID de clúster para dar soporte a los clientes que utilizan AWS PrivateLink con Direct Connect o la interconexión de Amazon VPC. Incluye instrucciones para conectarse sin DNS privado utilizando la opción de conexión `amzn-cluster-id` . Para obtener más información, consulte Administración y conexión a clústeres de Aurora DSQL mediante AWS PrivateLink.	11 de diciembre de 2025

Contenido actualizado: Ciclo de vida del clúster de Aurora SQL	Documentación actualizada para la administración del ciclo de vida del clúster de Aurora DSQL. Explica las definiciones del estado del clúster, las transiciones de estado y las operaciones disponibles durante los estados sin uso e inactivo. Para obtener más información, consulte Ciclo de vida del clúster de Aurora DSQL.	4 de diciembre de 2025
Contenido nuevo: Conectores de Aurora DSQL para Python y Node.js	Se ha añadido documentación para los conectores de Aurora DSQL para Python (psycopg, psycopg2, asyncpg) y Node.js (node-postgres, Postgres.js). Estos conectores integran la autenticación de IAM para conectar aplicaciones a clústeres de Aurora DSQL. Para obtener más información, consulte Conectores de Aurora DSQL.	21 de noviembre de 2025

Contenido nuevo: Uso de JupyterLab con Aurora DSQL	Se ha añadido una guía paso a paso para conectar y consultar Aurora DSQL utilizando JupyterLab con Python. Incluye instrucciones tanto para instalaciones locales de JupyterLab como para entornos de Amazon SageMaker AI. Para obtener más información, consulte Uso de JupyterLab con Aurora DSQL.	20 de noviembre de 2025
Contenido actualizado: Cuotas de Aurora DSQL	Se ha actualizado la cuota máxima de almacenamiento del clúster de 128 TiB a 256 TiB. Para obtener más información, consulte Cuotas para Aurora DSQL.	19 de noviembre de 2025
Nuevo contenido: Introducción al editor de consultas de Aurora DSQL	Se ha añadido documentación sobre el uso del editor de consultas de Aurora DSQL en la consola de administración de AWS. Incluye requisitos previos, configuración de la conexión e instrucciones para la ejecución de consultas. Para obtener más información, consulte Introducción al editor de consultas de Aurora DSQL.	18 de noviembre de 2025

Compatibilidad de la política basada en recursos para Amazon Aurora DSQL	Se ha agregado compatibilidad con políticas basadas en recursos (RBP) con nuevos permisos: `PutClusterPolicy` , `GetClusterPolicy` y `DeleteClusterPolicy` . Estos permisos permiten administrar políticas insertadas en los clústeres de Aurora DSQL para el control de acceso detallado. Se han actualizado las políticas administradas AmazonAuroraDSQLFullAccess, AmazonAuroraDSQLReadOnlyAccess y AmazonAuroraDSQLConsoleFullAccess para incluir capacidades de RBP. Para obtener más información, consulte Políticas administradas de AWS para Amazon Aurora DSQL.	15 de octubre de 2025
Conector de JDBC de Aurora DSQL	Se ha añadido documentación del conector de JDBC de Aurora DSQL, un conector de PgJDBC que integra la autenticación de IAM para conectar aplicaciones Java a clústeres de Amazon Aurora DSQL. Para obtener más información, consulte Conexión a clústeres de Aurora DSQL con un conector de JDBC.	2 de septiembre de 2025

Actualizaciones de políticas administradas de AWS para la integración de AWS FIS	Las políticas de `AmazonAuroraDSQLFullAccess` y `AmazonAuroraDSQLConsoleFullAccess` se han actualizado para apoyar la integración de AWS Fault Injection Service con Aurora DSQL. Esto le permite inyectar errores en los clústeres de Aurora DSQL de una región y mutirregionales para probar la tolerancia a errores de las aplicaciones. Para obtener más información sobre estas políticas, consulte Actualizaciones de políticas administradas de AWS.	19 de agosto de 2025
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 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 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 al servicio 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